A Smart-Decision System for Realtime Mobile AR Applications

Siqi Huang, Tao Han, and Jiang Xie
Department of Electrical and Computer Engineering
University of North Carolina at Charlotte, Charlotte, NC 28223, U.S.A.
E-mail: shuang9@uncc.edu; tao.han@uncc.edu; linda.xie@uncc.edu

***Abstract*—With the development of the hardware and software platforms, we can implement the deep learning model on the mobile device for mobile augmented reality (AR) applications. However, not all mobile AR tasks can be finished on mobile devices. Meanwhile, the limited computation resources on mobile devices are still the main obstacle to achieve realtime mobile AR applications. In this paper, we proposed a smart-decision framework which combines the advantages of the on-device mobile AR system and the edge-based mobile AR system to achieve real-time object recognition. High computation complexity tasks will be offloaded to the edge servers. Low complexity tasks will be executed on mobile devices or the edge server depending on the network latency. To overcome the dynamic changes of network condition and the limitations of the on-device deep learning models, we design a cache and matching algorithm on the mobile devices to enhance the performance of the recognition tasks. With our proposed system, the quality of the mobile AR application is improved. The performance of the smart-decision framework is validated through experiments with a testbed[1].**

I. Introduction

Mobile augmented reality (AR) applications attract more and more attention in recent years. Companies are developing AR platforms and devices such as Google ARCore and Microsoft HoloLens to encourage developers to create mobile AR applications. We can see an increasing market size of AR applications. However, one of the most important challenges is the limited resources on the mobile devices[2], such as CPU, GPU, memory, battery, and storage.

Mobile AR applications need a large amount of computation resources to support the scene analysis and interactions with users. For instance, in a smart shopping application [1], the mobile device needs to recognize the product, including its shape, color, and brand. Another example is the virtual tourism application like HoloTour [2]. HoloLens will recognize the gesture or voice commands so that a user can interact with the virtual objects. To recognize the objects and human actions, advanced computer vision algorithms will be deployed. Running these algorithms requires high complexity computation, and a large amount of computation resources are required.

Existing mobile AR systems can be mainly classified into two categories. The first one is the cloud/edge-based mobile AR system [3]–[6]. They choose to offload recognition tasks to powerful servers. As shown in Fig. 1, the mobile device is used to capture the scenes and visualize the results. The captured data will be uploaded to the servers including both edge and cloud servers. The servers will provide perform image analysis using computer vision algorithms. The main advantage of cloud/edge-based mobile AR systems is that we can apply the advanced recognition algorithms with high precision on the servers. However, current cloud/edge-based mobile AR system cannot achieve fast response which is a vital feature of mobile AR applications. The reason is that offloading will cause high network latency.

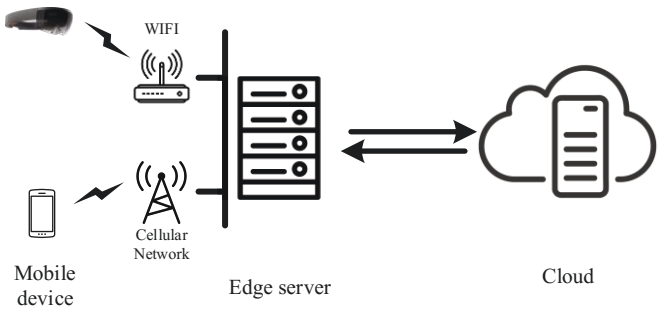

Fig. 1. The cloud/edge-based mobile AR system.

The second one is the on-device mobile AR system [7]–[9]. Before we are able to implement the deep learning based algorithms, the recognition tasks are achieved with low complexity methods such as image feature extraction, classification, and object tracking. The computation complexity of these methods is lower enough to be supported by mobile devices. However, with the development of the hardware (GPU on mobile devices) and software (Google MobileNets and Tensorflow Lite), the simple deep learning based algorithms can be directly deployed on the mobile devices for some recognition tasks such as the classification and object detection. The advantage of the on-device architecture is that we can do the recognition tasks on the device. There is no network latency involved. However, there are several challenges to apply the on-device models. As shown in Fig. 2. The first one is that not all the deep learning models can be implemented on mobile devices for mobile AR applications. The low complexity tasks such as image recognition, object detection, and landmark recognition can be implemented on mobile devices. The high complexity algorithms such as the object segmentation, 3D object detec-

[1]This work was supported in part by the US National Science Foundation (NSF) under Grant No. 1718666, 1731675, 1910667, and 1910891.

[2]In this paper, mobile devices include smart phones and wearable equipment.

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

tion, and image semantic analysis can only be supported by the edge server and cloud. Secondly, the recognition performance is not robust. For instance, the on-device models produce wrong classification and detection results very frequently. We can only get satisfied recognition results in some specific scenes and applications.

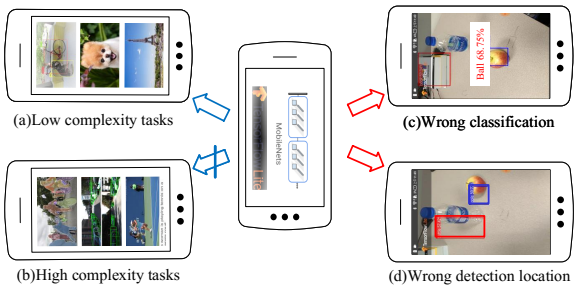

Fig. 2. The mobile AR applications challenges.

The question that we investigate in this paper is: How can we achieve real-time recognition with high quality for mobile AR applications? The performances of recognition tasks are very sensitive to the latency because of the dynamic movements of the objects and users. The round-trip time of mobile AR applications is defined as the duration between the scene captured and the recognition results received. For cloud/edge-based system, the data size of the recognition results is very small (less than 1KB). The latency of cloud/edge-based mobile AR system is mainly composed of two parts: the time for uploading the image to the remote server and time for the vision analysis. With the help of the edge server, users can use the basic recognition services which are deployed very close to them. As a result, the network latency is reduced by avoiding transmitting images to the remote cloud. With the help of powerful GPU on the edge server, the speed of recognition task execution can easily be faster than 24 FPS. For on device mobile AR applications, the speed of deep learning based algorithms on the mobile device is about 200ms per frame.

In this paper, we proposed a smart-decision framework which combines the advantages of the on-device mobile AR system and the edge-based system to achieve real-time recognition tasks. The framework will decide where to execute the recognition tasks. To overcome the dynamic changes of network condition and the limitations of the on-device deep learning models, we design a cache and matching algorithm on the mobile devices. Compared with other related works, we focus on enhancing the performance of mobile AR applications. With our proposed system, the quality of the mobile AR applications is improved. The rest of the article is organized as follows. Section 2 shows some background and related work. Section 3 presents the overview of the smart-decision framework. Section 4 presents the design of the cache and matching algorithm. Section 5 shows the performance evaluation. We conclude the paper in Section 6.

II. BACKGROUND AND RELATED WORKS

A. Image features and object tracking

Image Features are numerical descriptors that individually or collectively form unique signatures to describe the image. Typically, image recognition techniques rely on discovering multiple interesting points in the image and extracting descriptors for these points. These interesting points collectively form a signature for the image. This technique is used by many feature detection and extraction techniques such as SIFT [10], SURF [11] and ORB [12]. **Object tracking** is widely used in Mobile AR/VR systems [3], [4], [6] before the invention of deep learning algorithms. The advantage of adapting object tracking is that once you recognize the objects, you don't need to recognize again if you can keep tracking the objects based on their unique features. In this case, a lot of redundant computations are void and using the tracking algorithm is fast enough to achieve real-time processing.

B. Recognition tasks

The **recognition** is vital for mobile AR applications. Image recognition is used for identifying the objects in the image and analyzing the semantic information(like human actions) in the image. It can help the device to understand what you see and what you want to do, which is the first step to allow the interaction with the environments. Recognition can be divided into several small tasks, such as object detection, object segmentation, semantic analysis and so on. State-of-the-art recognition algorithms [13]–[15] can achieve high-quality results, and they all use the deep convolutional neural networks (CNN). Unfortunately, these deep and complicated networks have significant computation and memory needs. In this case, deploying them on mobile devices is very challenging. For instance, SSD300 and Faster-RCNN300 need 34.9 and 64.3 billions of Multi-Add calculations; 33.1 and 138.5 millions of parameters need to be calculated and maintained, respectively.

In computer vision field, there are some works which try to build light deep neural networks to balance the latency and accuracy of the detection and recognition models. For instance, Howard *et al.* [16] presents an efficient model-MobileNets for mobile and embedded vision applications. Although it reduces about 80% computation complexity of current object detection models, it still needs process millions of parameters, which cause billions of Muti-Adds calculations. Apte *et al.* [17] deploys the tiny-YOLO model on iPhone 7 to achieve the real-time object detection. It can achieve 8-11 FPS, but the accuracy is low. With the invention of Tensorflow Lite, some recognition algorithms can be deployed on the mobile device with high accuracy. The speed of running the object detection task on the android mobiles phones is about 5-6 FPS.

C. Mobile AR systems

There are a lot of works related to the mobile AR systems. Chen *et al.* [4] proposed a system called Glimpse which combines tracking and caching to achieve object detection on mobile devices. However, it can only detect the pre-trained objects such as road signs and human faces. Drolia *et al.* [5]

developed an edge caching solution for image recognition. It will predict the objects which user might meet in a short time and fetch the classification model from the edge server. However, this solution can not get the locations of the objects, which is an essential function for mobile AR applications. Zhang *et al.* [6] proposed a cloud-based framework, CloudAR, to achieve real-time image detection based on object tracking. They segment and track the objects on the client side and do the detection on the server. Ran *et al.* [18] proposed DeepDecision to determine an optimal offloading strategy for AR tasks.

However, these works mainly focus on making the decision of when to offload the tasks to the edge server, they do not take the performance of the on-device models into consideration. We focus on enhancing the performance of the mobile AR applications. A cache and matching algorithm will be used when the performance of on-device deep learning models is poor.

III. System Architecture

In this section, we present our proposed system in detail. As shown in Fig. 3, our system is composed of two parts: mobile device and edge server. We divide all the recognition tasks into two categories. A task which can be executed on mobile devices are defined as the light task T_L, e.g., classification and detection. A task that has to be offloaded to edge servers are defined as the heavy task T_H, such as the semantic analysis. The smart decision algorithm is shown in Algorithm. 1.

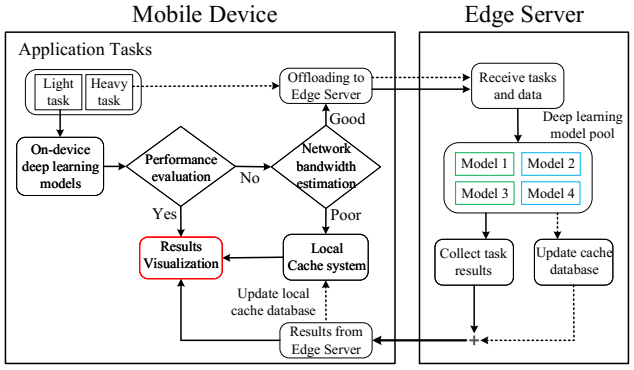

Fig. 3. The system framework.

A. Mobile device

For each mobile AR application, the mobile device will keep capturing the video frames using the camera. For each video frame, there will be different recognition tasks needed to be executed. The heavy task will be directly offloaded to the edge servers because the mobile devices cannot provide enough computation resources. The light tasks will be executed locally with the on-device deep learning models. However, since the performance of the on-device deep learning models is not robust as compared with the deep learning models on the servers. In this case, we design a performance evaluation module. If users are not satisfied with the performance of current recognition tasks, the next task will not be executed

Algorithm 1: The smart decision algorithm

Input : A set of tasks $\{T_1, T_2, ..., T_n\}$, real-time network throughput $\mathcal{B}$, network bandwidth threshold N_σ;
Output: The decision of where to execute the tasks;

```
Initialize the task index i = 1;
while i <= n do
    if T_i ∈ T_H then
        Offload the task T_i to edge server;
    else
        Execute the task T_i with the on-device deep learning model;
        Receive performance feedback P;
        if P = Poor then
            if B > N_σ then
                Offload the task T_i to edge server;
            else
                Execute the task T_i with the cache system;
    i=i+1
```

with the on-device deep learning models. First, the network bandwidth will be estimated. If the network bandwidth is higher than a threshold T_b, the light task will be sent to the edge server. However, if the network bandwidth is lower than T_b, it means that the user is under the poor network condition. In this case, if we still do the task offloading, the performance will be even worse. The reason is that the mobile AR application is very sensitive to the latency. The scene will change a lot when you get the results from the edge server. To overcome this issue. We design a cache and matching algorithm on the mobile device, which can help to improve the performance of light tasks on the device with a lower latency.

In the cache and matching module, we extract the image features using the feature extraction methods such as SIFT. These features are used to match with the objects stored in the cache on mobile devices. After a successful matching object is found, the matching process is ended, and the detection result is presented to the user. There are many different image feature extraction algorithms, the time cost and result quality of applying these algorithms are different. In addition, the size of the cache on the mobile is also vital for the matching system. If the number of objects in the cache is very large, it will take a long time to find the successful matching, and it will neutralize the advantages of the cache system. In this case, we build a hierarchical cache system. For each video frame, we matching the frame with high ranked objects for each class. And once the users offload the tasks to the edge server, the cache system will be updated. There are several trade-offs between the time and the performance in cache and matching algorithms. We show more details in Section IV.

B. Edge server

On the edge server, we will receive the data and tasks from the users. For different tasks, we have different algorithms and models implemented on the edge server. Edge server can provide the service for both heavy and light tasks. For the heavy tasks, the edge server will execute the tasks and send back the results. At the same time, the edge server will backup all the recognition results and extract the image features of the objects. These results and image features will be used to update the cache database. For the light tasks, if we get poor

performance with the on-device deep learning model, or we cannot get a successful match with the cache and matching module, the image frame will be sent to the edge server. Besides sending back the results of light tasks to the mobile device, we will also synchronize the cache on the mobile device with the database on the edge server. In this case, we can still achieve good performance by using the cache and matching module when the network bandwidth is low. Another reason for building the database on the edge server is that we can share this database with multiple users.

IV. Cache and Matching Module

As shown in Fig. 4, we propose a feature matching method for the classification and object detection tasks. One of the main challenges of mobile AR is the limited computing capacity of mobile devices. A high network latency will occur if we offload all the computation to the cloud, or the edge server. We propose a feature matching method which only needs low complexity computation on the mobile device to achieve classification and object detection tasks. Mobile devices can support enough computing resource for feature extraction and matching algorithms. The main process is as following: First,

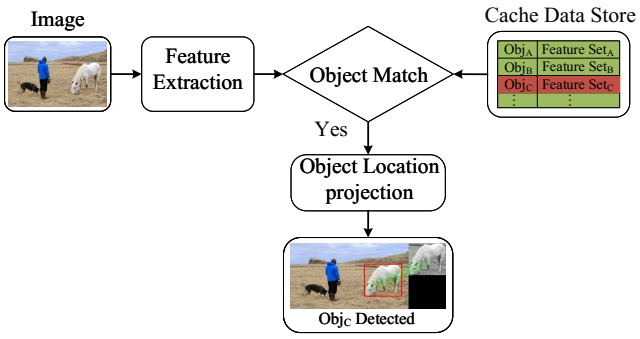

Fig. 4. The cache matching system.

we build a cache data store to store objects and corresponding feature vector sets on mobile devices. This cache store will be updated by the edge server. Then, we extract the features of the target image. After that, we match objects in the cache data store with the target image feature set one by one. For each feature vector in the vector sets, we measure the closest distance between it and the features of the target image. If the closet distance is less than a threshold, we call it a successful match feature vector; if the number of the total successful match feature vector is large than a threshold K, it means the object is in the target image. Then, we project the successfully matched feature vectors on the target image, and a projection box is obtained. In this paper, if the project box cover over 80% of the object, we treat it as a successful projection. In this case, we can achieve low complexity object detection.

A. Features extraction and matching algorithms

The image feature is like a signature of the image. For different feature extraction algorithms, the feature vector dimensions and the matching algorithms are both different. In this paper, we use three feature extraction algorithms, SIFT [10], SURF [11] and ORB [12]. As shown in Table. I, we compare the differences of feature extraction and matching algorithms. For different feature extraction methods, the feature extraction and matching time costs are different. For instance, ORB uses the binary number to construct the feature vector and uses the Hamming method to measure the distance of the feature vectors for matching. So it is the fastest among these three methods. However, when we use the same number of feature vectors for matching, SIFT can achieve higher matching accuracy.

TABLE I
Feature Extraction and Matching Algorithms

Algorithm	Dimension	Data type	Time(s)	Matching method
$SIFT$	128	uint	0.05	FLANN
$SURF$	64/128	uint	0.01	FLANN
ORB	256	Binary	0.003	Hamming

Except the time cost, the number of image features extracted from each image is another important factor that we need to take into consideration. Extracting more image features means a better presentation of the image, and we can get better performance in the matching process. Meanwhile, it will take more computing time and matching time. So the number of features should be properly selected.

B. Cache and Object Database

On the edge server, we create a large object database. In the object database, we store about 10000 objects and the corresponding feature vector sets of each object. These objects are obtained from 20 classes of images in ImageNet dataset [19]. When we offload the tasks to the edge server, all the objects detected by the recognition models will be added into the database with a high rank value. On the mobile device side, we design a cache system, which will fetch the high ranked list of objects from the database on the edge server.

$Class_{ID}$	Object $_{UniqueID}$	Object Feature
$Class_A$	$Rank_1$	Feature Set_1
	$Rank_2$	Feature Set_2
	⋮	⋮
$Class_B$	$Rank_1$	Feature Set_1
	⋮	⋮
⋮	⋮	⋮

Fig. 5. The hierarchical cache data base.

As shown in Fig. 5, there are three layers in the cache database, and the first layer is the class label. The objects belong to the same class are collected together. The second layer is the unique ID of each object. In this paper, the unique ID is the rank value from the edge server. High ranked objects will have a high priority to be matched. When the number of objects in the cache data store is larger than a threshold, the

objects with the lowest rank will be removed from the cache data store. The last layer is the feature set of each object. This feature set composes the fingerprint of the object. We use these object fingerprints to match the feature set of the video frame.

As mentioned early, we match each object in the cache with the target image until we get a successful match. If we put the total database into the cache on the mobile device, the total time for matching is much higher than offloading tasks to the edge server. If there are only a few objects in the cache mobile device, the successful matching rate will be reduced.

C. Trade-offs

The speed and accuracy of cache and matching system are influenced by three key parameters, the methods of feature extraction, the number of features to be extracted for each image, and the cache data store size. Considering the robust of the system and the time cost, we choose SURF as the main image feature extraction method. As shown in Fig. 6, with the increasing of the cache size, the time to finish the matching keep increasing. In the same time, the matching time cost will be much higher if we set the image features size to 800 compared with other settings. To fit the real time processing requirements for the mobile AR applications. We set the cache size to 100 and the image feature size to 500.

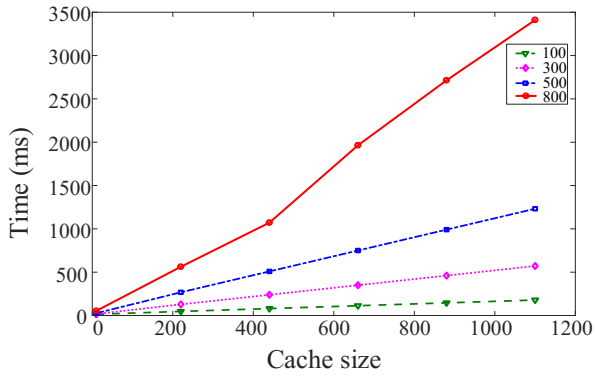

Fig. 6. The trade-offs in the cache system.

V. Evaluation

A. Testbed set up

To evaluate the performance of our proposed system. We build a testbed to implement all the modules proposed in our framework. We simulated a network with MiniNet combined with the SDN controller ONOS. MiniNet is used to generate the network nodes and links, while the ONOS is used to control the routing and data flows. The Mininet and the ONOS are implemented on an HP EliteDesk 800 G2 workstation.

As shown in Fig. 7, there are are total 30 switch-nodes$N = \{n^1, n^2, ..., n^{30}\}$, and 69 links in the network. The link data rate is 50 Mbps, and the delay of the links follow the normal distribution with $\mu = 6ms, \sigma = 1$. The blue node represents the edge server. It is bridged with the Jetson AGX Xavier to support the recognition tasks for the users. We have two users in our testbed system. Each user is an ASUS Zenfone-AR

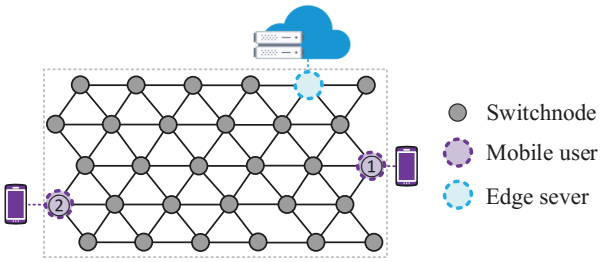

Fig. 7. The network topology of the simulated network.

mobile phone. We use the Tensorflow Lite to develop an image classification and an object detection android applications on the mobile phones. Since the mobile phones cannot be directly bridged to the simulated network, we use two Jestons-Tx2 as the mobile device access node, which is represented by the 2 purple nodes in the simulated network. The mobile phones and Jetson-Tx2s are connected through a router. The mobile phone application will send the data to the Jetson-Tx2 at first. Then, the Jetson-Tx2 will send the received data to the edge server. Once Jetson-Tx2 receives the results from the edge server, it will send back the results to the mobile phone. The time cost between the mobile phone and the Jetson-Tx2 will not be included in the end-to-end latency. We send totally 1500 frames from the two mobile phones to the edge server.

B. Experiment Results Analysis

In this section, we evaluate the performance of the proposed smart decision framework through the experiments on our testbed. Since the heavy tasks will be only executed on the edge server, we do not put the performance of heavy tasks in this part.

1) End-to-end Latency: We implement an image classification and an object detection algorithms on the edge server, and we develop an image classification and an object detection android applications on two Android phones. As shown in Fig. 8 and Fig. 9, the end-to-end delay for running the classification and detection applications on androids phones are 155ms and 224ms, respectively. The inference times are 115ms and 184ms respectively. If we choose to offload the tasks to the server, the network latency is involved. Since the distances between the users and the edge server are different, the network latency for two users is different. The network latency is 850ms and 1380ms for $user_1$ and $user_2$, respectively. On the edge server, the inference times for classification and detection are 64ms and 111ms, respectively. For $user_1$, the end-to-end delays for classification and detection are 924ms and 961ms respectively. For $user_2$, the end-to-end delays for classification and detection are 1452ms and 1501ms, respectively.

Miss classification happens a lot for the on-device classification and detection applications. For example, an apple is mistakenly recognized as a ball. When this error happens, we can choose to do the classification detection with our proposed cache and matching system. The end-to-end delay for the on-device cache and matching system is 1039ms. If we do not need to know the location of the project, the delay is 980ms. We can find that if the network condition is good, using the

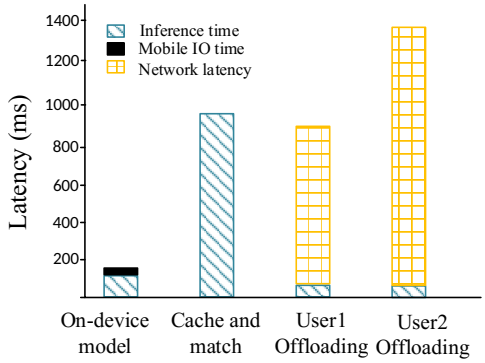

Fig. 8. The end-to-end delay of classification application.

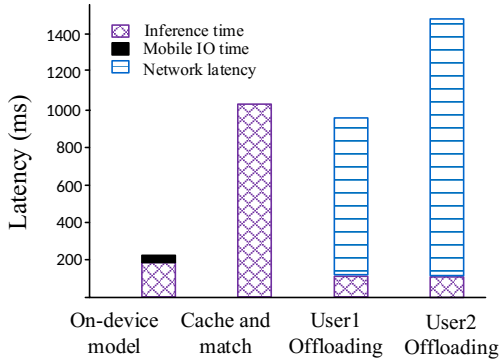

Fig. 9. The end-to-end delay of object detection application .

cache and matching system is not necessary. The reason is that offloading the task to edge server is faster. However, when users are under the poor network condition, like $user_2$, the cache and matching system can help users to get better accuracy with less time cost compared to offloading the tasks to the edge server.

2) Inference Accuracy: The inference accuracy is decided by the deep learning models. High complexity models can achieve a higher precision. On the mobile object detection applications, we use the ssd+MobileNet model. On the edge server, we use the SSD300. The mean Average Precision(mAP) of these model are list in Table. II. The mAP of the cache and matching system is lower than the model on the edge server. The reason is that the cache size is limited, not all the objects will be successfully matched.

TABLE II
THE MAP OF DIFFERENT MODELS

Model	mAP
$SSD300(server)$	81.2%
$SSD + MobileNet(on-device)$	72.7%
$Cache + matching(on-device)$	76.3%

VI. CONCLUSION

In this paper, we proposed a smart-decision framework which combines the advantages of the on-device mobile AR system and the edge-based mobile AR system to achieve real-time recognition tasks. High computation complexity tasks will be offloaded to the edge servers. Low complexity tasks will be executed on the mobile devices or the edge server depending on the network latency. We design a cache and matching system to enhance the performance of mobile AR applications when the on-device deep learning models have poor performance. With our proposed system, the quality of the mobile AR applications is improved. The performance of the smart-decision framework is validated through experiments with a testbed.

REFERENCES

[1] [Online]. Available: Amazon Visual Search. https://www.a9.com/what-we-do/visual-search.html

[2] [Online]. Available: HoloTour. https://www.microsoft.com/en-us/hololens/apps/holotour

[3] S. Gammeter, A. Gassmann, L. Bossard, T. Quack, and L. V. Gool, "Server-side object recognition and client-side object tracking for mobile augmented reality," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, June 2010.

[4] 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 the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2015, pp. 155–168.

[5] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," in *2017 IEEE/ACM Symposium on Edge Computing (SEC)*, San Jose, CA, Oct 2017.

[6] Z. WENXIAO, S. LIN, F. H. BIJARBOONEH, and H. CHENG, "Cloudar: A cloud-based framework for mobile augmented reality," 2017.

[7] D. Wagner, D. Schmalstieg, and H. Bischof, "Multiple target detection and tracking with guaranteed framerates on mobile phones," in *Mixed and augmented reality, 2009. ISMAR 2009. 8th IEEE international symposium on*. IEEE, 2009, pp. 57–64.

[8] 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*, vol. 16, no. 3, pp. 355–368, 2010.

[9] 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*. Springer, 2015, pp. 167–199.

[10] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[11] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer vision–ECCV 2006*, pp. 404–417, 2006.

[12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, 2011, pp. 2564–2571.

[13] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

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

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[17] M. Apte, S. Mangat, and P. Sekhar, "Yolo net on ios," 2017.

[18] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.