Computer Networks 200 (2021) 108513

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

VCMaker: Content-aware configuration adaptation for video streaming and analysis in live augmented reality

Ning Chen, Sheng Zhang *, Siyi Quan, Zhi Ma, Zhuzhong Qian, Sanglu Lu

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

ARTICLE INFO

Keywords:
Mobile augmented reality
Deep reinforcement learning
Video configuration adaptation

ABSTRACT

The emergence of edge computing has enabled mobile Augmented Reality (AR) on edge servers. We notice that the video configurations, i.e., frames per second (fps) and resolution, significantly affect the key metrics such as detection accuracy, data transmission latency and energy consumption in real AR application. Besides the time-varying bandwidth, we observe that the video contents, such as moving velocities of target objects, have remarkable impacts on the configuration selection. In addition, we take the energy consumption on data transmission into consideration. In this paper, we propose VCMaker, a system that generates video configuration decisions using reinforcement learning (RL). VCMaker trains a neural network model that selects configuration for future video chunks based on the collected observations. Rather than rely on any pre-programmed models, VCMaker learns to make configuration decisions solely through empirical observations of the resulting performances of historical decisions. In addition, we leverage the dynamic Region of Interest (RoI) encoding and motion vector-based object detection mechanisms to advance VCMaker. We implemented VCMaker and conducted extensive evaluations. The results show that VCMaker achieves a 20.5%–32.8% higher detection accuracy, and 25.2%–45.7% lower energy consumption than several state-of-the-art schemes.

1. Introduction

With the advancement and popularity of Deep Learning in recent years, we are capable of accurately detecting and classifying much more complex objects in our surroundings by deploying fine-designed machine learning models, which make mobile Augmented Reality (AR) [1–7] applications highly intelligent in the fields about entertainment, tourism and education. Existing AR systems, such as ARKit, Microsoft HoloLens [8] and the announced Magic Leap One [9], facilitate the close interaction between humans and the virtual world.

However, only a small amount of AR applications are deployed in mobile devices and developed based on deep learning framework because (1) neural inference on mobile devices is significantly energy-guzzling; (2) executing computation-intensive inferences in resource limited mobile devices may not meet users' performance requirements [10]. Hence, a promising approach is to offload the AR input, including the video frames, to an nearby edge server which is equipped with enough resources to support compute-intensive deep learning inference. The edge server leverages state-of-the-art detecting algorithms (i.e., YOLOv3 [11], YOLOv5 [12]) that adopt a detector strategy that views the object detection problem as a regression problem and learns the object boundary coordinates, as well as the corresponding class probability. Nonetheless, offloading object detection tasks to an edge server is never trivial due to the stringent requirements on detection accuracy and end-to-end latency, which largely depend on the video configuration for transmission and detection.

The choice of configuration directly affects the transmission latency, detection accuracy and energy consumption. Specifically, we take the fps and resolution selection as an example to elaborate the impacts of AR video configuration on these metrics. When faced with a terrible network, AR videos with expensive configurations (i.e., with high fps and resolution) are expected to cause unpredictable latency of data transmission and object detection, which may significantly reduce the detection accuracy due to the changes in user's view–the frame locations where the object was originally detected may no longer match the current location of the object. In addition, an expensive configuration may lead to excessive energy consumption. How to efficiently choose the optimal configuration is the key problem we intend to solve.

However, for a video analytics pipeline in an AR system, it is never trivial to determine the best video configuration because it varies over time at a timescale of minutes or even seconds, which is mainly a result of (1) the fluctuation of network bandwidth; and (2) the time-varying moving speeds of target objects. We may choose a cheap configuration

* Corresponding author.
E-mail addresses: ningc@smail.nju.edu.cn (N. Chen), sheng@nju.edu.cn (S. Zhang), siyiquan2021@gmail.com (S. Quan), marszer@foxmail.com (Z. Ma), qzz@nju.edu.cn (Z. Qian), sanglu@nju.edu.cn (S. Lu).

https://doi.org/10.1016/j.comnet.2021.108513
Received 28 June 2021; Received in revised form 22 September 2021; Accepted 24 September 2021
Available online 14 October 2021

when the network is degraded, and enhance the configuration if the network condition is improving.m For instance in the traffic tracking scenario, we may use a low frame rate (e.g., 3 fps instead of 30 fps) if cars are moving slowly (i.e., at a traffic stop). However, assigning a low frame rate to count the number of cars will significantly hurt the accuracy because large number of fast-moving cars are not captured successfully. Hence, we have to frequently adjust the configuration of video pipeline to improve detection latency and energy consumption while achieving the desired accuracy. It is unrealistic to test all possible configurations in a given period.

Unfortunately, using traditional model-based techniques such as Bayesian optimization [13], multi-armed bandits [14], or optimal experiment design [15] to get the best configurations policy at the granularity of seconds is time consuming. Indeed, these techniques typically assume a near stationary environment (i.e., with stable bandwidth and same type of videos), where it is beneficial to profiles only once upfront or infrequently. Our proposed scenario, however, is non-stationary. For example, tracking vehicles when they move quickly requires a much higher frame rate to capture all the objects than they move slowly, but the time at which each possible condition occurs may vary by minutes, or even seconds.

In this paper, instead of relying on detailed analytical performance modeling, we adopt a black-box approach to get adaptive configuration for video streaming transmission in live Augment Reality. Motivated by recent achievements of deep reinforcement learning (DRL) [16–18] in video streaming [19], and job scheduling [20], we propose the learning-based VCMaker, an intelligent content-aware decider for adaptive video configuration selection, which starts with no prior knowledge, and learns the best configuration decisions through continuous reinforcement based on the reward signals that indicate the key metrics from previous decisions. VCMaker uses a neural network to map "raw" observations (e.g., a combination of estimated bandwidth, captured velocity as well as historical configurations) to the configuration decision. VCMaker incorporates diverse observations into the configuration space in a scalable way. In our scenario, VCMaker maximizes the accumulative discounted reward rather, hence a temporary better configuration may not benefit the following configurations. In particular, VCMaker leverages the state-of-the-art A3C (asynchronous advantage actor–critic network) [17] model to train its policy network. Having trained the convergent model, VCMaker is capable of making efficient configuration decisions for video streaming transmission in real AR scenarios.

In addition, we propose several extra techniques to advance VCMaker. We adopt Dynamic RoI Encoding technique, which set different bitrates based on the Regions of Interest (RoIs) detected in the last frame. We need to decide the frame resolutions both inside and outside the RoIs during each encoding period. Besides, on the local AR device, we adopt a lightweight and fast object detection approach based on the extracted motion vector from the current frame as well as the cached detection result of previous frames to adaptively adjust the bounding box on the current frame. We summarize our major contributions of this work as follows:

- We quantify accuracy, latency, and energy requirements in an end-to-end AR system. Besides, we identify several key factors that affect the video configuration, such as the time-varying bandwidth, the time-shifted AR scenes, and the conflicted latency–accuracy–energy tradeoff. To our best knowledge, it has not been revealed in the existing works to take the video content into consideration for streaming transmission in live augmented reality.
- We present VCMaker, an intelligent content-aware encoding system that learns a smart configuration policy from past experiences. We use A3C algorithm to train VCMaker, which views the observed state (i.e., estimated bandwidth, captured velocity, et al.) as input, and output the policy (i.e., the probability distribution of all configurations). Based on this policy, VCMaker is likely to select a best configuration.
- We implement a prototype of VCMaker based on commodity hardware (e.g., Jetson TX2 and edge server with RTX2080 Ti GPUs), and evaluate its performance using diverse types of AR videos. We find VCMaker rivals or outperforms these algorithms by improving the average detection accuracy by 20.5%–32.8%, and reducing the energy consumption by 25.2%–45.7%.

The remainder of this paper is organized as follows. Section 2 describes the observations and challenges. Section 3 presents the system architecture. Section 4 details our learning-based framework VCMaker. Section 5 shows the implementation and evaluation of VCMaker. We review some related works in Section 6, while we provide the conclusion in Section 7.

2. Observations and challenges

Proceeding from the AR requirements, we expound some vital observations that motivate us to design adaptive configurations for AR video streaming, and point several primary challenges that we have to address.

2.1. Insights on the environment

We analyze some key factors including the network bandwidth and video content that affect the video configuration, and do some preliminary works to demonstrate them.

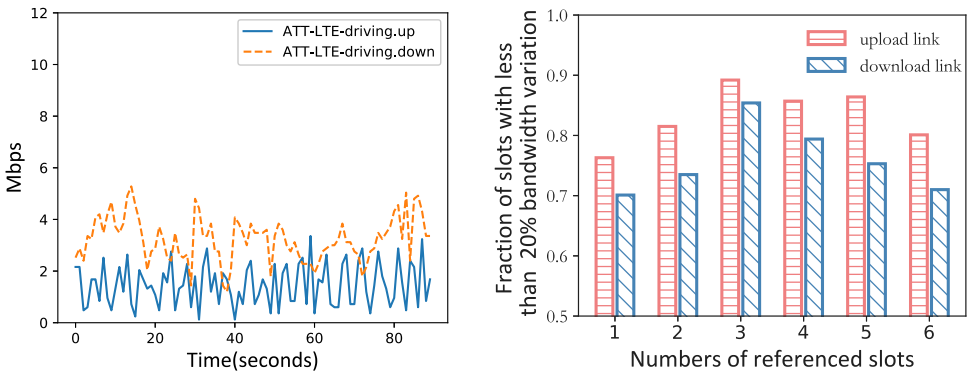
2.1.1. Fluctuation in network bandwidth

Most AR applications are deployed in mobile devices and communicate with edge server over cellular networks such as LTE, which is likely to experience bandwidth fluctuation [21]. To fit the variable bandwidth, rather than a fixed configuration (i.e., fps and resolution), VCMaker aims to adaptively select the ideal configuration for transmission and object detection. We collect two ATT-LTE network bandwidth traces from the Mahimahi [22] project and depict them as Fig. 1 shows. These traces represent the time-varying capacity of U.S. cellular networks as experienced by a mobile user. Each line gives a timestamp in milliseconds (from the beginning of the trace) and represents an opportunity for one 1500-byte packet to be drained from the bottleneck queue and cross the link. If more than one MTU-sized packet can be transmitted in a particular millisecond, the same timestamp is repeated on multiple lines. Based on the download and upload traces, we had the following observations:

▷ *Periods with either extremely low or high bandwidths are uncommon even the throughput is volatile.* We show the bandwidths in a period as Fig. 1(a), and we find only 14.6% of the time, the upload bandwidth is smaller than 2 or larger than 5 Mbps. Similarly, it occurs only 14.8% for the download bandwidth, which indicates the extreme bandwidth can hardly impact the configuration decision;

▷ *Most of the slots share the similar bandwidth distribution.* In other words, the bandwidth of the next slot is closely related to the average value of the past several slots. As Fig. 1(b) shows, in LTE uplink, 76.3%, 89.2%, and 80.1% of the slots have less than 20% bandwidth variation compared to the previous one, three, and six slots, respectively, which implies that there is only a slight fluctuation in consecutive slots. The bandwidth in LTE downlink has similar laws.

The above observations suggest that the bandwidth fluctuates in a limited range (e.g., [0,10] in Fig. 1) in a given time scale, but varies less (e.g., [0,5]) in a much small interval (e.g., from 350 to 360), which provides a promising approach to estimate the future bandwidth based on past values without future network information. Then, this estimation can benefit the configuration selection.



(a) Available network bandwidth (b) Bandwidth estimation

Fig. 1. Bandwidth fluctuation over time. (a) The uplink and downlink bandwidth. (b) The Y-axis denotes the fraction of slots, in which the bandwidth is within $[80\%, 120\%]$ of the average bandwidth of the past 1 to 5 slots.

2.1.2. Time-shifted viewpoint-moving speed

In real AR scenarios, the user's viewpoint follows with the target objects, which may not always keep still. In general, the target objects move with a time-shifted speed. In such cases, it is unwise to fix the video configuration for transmission and detection. For instance, if we select a low fps while the target objects move at a high velocity, the locations (i.e., bounding boxes) of target objects detected in the last frame may no longer match the current locations of the same objects, which may cause a deteriorated accuracy. Similarly, choosing a high fps for the AR videos with low-speed targets may incur extra transmission and detection cost, but gain little improvement in accuracy. As Fig. 2(a) shows, if we fix the configuration, then it seems that the detecting accuracy decreases with the increase of the moving velocity of the target object. Hence, the selected fps is expected to fit the viewpoint-moving velocity. In addition, if we view the AR experience through the perspective of the users, what they care about is the regions with potential objects of interest (e.g., Regions of Interest).

2.2. Latency–accuracy–energy tradeoff

The mobile devices offload the AR input (i.e., the sampled images) to the edge server for object detection, and the majority latency results from the data transmission and image inference. As stated before, the expensive configuration (i.e., a high *resolution* and *fps*) promises to gain a gratifying accuracy, but causes an unpredictable data transmission latency, and increases the energy consumption. Users may show various weights of preference on accuracy, latency and energy. We conducted an experiment to measure the impact on detecting accuracy and total latency resulting from the video configuration. We connected a Nvidia Jetson TX2 to an edge server equipped with RTX2080 Ti GPU through WiFi-2.4 GHz, and streamed video with diverse configurations from the Jetson to the edge server for inference, and used Python power monitor library to collect the energy information. As the upper half part of Fig. 2(b) shows, when the bandwidth is sufficient, the high detection accuracy is greatly beneficial from an expensive configuration, e.g., a high resolution and fps. However, as the lower half part of Fig. 2(b) shows, when facing a terrible network, the expensive configuration may decrease the accuracy due to the long transmission and detecting latency. Besides, selecting an expensive configuration may cause energy-guzzling. Intuitively, there exists an implicit nonlinear correlation among these metrics. An efficient configuration decider is expected to identify the nonlinearity.

2.3. Excessive energy consumption of AR applications

Compared to running compute-intensive deep learning inferences locally on a mobile device, an edge assisted approach promises to extend the battery life of a device. However, it still requires huge energy consumption for multiple processes such as screen rendering, image conversion, and data transmission. For example, the results from an AR testbed in [23] shows that if continuously transmitting the latest videos captured from a camera to a nearby edge server for object detection, a 3000 mAh battery of a mobile device (i.e., smartphone) is likely to be exhausted within about 2.3 h, and over 60% of the energy is consumed by the pre-process. Therefore, the energy efficiency impends AR clients to gain a better performance. It may make sense by decreasing the video configurations at the cost of degrading the detection accuracy. Hence, a fine-designed configuration is crucial to the edge assisted AR application. In this paper, the configuration of playback AR video is user-defined, while the configuration of videos for transmission and detection is adaptively decided by VCMaker, hence we only take the energy consumption for data transmission into consideration, and ignore the cost of object detection in an edge server.

Achieving adaptive configuration for AR video streaming is never trivial and it faces several great challenges. We summarize them as follows: (1) We lack future bandwidth and moving velocities of target objects; (2) We have to tradeoff the each resulting metric (i.e., latency, accuracy and energy); (3) How to reduce the energy consumption and extend the battery life is necessary.

3. System architecture

To resolve these challenges listed in Section 2, we propose a system named VCMaker to obtain high detection accuracy with an acceptable overhead in the energy and latency. It learns to adaptively choose the best configuration from historical experience. In this section, we first introduce the workflow of VCMaker, and then propose two advanced mechanisms.

3.1. Workflow of VCMaker

Fig. 3 illustrates the system architecture of our proposed VCMaker, and we mark the key steps with numbers. At a high level, a general AR system contains the AR client (i.e., an AR headset or a smartphone) and the powerful edge server.

In the AR client, the camera captures the real time frames, and fills them into the image buffer. Then, two parallel steps utilize the buffered images for transmission and motion vector extraction, respectively. Before each transmission period, VCMaker will select a configuration for the video encoder (step 1), which then pulls the cached detection results of the last frame (step 3) and leverages the RoI encoding mechanism to encode the video with selected configuration (step 2). Meanwhile, the local device incorporates the cached detecting results of the last frame into the extracted motion vector of the current frame

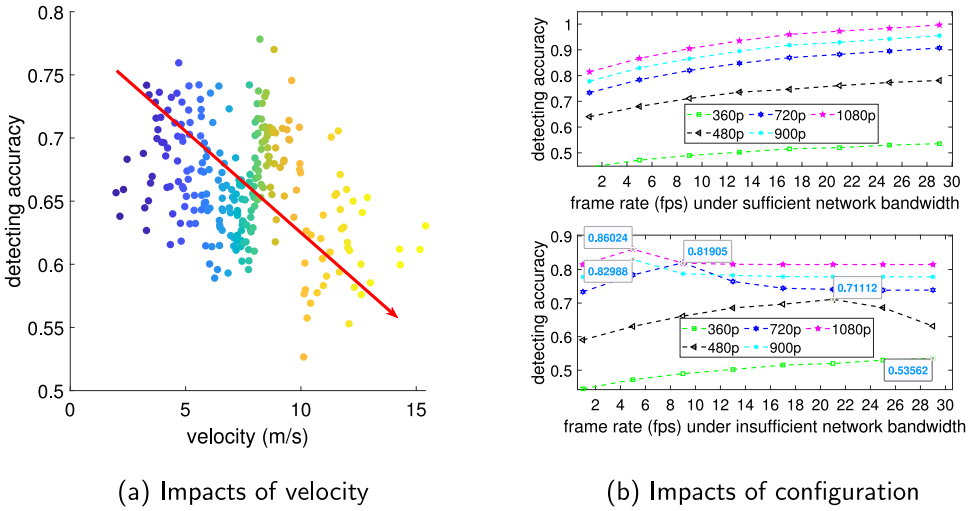


Fig. 2. Factors affecting the detecting accuracy. (a) The impact of velocity on the detecting latency; (b) the upper and lower half show the variety of accuracy under sufficient and insufficient bandwidth respectively.

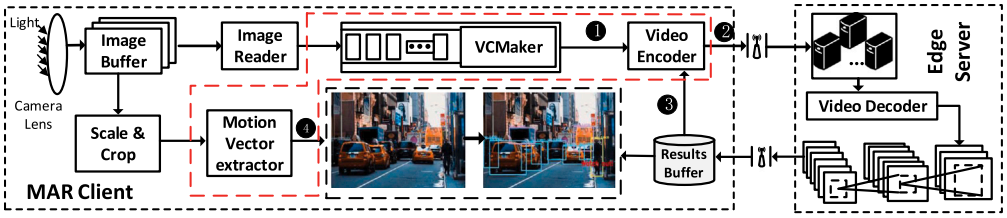


Fig. 3. System architecture. VCMaker selects the best configuration that matches current state, uploads the encoded video to the edge server for object detection, and stores the results for motion vector-based object detection.

to achieve fast and accurate detecting (step 4). The edge server with powerful detection algorithms such as YOLOv3, SSD or fast-RNN is developed to process received image frames with Convolution Neural Network (CNN) and send the detection results back to the AR client. The client allocates an additional memory to store the results.

3.2. Dynamic RoI encoding

During the playback of AR video, users may show interests only in several objects, but ignore the surroundings. Therefore, it makes sense to identify the regions with potential objects of interest, which we will refer to as regions of interest (RoIs). Intuitively, this region is likely to adopt near-lossless compression method to maintain high quality while lossier compression method for the background or non-RoI area. Hence, we adopt Dynamic RoI Encoding technique to reduce the data transmission latency while keeping a satisfied object detection accuracy. In this paper, Dynamic RoI Encoding uses higher degrees of compression in the parts that contain less objects of interest and maintains high quality within RoIs, which largely reduces the scale of encoded frame.

Note that in VCMaker, the term RoI is applied both in video compression and object detection. In the field of video compression, RoIs are the areas that contain more potential objects inside the frames and will be encoded with a high bitrate. While for object detection applications, RoIs are the output bounding boxes of the region proposal network. Therefore, we can exploit this nature and combine them together. We advance the dynamic RoI encoding technique that takes the result (i.e., internal RoI) generated from object detection CNNs as the RoI in the encoding process. Specifically, the image encoder applies the output RoI of the last processed frame using CNNs to encode the next frame. Based on the similarity between two frames captured in a short moment, it is likely to accommodates a degree of motion by slightly extending each RoI by one macro-block [24].

3.3. Motion vector-based object detection

Based on the motion vector extracted from the encoded frame and the cached object detection result of the last frame, the Motion Vector-Based Object Detection technique could estimate the object detection result (i.e., bounding box) of the current frame. Motion vectors are widely used in main encoding format (e.g., H.264 and H.265) to represent the pixels offset among frames in order to get a higher compression rate. A large number of commodity mobile devices deploy specific hardware to advance video encoding and obtain the motion vectors quickly. In the camera side, once a new frame is captured, VCMaker will pass this frame to the Dynamic RoI Encoding session. The video encoder takes the frame which corresponds to the last detection result in the cache as its reference frame to do inter-frame compression. Then, VCMaker will extract all motion vectors of the encoded frame. To track each target object in the current frame, VCMaker first gets their bounding boxes in the last uploaded frame, then compute the mean of each target object with its all motion vectors in the bounding box, and use them to shift the former position to the current position in the current frame.

We apply the above two mechanisms to advance the AR experience. VCMaker conducts the configuration decision process for RoI encoding, and the performance of motion vector-based object detection largely relies on the cached detection results of the last frame, which is affected by the video configuration as well. Hence, VCMaker plays an essential role in the AR system. We focus on designing an efficient configuration decider, which learns the configuration policy from the current environment including the network bandwidth usage, energy consumption of data transmission, and the feedback accuracy. For instance, if the energy consumption exceeds the budget, it may select a cheaper configuration (i.e., a low *resolution* and fps). However, a low configuration may degrade the detection accuracy. Thus how to assign

an adaptive configuration is essential to the user's QoE. In the next section, we will show the detailed learning method of VCMaker.

4. Learning-based algorithm design

In this section, we will introduce the detailed design of VCMaker. Considering that the VCMaker dynamically adjusts the configuration through the feedback performance of the past configurations, which is viewed as a learning process, we model it as a learning-based framework. We first illustrate the basic learning mechanism, and present the formal definition of our DRL framework of VCMaker. Then, we elaborate on the detailed training methodology

4.1. Basic learning mechanism

Without future information, learning-based algorithm learns an effective policy largely from the historical experience. Specifically for deep reinforcement learning (DRL), it continuously interacts with the environment and adjusts the policy based on the feedback resulting reward. The environment is a highly abstract that integrates the surrounding information, but its agent can observe only a small part of the environment, which is referred to as the state. To simplify the description, we divide the total time $\mathcal{T}$ into multiple time slots of equal length. At each time slot t, the RL-agent observes a state s_t and chooses an action a_t based on a specific policy π. When the action is done, the agent will receive an instant reward r_t and transit to the next state s_{t+1}. Through constant interactions with the environment until done, the RL-agent is expected to obtain a high accumulative reward.

4.2. DRL framework

Considering that we can hardly gain the accurate future information of the environment and the state transition probability, we propose the model-free DRL-based VCMaker to adaptively generate configurations for the transmission and object detection. The detailed designs are shown in Fig. 4.

4.2.1. Action space

VCMaker adaptively chooses a configuration (i.e., the fps and resolution) for transmission and detection, hence the components of configuration make up the action. In addition, we introduce the dynamic RoI encoding mechanism to advance the transmission. Therefore, we couple three elements to form the action space, i.e., $a_t = (fps_t, res_t^{in}, res_t^{out})$, where res_t^{in} and res_t^{out} are the selected resolutions inside and outside the RoIs, respectively. For a newly received state s, the DRL-agent selects an action a based on the policy $\pi_\theta(s, a)$, which is defined as the probability distribution over the action space, and then gets an instant reward. The policy $\pi_\theta(s, a)$ is the output of *policy network*, whose parameter is set to θ.

4.2.2. State space

In fact, the state is the available observation of RL-agent (i.e., the encoder in this paper to make configuration decisions) from the environment. Obviously, it is impossible for the RL-agent to gain the whole information of the environment. Hence to approaches the role of the God with comprehensive future knowledge, the RL-agent has to learning from historical experience continuously. In such case, an exhaustive definition of state is vital to the model training. In our scenario, we combine four elements as follows to form the state:

▷ Historical k decisions $\{a_{t-i} | 1 \le i \le k-1\}$. We divide the whole time scale $\mathcal{T}$ into many time slots of equal length. In slot t, we assume that the encoder encodes the frames with fps_t, res_t^{in}, and res_t^{out}. It is common that the environment seems not to arise huge changes in two consecutive time slots, thus the previous configuration is able to guide the video encoding for next slot. We denote the number of referenced configurations used in VCMaker by k, which depends on video content, the feedback performance and network status. For instance, if the camera is a capturing a racing game, it makes no sense to set a large k, thus we set k to 1. While when tracking a slow-moving object, a large k is expected. In practice, it is never trivial to choose an optimal k, because even a near static camera can capture both slightly changeable videos (e.g., in midnight) and dynamic ones (e.g., rush hours). The scheduling of k is left as our future work, and incorporating it into VCMaker can elevate VCMaker significantly.

▷ Estimated bandwidth $B_{est}^{(t+1)}$. The encoder aims to select the optimal configuration (i.e., resolution and fps) that best match the available network bandwidth, yet lacks the prior knowledge of future bandwidth. As stated before, the bandwidth varies in a limited range during slot t, and adopting more past bandwidths can obtain a more accurate bandwidth estimation of next slot. Thus in this paper, we take the weighted bandwidth of past k slots as the estimated bandwidth $B_{est}^{(t+1)}$ of slot $t+1$,

$$B_{est}^{(t+1)} = \sum_{i=t-k+1}^{t} \omega_i B_i, \tag{1}$$

where $\omega_i < \omega_j$ if $i < j$, and $\sum \omega_i = 1$. In Section 5, we set these weights to emulate exponential moving average.

▷ Average velocity v_t. It is known to us that VCMaker should assign a high f ps when capturing fast-moving objects and a low fps for slow-moving object. Otherwise, the locations (i.e., bounding boxes) of target objects detected in the last frame may no longer match the current locations of the same objects, which may cause a deteriorated accuracy. Assume the target objects set is $\mathcal{Z} = \{z_1, z_2, \ldots, z_n\}$, and the previous configuration is $(fps_t, res_t^{in}, res_t^{out})$. The video frames are offloaded to the nearby edge server for object detecting using the YOLOv3 algorithm, which predicts the positions of potential objects, namely Bounding Box Prediction. For each frame f in slot t, and $i \in \mathcal{Z}$, suppose that the feedback *Bounding Box Prediction* set is $(x_i^f, y_i^f, w_i^f, h_i^f, c_i^f)$, which contains the center coordinates of x-axis and y-axis (x_i^f, y_i^f), box height and width (w_i^f, h_i^f), and class c_i^f. Considering the target object moves regularly, we use *Manhattan distance* [25] rather than *Euclidean distance* to calculate the distance the object moves in one slot. Thus the velocity v_t is defined as the average distance of all objects at slot t, i.e.,

$$v_t = \frac{1}{|\mathcal{Z}|} \sum_{z_i \in \mathcal{Z}} \left[\left| x_i^{fps_t} - x_i^1 \right| + \left| y_i^{fps_t} - y_i^1 \right| \right], \tag{2}$$

where (x_i^1, y_i^1) represents the center coordinate of target object z_i in the first frame at slot t. We find that some objects may disappear during this slot; and then, we set its location in the frame it does not appear at that slot to be the farthest corner among all four corners.

▷ Feature map of the latest frame. It is known that a convolution neural network is used to simulate the specific characteristics in the visual pathway, of which color shades and the shape edges are beneficial to make efficient video configuration. Diverse filters are designed to mine best knowledge from every perspective. We provide the tuned filters for convolution and pooling in our implementation and evaluation section.

To sum up, in this paper we combine the historical decisions, estimated bandwidth, and the velocity to form the state space.

4.2.3. Reward

In our system, VCMaker is likely to get an instant reward r_t when receiving a_t to state s_t. In real AR scenarios, mobile users emphasize detecting accuracy, total latency and energy consumption, thus these three metrics make up of reward. Suppose that $a_t = (fps_t, res_t^{in}, res_t^{out})$ during slot t.

▷ Latency. As mentioned before, when offloading the detection tasks to the more powerful edge servers, the image encoding and transfer process add significant latency. Long latency may reduce the detection accuracy and further degrade the user's QoE. Hence, we take it as a

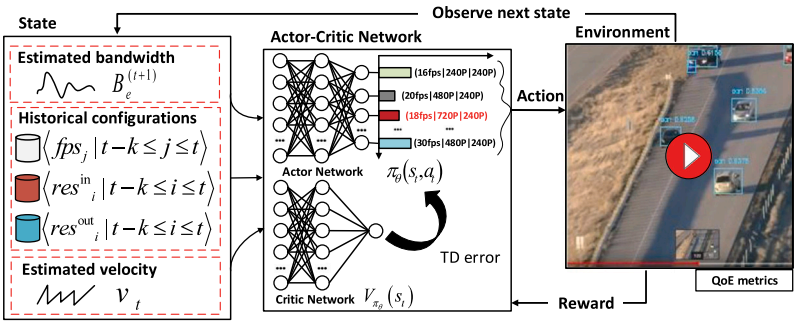


Fig. 4. Design details. The actor–critic networks take the observations as input, and output the policy distribution and Q-value.

component of the tuned reward. We model the end-to-end latency make a normalization during slot t as follows,

$$d_t = \sum_{f=1}^{fps_t} \frac{d^f_{e2e}}{fps_t} = \sum_{f=1}^{fps_t} \frac{d^f_{stream} + d^f_{infer} + d^f_{back}}{fps_t}, \tag{3}$$

in which the d^f_{e2e} is the end-to-end latency of frame f, which is determined by three main components: (1) d^f_{stream} is the time to stream frame f from the AR device to the edge server; (2) d^f_{infer} represents the time to run the object detection inference on frame f on the edge server; (3) d^f_{back} indicates the time to transmit the detection results back to the AR device. We use the average latency metric d_t to represent the latency of current configuration.

▷ Detecting Accuracy. Based on the cached inference result of last frame, the AR device adopts the Motion Vector technique to estimate the object detection result (i.e., bounding box) of current frame. To evaluate the detection accuracy,we first obtain the ground truth of each frame in an offline manner, then calculate the IoU (the standard detection metric used in the object detection task [26]) of each detected bounding box and its ground truth as the accuracy of this detection. In addition, we define the percentage of detected bounding boxes with less than 0.75 IoU as false detection rate. Suppose that the raw video is captured by the AR device with frame rate fps_t at slot t, then we calculate the false detection rate fdr_t as

$$fdr_t = \frac{\left|\{f_i | IoU_i \le 0.75, 1 \le i \le fps_t\}\right|}{fps_t}, \tag{4}$$

where IoU_i indicate the IoU of frame f_i.

▷ Energy Consumption. Performing an AR application is energy-guzzling, which is mainly caused by the camera sampling, frame rendering and data transmission. In fact, the energy consumptions about camera sampling and frame rendering are up to the display configuration chosen by users, and thus only the data transmission step depends on the configuration decided by our proposed VCMaker. Hence we only take this part of energy consumption into consideration, and denote the energy consumption of data transmission during slot t by e_t.

As users may show diverse preferences in these three components, hence we define the final reward r_t of once configuration decision at slot t by a weighted sum of these three vital components,

$$r_t = -\alpha_1(d_t - \bar{d}) + \alpha_2(fdr_t - \bar{fdr}) + \alpha_3(e_t - \bar{e}), \tag{5}$$

where α_1, α_2 and α_3 represent the weight factors to indicate the preference on latency, accuracy and energy consumption. It is a quite general and widely used definition as it models varying user preferences on multiple contributing factors. In the final implementation, to mitigate the influence by diverse fluctuation ranges of these metrics, we add $\bar{d}$, $\bar{c}$ and $\bar{u}$, which are set to the corresponding average of delay, average detecting accuracy and average energy consumption respectively. We measure them with substantial empirical video traces.

4.3. Training methodology of VCMaker

As stated before, the sophisticated state has multiple dimensions and each dimension has lots of possible values, though the action (e.g. configuration) space is bounded, there still have infinite (s_t, a_t) pairs. We leverage the policy gradient based method whose policy π is represented by a neural network, and we refer the parameter of this neural network to as the policy parameter θ. Hence, the policy (i.e., the output of policy network) is represented by $\pi(a_t|s_t;\theta) \to [0,1]$, which indicates the probability distribution of taking each action at state s_t. DRL aims to maximize the accumulated discounted reward $J(\theta) = \mathbf{E}\left[\sum_{t=t_0}^{t_0+|\mathcal{T}|} \gamma^t r_t\right]$ through training a best policy mapping a state to each action, where $\gamma \in (0,1]$ is the discount coefficient of future reward.

In this paper, our proposed VCMaker takes good use of the A3C algorithm consisting of the Actor–Critic networks that continually trained with *policy gradient method*, which calculates the gradient of the accumulated reward obtained by following current trained policy. We highlight the key steps of the training processes. Firstly, to update the Policy (e.g., Actor) Network parameters in slot t, we calculate the gradient of $J(\theta)$ with respect to θ as [27]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \nabla\theta \log\left(\pi_\theta(s_t, a_t)\right) A^{\pi_\theta}(s_t, a_t)\right], \tag{6}$$

where $A^{\pi_\theta}(s_t, a_t)$ represents the advantage function that shows the deviation between the expected accumulative discounted reward when choosing a_t at state s_t under policy π_θ and the expected reward for all actions at state s_t derived from policy π_θ. In fact, the advantage function indicates how much better a current specific selected action performs compared to the "average action" selected under current policy. Hence, the goal of advantage function is to reinforce the actions with positive and big advantage value $A^{\pi_\theta}(s,a)$, and degrade the actions with negative and small advantage value $A^{\pi_\theta}(s,a)$.

Rather than use $A^{\pi_\theta}(s,a)$ directly, the agent of VCMaker extracts a trajectory of video configuration decisions and calculates the empirically advantage $A(s_t, a_t)$ as the unbiased estimated $A^{\pi_\theta}(s_t, a_t)$. Hence the update method of Policy (e.g., Actor) network parameter θ follows

$$\theta \leftarrow \theta + \alpha \sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(s_t, a_t) A(s_t, a_t), \tag{7}$$

where α is the learning rate. Behind the update law, we learn that the gradient direction $\nabla_\theta \log \pi_\theta(s_t, a_t)$ indicates the way how to adjust parameter θ to enhance $\pi_\theta(s_t, a_t)$. In Eq. (7), θ is updated along the gradient descent direction. Note that the step size during training is decided by the advantage value $A^{\pi_\theta}(s_t, a_t)$.

As stated before, each update is to reinforce actions to obtain better feedbacks based on the advantage value $A(s_t, a_t)$. Specifically, to obtain $A(s_t, a_t)$, we also need to calculate the estimated *value function* $v^{\pi_\theta}(s)$ (i.e., the total expected reward following the policy π_θ). As Fig. 4 shows, to accelerate the computing, a *critic network* is used to estimate $v^{\pi_\theta}(s)$

from current observed rewards. Based on the *Temporal Difference* [28] method, the critic network parameter θ_v follows

$$\theta_v \leftarrow \theta_v - \alpha' \sum_t \nabla_{\theta_v} \left(r_t + \gamma V^{\pi_\theta}\left(s_{t+1};\theta_v\right) - V^{\pi_\theta}\left(s_t;\theta_v\right)\right)^2, \quad (8)$$

where $V^{\pi_\theta}\left(s_t;\theta_v\right)$ represents the estimated $v^{\pi_\theta}\left(s_t\right)$ that is generated by the critic network. For a specific experience (s_t, a_t, r_t, s_{t+1}),[1] we calculate the advantage value $A(s_t, a_t)$ as $r_t + \gamma V^{\pi_\theta}\left(s_{t+1};\theta_v\right) - V^{\pi_\theta}\left(s_t;\theta_v\right)$. Note that the role of critic network is only to evaluate the policy of actor network rather than train the actor network directly. In our proposed scenarios, only the Actor (i.e., Policy) Network is used for making real time video configuration decision.

To further balance well of exploration and utilization to avoid falling into suboptimal solution, an *entropy regularization* [17] term is added to encourage exploration to discover better policies during training, which significantly makes the agent converge to a better policy. Given this term, we modify Eq. (7) as

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta\left(s_t, a_t\right) A\left(s_t, a_t\right) + \beta \nabla_\theta H\left(\pi_\theta\left(\cdot|s_t\right)\right), \quad (9)$$

where $H(\cdot)$ represents the policy entropy which push θ to the direction with higher entropy to encourage exploration, and β is entropy weight that indicates the importance of this entropy regularization term, which is set to a large value in the beginning to find the best actions, and decreases over time to allow VCMaker to have more opportunities to utilize the best actions.

Considering there have infinite pairs of (s_t, a_t), it is inefficient to train VCMaker with single agent. Hence a parallel approach is applied to enhance exploration and speed up training. We set n threads (i.e. RL agents) simultaneously with diverse environment settings (e.g., different types of AR videos and diverse network traces), and thus each agents will experience different states and transitions, thereby avoiding the correlation. During training, each agent continuously collects the generated samples (i.e., tuple $\{s_t, a_t, r_t, s_{t+1}\}$), and computes the policy gradient and perform a gradient descent method following the law Eqs. (8) and (9), independently. Once a local model is convergent, the central agent will pull the actor parameters of this model. When every agent has finished the local training, the central agent will integrates the parameters, and generates a global actor network. Finally, the central agent pushes the global model to each agent for next new training episode until the global actor network is convergent. Having trained a convergent actor–critic network, we can select a efficient action based on the output of policy network (e.g., action probability distribution).

5. Implementation and evaluation

In this section, we implement the VCMaker, and evaluate its performance in term of detection accuracy, processing latency and energy consumption. We first give the prototype implementation of VCMaker. Next, we list several state-of-the-art comparing schemes. Finally, we show and analyze the experimental results through numerous real AR videos. Our final results answer the following questions:

Question #1: How to identify the convergence of VCMaker? We track the policy entropy and accumulative discounted reward through 1000 training episodes, and discover that the former (resp. later) factor gradually decreases (resp. increases) and converges to a nonzero value.

Question #2: How does VCMaker perform compared to existing carefully-tuned heuristics? We find that VCMaker achieves a 20.5%–32.8% higher detection accuracy, and 25.2%–45.7% lower energy.

Question #3: How does VCMaker's learning generality when applied in other types of AR videos or other types of bandwidth traces? We find that VCMaker is capable of maintaining a good performance in other scenarios where new network conditions and new videos are added.

[1] The RL-agent takes action a_t for state s_t in the beginning of slot t, then obtains instant reward r_t, and transits to next state s_{t+1}.

Table 1
Actor–Critic network design of VCMaker.

Types	Actor network	Critic network
Input layer	3 × 1D-CNN+VGG16+2	3 × 1D-CNN+VGG16+2
Hidden layer	256 × 256 × 256	256 × 256 × 256
Output layer	$\|action\_space\|$	1

5.1. Prototype implementation

5.1.1. Hardware setup

As Fig. 5 shows, we use a mobile development board Nvidia Jetson TX2 as the mobile AR device, which connects a Hikvision camera as the Magic Leap One AR glass. We emulate an edge cloud with a server (PowerEdge R740) equipped with two NVIDIA GeForce RTX 2080 Ti GPUs. Both the Jetson board and the edge server run an Ubuntu 16.04 OS, and connect to a TP-Link AC1200 router through a WiFi connection.

5.1.2. Training setup

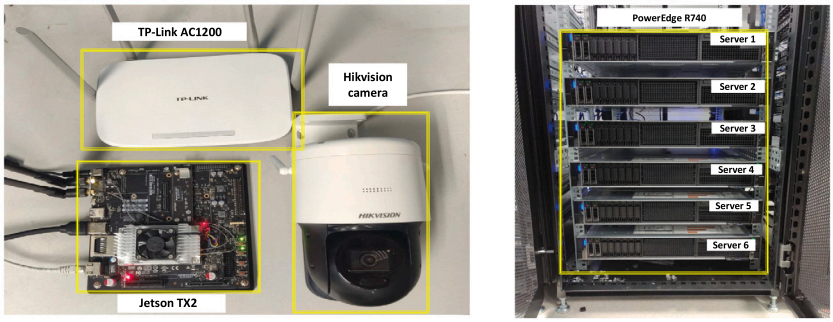
We deploy YOLOv3 detecting algorithm in the edge server, implement VCMaker using libraries on Pytorch [29] and train with the A3C algorithm. VCMaker maintains the Actor and Critic networks, which share the same parameters in the input and hidden layer, and output the probability distribution of each action and the Q-value respectively. Corresponding designs of the network structure are listed in Table 1.

We train the actor–critic networks in a parallel method with multiple workers, each of which calculates gradients locally and independently, and then pushes its gradient to central worker for aggregation synchronously and pulls the global parameters. We adopt the Adam optimizer to run gradient descent, with fixed learning rate 0.0001, mini-batch size of 32 samples per worker, entropy weight $\beta = 0.01$, and reward discount factor $\gamma = 0.9$. We first show VCMaker's convergence in Fig. 6. A larger policy entropy is initially set to perform an exhaustive exploration, but gradually tends to a smaller value with the increase of training episodes, i.e., the policy network is convergent, and VCMaker emphasizes utilization in terms of actions. We find that in the time-varying scenarios, the entropy is not likely to be 0 in order to keep compatible with the newly states. Concurrent with this increase, there has been a spiral rise in the accumulative discounted reward. Due to a random policy is adopted in the initial episodes, VCMaker performs badly in terms of stability. However, through an efficient exploration, VCMaker receives a satisfied and steady accumulative reward.

5.2. Comparing schemes

To improve VCMaker's feasibility, we utilize several advanced techniques as follows: (1) We use sparse optical flow [30] to track the objects appeared in the first and last frame at the same slot, considering that many objects belong to the same class; (2) We extract the complex feature map of the last frame at each slot with the advanced VGG16 [31]; (3) We obtain numerous training experiences (s_t, a_t, r_t, s_{t+1}) in an offline manner, which significantly accelerates the training. To further analyze and evaluate VCMaker's performance, we compare it to the four approaches:

- The baseline solution (Baseline): the mobile clients transmit the raw videos to the edge server, and apply the returned latest detecting results on the current frame.
- The baseline solution and the motion vector-based object detecting (Baseline+MVOD): The mobile clients transmit the raw videos to the edge server, and estimate the object detection result of the current frame using the motion vector extracted from the encoded video frames and the cached latest object detection result.



(a) TX2, router, and camera (b) Edge server

Fig. 5. Hardware setup. We view the Nvidia Jetson TX2 as the mobile AR device.

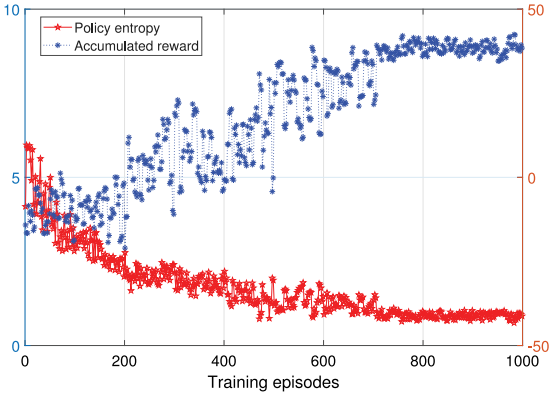


Fig. 6. The entropy and accumulative reward.

- Bandwidth Based Adaptation (BBA). For state s_t, the encoder first finds out all possible combinations of fps and resolution that match the estimated bandwidth B^t_{est}, i.e., $\{(res^{in}, res^{out}, fps) | res \times fps \approx B^t_{est}\}$, $res = res^{in} \times \beta_1 + res^{out} \times \beta_2$, where β_1 and β_2 is the area rate that is inside and outside the RoI of the last frame. We randomly choose one configuration from this set.

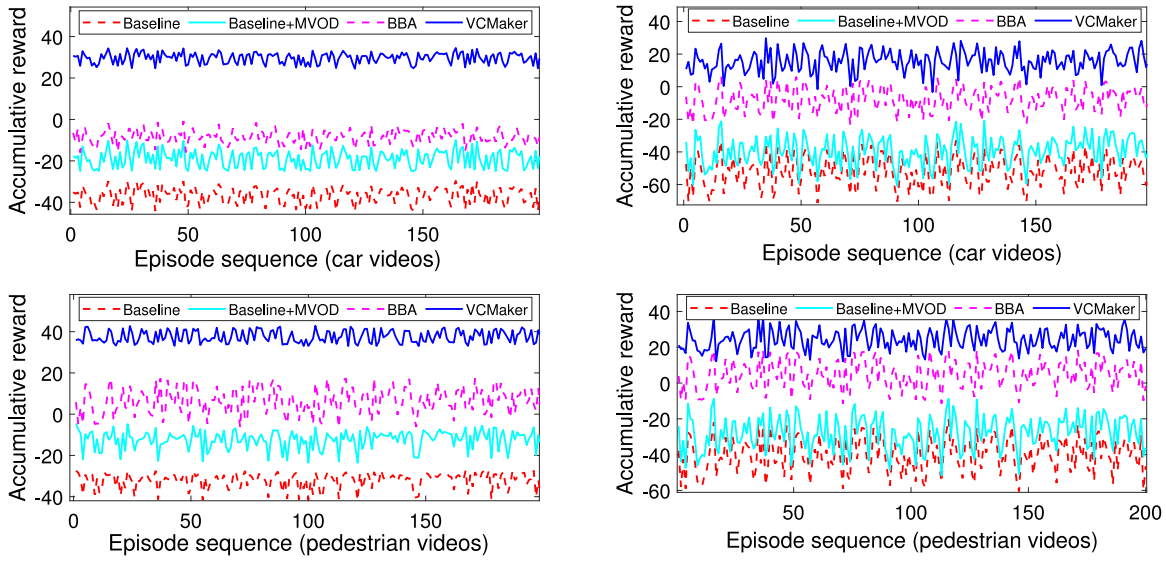
5.3. Experimental results and analyses

Two types of live videos, captured by fixed street cameras for monitoring high-speed moving cars and on-board mobile cameras for tracking low-speed walking pedestrians respectively, are used to compare VCMaker to other schemes. We refer to these two types of videos as car videos and pedestrian videos, respectively. Obviously, the tracked targets in the car videos have higher moving speeds than those in the pedestrian videos. We set 200 episodes for these two types of video, and each episode contains 200 slots (seconds), thus a total of 40 thousand seconds.

In practice, VCMaker aims to maximize the long-term benefits, hence the accumulative reward of a whole episode is the most important metric to evaluate the performance of VCMaker. We evaluate this metrics under two types of network condition, e.g., stable network (i.e., LTE) and unstable network (i.e., WiFi). We first consider the former case. The upper half and lower half of Fig. 7(a) show how VCMaker performs in car and pedestrian videos respectively. Lacking the configuration selection process, the Baseline offloads the raw videos to the edge server for detecting, which inevitably causes a long transmission latency, and then decreases the detecting accuracy and increases the energy consumption. Hence, it performs poorly. Baseline+MVOD can largely improve the accuracy. BBA makes best use of the bandwidth, which reduces the transmission latency, and further enhances the detecting accuracy. The proposed VCMaker, taking network condition, velocity and energy consumption into consideration, outperforms the state-of-the-art BBA by roughly 45.6%. It is worth mentioning that VCMaker performs better on videos with pedestrians than those with cars. The reason behind this phenomenon is that the videos with high-speed targets lead to a larger state space (i.e., a wider range of velocity), which increases the training difficulty and finally translates into the reward loss. Under an unstable network, as Fig. 7(b) shows, the accumulative reward shares the same rules, but it has a much more significant fluctuation, which is largely due to the time-varying bandwidth. It causes an unpredictable transmission latency, energy consumption and detection accuracy.

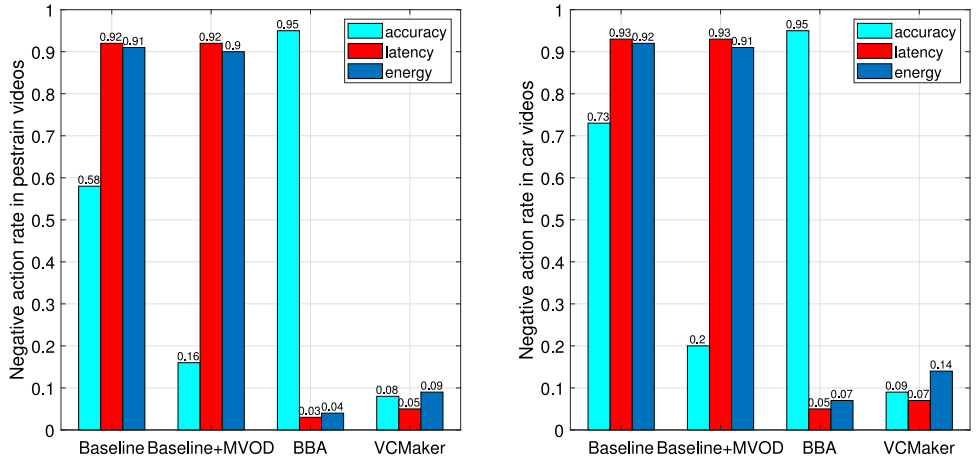
In addition, we are interested in how VCMaker performs in terms of detecting accuracy, latency, and energy consumption comprehensively. For accuracy, we define NAR (i.e., negative action rate) as the ratio of slots, in which the accuracy is lower than the given threshold (i.e., 0.7). The NAR for latency is the ratio of slots in which the total latency including the uploading, detecting, and downloading latencies is larger than the length of a time slot. Similarly, the NAR for energy consumption is the ratio of slots, in which the consumed energy is 40% higher than the max value Jetson TX2 can afford. We evaluate NAR on these two types of videos across 500 episodes under both stable and unstable networks. In the former case, we first calculate the average of accuracy, latency and energy. The results show that VCMaker improves the detection accuracy by 20.5%–32.8%, and reduces the energy consumption by 25.2%–45.7%. Specifically, we calculate the average performance improvements in terms of total latency, detecting accuracy and energy consumption. We find that VCMaker achieves a 32.8% higher detection accuracy compared to Baseline, and 20.5% to the-state-of-the-art scheme BBA. What is more, VCMaker reduces 45.7% energy consumption compared to Baseline, and 25.2% to BBA. Then, we calculate the NARs for these metrics. As Fig. 8(a) shows under a stable network, the proposed VCMaker, which has the lowest NARs on both latency and accuracy, rivals other schemes, indicating that VCMaker can well mitigate the latency–accuracy–energy tradeoff. Baseline shows high NARs in these three metrics, which results from offloading raw videos with expensive configurations. When applied to the videos with cars, as shown in Fig. 8(b), VCMaker still works well. Under an unstable network, as Fig. 9(a) and Fig. 9(b) show, VCMaker still has a gratified performance, which verifies its generalization ability.

VCMaker still has several potential limitations as follows: (1) it lacks of a more representative state space; (2) the mechanism of estimating bandwidth may not be accurate enough in all scenarios; and (3) the deployment of VCMaker in practice is not trivial. In the follow-up work, we plan to estimate future bandwidths using another neural network, explore a more representative state space, train VCMaker with multiple types of videos, and finally deploy it in real AR systems.



(a) Reward under a stable network (b) Reward under an unstable network

Fig. 7. Performance comparison. (a) Accumulative reward using car videos (top half) and pedestrian videos (bottom half) under a stable network; and (b) Accumulative reward using car videos (top half) and pedestrian videos (bottom half) under an unstable network.



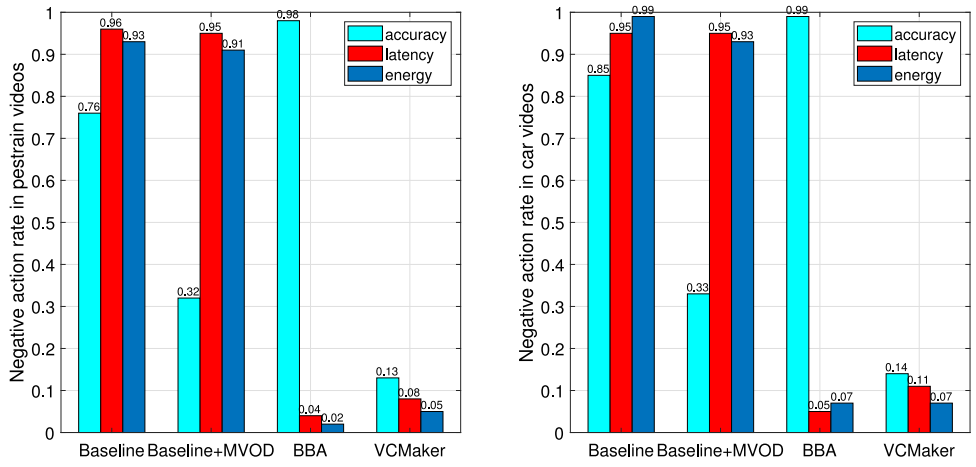
(a) NARs using pedestrian videos under a stable network (b) NARs using car videos under a stable network

Fig. 8. Three types of negative action rates using pedestrian and car videos under stable.

6. Related work

AR applications provides positive experiences for users, however it is costly in terms of processing time, computing resources and energy consumption. Therefore, it promises to upload the detecting computation to cloud. Existing research aims to tradeoff delay and accuracy through efficient offloading or adaptive AR configuration. Liu et al. [32] decoupled the rendering pipeline from the offloading pipeline, and applied a fast objecting tracking method in local device, without considering the AR video configuration selection. Wang et al. [33] formulate the video summarization as an optimization problem and develop an online greedy algorithm EVS to solve it. Liu et al. [34] proposed an edge network orchestrator including two components for server assignment and frame resolution selection in order to mitigate the latency–accuracy tradeoff, but they did not considering the impact of video content. Jiang et al. [35] proposed Chameleon to adaptively pick the best video configuration for learning-based video analytics pipelines, while ignoring the latency and energy. Zhang et al. [36] proposed AWStream to get an optimal profile that models accuracy-bandwidth tradeoff. Chen et al. In comparison, we consider an adaptive video configuration through learning methods from past empirical experiences to solve the above problems. Different from the origin version Cuttlefish [37] we have implemented, VCMaker has the following advantages: (1) VCMaker considers a fine-grained resolution by introducing the define of Region of Interest (RoI); (2) VCMaker optimizes the definition of detecting accuracy; (3) VCMaker runs lightweight motion vector based detection in local device to correct the position of target objects, thereby improving the detecting accuracy; (4) VCMaker takes the energy consumption into consideration.

Recent years, DRL has achieved amazing results in many different fields. Mao et al. [19,20,38] applied DRL to adjust streaming rates under unstable networks, scheduled Spark jobs with efficient resources usage, and proposed open Park for researchers. Mirhoseini et al. [39] adopted DRL to optimize the operator placement for a TensorFlow computation graph in a single machine. In [40], Xu et al. used DRL



(a) NARs using pedestrian videos under an unstable network

(b) NARs using car videos under an unstable network

Fig. 9. Three types of negative action rates using pedestrian and car videos under unstable networks.

to select routing path selection for network traffic. In [41–44], the authors presented a multi-user MEC system, and established a A3C based optimization framework to tackle resource allocation problem for MEC. Zhang et al. [45] proposed ReLeS in Multipath TCP, which provides real-time packet scheduling. To our best knowledge, the proposed VCMaker is the first to apply DRL to realize adaptive configuration for edge-assisted AR applications.

7. Conclusion

In this paper, we design a system that enables high accuracy, low latency and low energy consumption object detection for AR applications. To achieve this, we propose VCMaker, which adaptively learns the best video configuration for transmission and detection from historical experience. We observe the variability of bandwidth, the time-shifted moving velocity of target objects, and the similarity among adjacent frames. All these factors affect the final encoded configuration. Hence, VCMaker takes these factors as input and outputs the configuration distribution, from which, VCMaker selects the best one. In addition, we modify the RoI encoding and motion vector based object tracking to advance VCMaker. We prototype an end-to-end system on commodity hardware, and the results show that VCMaker achieves a 20.5%–32.8% higher detection accuracy, and 25.2%–45.7% lower energy.

CRediT authorship contribution statement

Ning Chen: Conceptualization, Methodology, Software, Writing – original draft. **Sheng Zhang:** Supervision, Writing – review & editing. **Siyi Quan:** Data curation, Software, Validation. **Zhi Ma:** Visualization. **Zhuzhong Qian:** Investigation. **Sanglu Lu:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

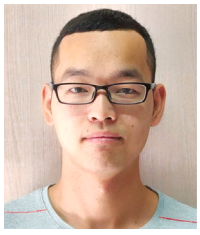
Acknowledgments

We thank the associate editor and anonymous reviewers. This work was supported in part by NSFC (61872175, 61832008), and Collaborative Innovation Center of Novel Software Technology and Industrialization, China.

References

[1] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, MacIntyre, Recent advances in augmented reality, IEEE CGA 21 (6) (2001) 34–47.
[2] G. Westerfield, A. Mitrovic, M. Billinghurst, Intelligent augmented reality training for motherboard assembly, Springer IJAIED 25 (1) (2015) 157–172.
[3] M. Akçayır, G. Akçayır, Advantages and challenges associated with augmented reality for education: A systematic review of the literature, Elsevier ERR 20 (2017) 1–11.
[4] A. Younis, B. Qiu, D. Pompili, Latency-aware hybrid edge cloud framework for mobile augmented reality applications, in: IEEE SECON, 2020, pp. 1–9.
[5] P. Jain, J. Manweiler, R. Roy Choudhury, Overlay: Practical mobile augmented reality, in: MobiSys, Association for Computing Machinery, 2015, pp. 331–344.
[6] P. Jain, J. Manweiler, R. Roy Choudhury, Low bandwidth offload for mobile AR, in: CoNEXT, Association for Computing Machinery, 2016, pp. 237–251.
[7] Vuforia object recognitio, https://library.vuforia.com/articles/Training/Object-Recognition/.
[8] Microsoft hololens, 2020, https://www.microsoft.com/en-us/hololens/.
[9] Magic leap one, 2020, https://www.magicleap.com/.
[10] L.N. Huynh, Y. Lee, R.K. Balan, Deepmon: Mobile gpu-based deep learning framework for continuous vision applications, in: ACM MobiSys, 2017, pp. 82–95.
[11] J. Redmon, A. Farhadi, YOLOv3: An incremental improvement, CoRR (2018) arXiv:1804.02767.
[12] yolov5, https://github.com/ultralytics/yolov5.
[13] O. Alipourfard, H.H. Liu, J. Chen, S. Venkataraman, M. Yu, M. Zhang, Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics, in: USENIX NSDI, 2017, pp. 469–482.
[14] D.N. Hill, H. Nassif, Y. Liu, A. Iyer, S. Vishwanathan, An efficient bandit algorithm for realtime multivariate optimization, in: ACM SIGKDD, 2017, pp. 1813–1821.
[15] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, M.J. Freedman, Live video analytics at scale with approximation and delay-tolerance, in: USENIX NSDI, 2017, pp. 377–392.
[16] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.
[17] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: ACM ICML, 2016.
[18] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, D. Meger, Deep reinforcement learning that matters, in: AAAI AAAI, 2018.
[19] H. Mao, R. Netravali, M. Alizadeh, Neural adaptive video streaming with pensieve, in: ACM SIGCOMM, 2017.
[20] H. Mao, M. Schwarzkopf, S.B. Venkatakrishnan, Z. Meng, M. Alizadeh, Learning scheduling algorithms for data processing clusters, in: ACM SIGCOMM, 2019.
[21] K. Winstein, A. Sivaraman, H. Balakrishnan, Stochastic forecasts achieve high throughput and low delay over cellular networks, in: USENIX NDSI, 2013.
[22] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, H. Balakrishnan, Mahimahi: Accurate record-and-replay for HTTP, in: USENIX ATC, 2015.

[23] H. Wang, J. Xie, User preference based energy-aware mobile AR system with edge computing, in: IEEE INFOCOM, 2020.
[24] J. Chong, N. Satish, B. Catanzaro, K. Ravindran, K. Keutzer, Efficient parallelization of h. 264 decoding with macro block level scheduling, in: ICME, IEEE, 2007, pp. 1874–1877.
[25] S. Craw, Manhattan distance, Springer EMLDM (2017) 790–791.
[26] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, S. Savarese, Generalized intersection over union: A metric and a loss for bounding box regression, in: IEEE CVPR, 2019, pp. 658–666.
[27] R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: NIPS, 2000.
[28] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: NIPS, 2019.
[30] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, T. Brox, Flownet: Learning optical flow with convolutional networks, in: IEEE ICCV, 2015.
[31] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[32] L. Liu, H. Li, M. Gruteser, Edge assisted real-time object detection for mobile augmented reality, in: ACM MobiCom, 2019.
[33] Y. Wang, Y. Dong, S. Guo, Y. Yang, X. Liao, Latency-aware adaptive video summarization for mobile edge clouds, IEEE Trans. Multimed. 22 (5) (2020) 1193–1207.
[34] Q. Liu, S. Huang, J. Opadere, T. Han, An edge network orchestrator for mobile augmented reality, in: IEEE INFOCOM, 2018.
[35] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, I. Stoica, Chameleon: scalable adaptation of video analytics, in: ACM SIGCOMM, 2018.
[36] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, E.A. Lee, Awstream: Adaptive wide-area streaming analytics, in: ACM SIGCOMM, 2018.
[37] N. Chen, S. Quan, S. Zhang, Z. Qian, Y. Jin, J. Wu, W. Li, S. Lu, Cuttlefish: Neural configuration adaptation for video analysis in live augmented reality, IEEE Trans. Parallel Distrib. Syst. 32 (4) (2021) 830–841.
[38] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, R. Addanki, M. Khani, S. He, et al., Park: An open platform for learning augmented computer systems, in: ICML Workshop, 2019.
[39] A. Mirhoseini, H. Pham, Q.V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, J. Dean, Device placement optimization with reinforcement learning, in: ACM ICML, 2017.
[40] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C.H. Liu, D. Yang, Experience-driven networking: A deep reinforcement learning based approach, in: IEEE INFOCOM, 2018.
[41] J. Li, H. Gao, T. Lv, Y. Lu, Deep reinforcement learning based computation offloading and resource allocation for MEC, in: IEEE WCNC, 2018.
[42] Y. He, F.R. Yu, N. Zhao, V.C. Leung, H. Yin, Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach, IEEE CM 55 (12) (2017) 31–37.
[43] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis, Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning, IEEE IoT-J 6 (3) (2019) 4005–4018.
[44] C. Zhang, Z. Liu, B. Gu, K. Yamori, Y. Tanaka, A deep reinforcement learning based approach for cost-and energy-aware multi-flow mobile data offloading, IEEE TCOM E101.B (7) (2018) 1625–1634.
[45] H. Zhang, W. Li, S. Gao, X. Wang, B. Ye, Reles: A neural adaptive multipath scheduler based on deep reinforcement learning, in: IEEE INFOCOM, 2019.



Ning Chen is currently working towards the PhD degree in the Department of Computer Science and Technology, Nanjing University, under the supervision of Prof. Sheng Zhang. His research interests including edge computing, deep reinforcement learning, and video streaming. To date, he has published several papers, including those appeared in ICPADS, SECON, TPDS, Computer Network, et al.

Sheng Zhang is an associate professor in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. He received the BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. His research interests include cloud computing and edge computing. To date, he has published more than 80 papers, including those appeared in TMC, TON, TPDS, TC, MobiHoc, ICDCS, and INFOCOM. He received the Best Paper Award of IEEE ICCCN 2020 and the Best Paper Runner-Up Award of IEEE MASS 2012. He is the recipient of the 2015 ACM China Doctoral Dissertation Nomination Award. He is a member of the IEEE and a senior member of the CCF.

Siyi Quan is an undergraduate student in the Department of Computer Science and Technology, Nanjing University. He is a member of the State Key Lab. for Novel Software Technology. His research interests include distributed computing and edge computing. So far, he has finished SRTP and his paper about blockchain has been accepted by CSCWD 2020.

Zhi Ma received the B.S. degree from the Department of Computer Science and Technology, Nanjing University, in 2017. He is currently working towards the PhD degree with the Department of Computer Science and Technology, Nanjing University, under the supervision of Prof. Sheng Zhang. His research interests include wireless charging and edge computing.

Zhuzhong Qian is a professor at the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Laboratory for Novel Software Technology. He received his PhD. Degree in computer science in 2007. Currently, his research interests include cloud computing, distributed systems, and pervasive computing. He is the chief member of several national research projects on cloud computing and pervasive computing. He has published more than 30 research papers in related fields. He is a member of CCF and IEEE.

Sanglu Lu received her Ph.D. degree in computer science from Nanjing University in 1997. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published over 80 papers in referred journals and conferences in the above areas. She is a member of CCF and IEEE.