Cuttlefish: Neural Configuration Adaptation for Video Analysis in Live Augmented Reality

Ning Chen, Siyi Quan, Sheng Zhang, *Member, IEEE*, Zhuzhong Qian, *Member, IEEE*,
Yibo Jin, *Member, IEEE*, Jie Wu, *Fellow, IEEE*,
Wenzhong Li, *Member, IEEE*, and Sanglu Lu, *Member, IEEE*

Abstract—Instead of relying on remote clouds, today's Augmented Reality (AR) applications usually send videos to nearby edge servers for analysis (such as objection detection) so as to optimize the user's quality of experience (QoE), which is often determined by not only detection latency but also detection accuracy, playback fluency, etc. Therefore, many studies have been conducted to help adaptively choose best video configuration, e.g., resolution and frame per second (fps), based on network bandwidth to further improve QoE. However, we notice that the video content itself has significant impacts on the configuration selection, e.g., the videos with high-speed objects must be encoded with a high fps to meet the user's fluency requirement. In this article, we aim to adaptively select configurations that match the time-varying network condition as well as the video content. We design Cuttlefish, a system that generates video configuration decisions using reinforcement learning (RL). Cuttlefish trains a neural network model that picks a configuration for the next encoding slot based on observations collected by AR devices. Cuttlefish does not rely on any pre-programmed models or specific assumptions on the environments. Instead, it learns to make configuration decisions solely through observations of the resulting performance of historical decisions. Cuttlefish automatically learns the adaptive configuration policy for diverse AR video streams and obtains a gratifying QoE. We compared Cuttlefish to several state-of-the-art bandwidth-based and velocity-based methods using trace-driven and real world experiments. The results show that Cuttlefish achieves a 18.4-25.8 percent higher QoE than the others.

Index Terms—Augmented reality, reinforcement learning, configuration adaption

1 Introduction

Intelligent mobile devices supporting Augmented Reality (AR) become sought after by the masses with diverse requirements. AR is defined as an approach to "augment" the real-world with virtual objects. According to Azuma *et al.* [1], the AR system has the following attributes: to combine real and virtual objects in a real environment; to geometrically align virtual objects and real ones in the real world; to run interactively and in real time. AR technology has been applied to a wide range of fields: tourism, entertainments, marketing, surgery, logistics, manufacturing, maintenance and others [2], [3]. Reports foretasted that 99 million AR/VR devices will be shipped in 2021 [4], and that the market will reach 108 billion dollars [5] by then. Existing mobile AR systems, such as ARKit, Microsoft HoloLens [6] and the announced Magic Leap One [7], facilitate the interaction between humans and the virtual world.

Benefited from the emerged Mobile Edge Computing (MEC) [8], [9], [10], the compute-intensive object detection in AR applications is pushed from remote cloud to edge servers. The AR device uploads the encoded video to the edge server for detecting and rendering, then downloads the well handled video. The AR system on the edge leverages the state-of-the-art detecting algorithms such as YOLO [11], [12], [13] that adopts one state detector strategy that views the object detection as a regression problem and learns the boundary coordinates as well as the corresponding class probability.

However, current AR systems lack effective mechanisms to achieve the adaptive configuration to bridge the performance gap resulting from (1) the fluctuation of network throughout over time; (2) the conflicting Quality of Experience (QoE) requirements (i.e., accuracy and latency of detecting, and fluency of video play); and (3) the time-shifted moving velocities of target objects. Specifically, we take the fps and resolution selection as an example to elaborate the impacts of AR video configuration on user QoE. We divide the total time of interest into multiple slots of equal length, and define the fps as the number of frames per slot. Images with higher resolutions, divided into multiple grid cells in YOLOv3, are likely to enhance the detecting accuracy, but inevitably cause longer transmission delays when fixing the fps. Similarly, AR videos encoded with a high fps may lead to a better fluency without any stutters, but they lead to larger uploading and detecting delays. When the network bandwidth changes over time, encoding the AR videos with an exorbitant configuration may lead to a deteriorating QoE and degraded network status, but assigning a poor configuration abates the network utilization as well as

- *Ning Chen, Siyi Quan, Sheng Zhang, Zhuzhong Qian, Yibo Jin, Wenzhong Li, and Sanglu Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China. E-mail: {ningc, yibo.jin}@smail.nju.edu.cn, siyiquan2021@gmail.com, {sheng, qzz, lwz, sanglu}@nju.edu.cn.*
- *Jie Wu is with Center for Networked Computing, Temple University, Philadelphia, PA 19122. E-mail: jiewu@temple.edu.*

Manuscript received 27 May 2020; revised 18 Sept. 2020; accepted 27 Oct. 2020. Date of publication 30 Oct. 2020; date of current version 11 Nov. 2020.
(Corresponding author: Sheng Zhang.)
Recommended for acceptance by Q. Zheng.
Digital Object Identifier no. 10.1109/TPDS.2020.3035044

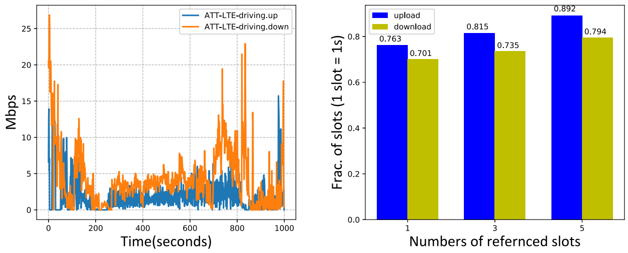
QoE. Apart from the unpredictable network bandwidth, the moving trends of objects in term of moving velocity and direction are also unknown. The video with high-speed objects usually needs a high fps to guarantee the fluency, but a much lower fps is enough if the objects are almost static. Hence, the video configuration should match the time-varying network bandwidth and the moving velocities of the objects in the videos. We will further describe these challenges in Section 2.

In this paper, we pursue a black-box approach for adaptive configuration of AR video that embraces inference while not relying on detailed analytical performance modeling. Encouraged by recent inspiring achievements of deep reinforcement learning (DRL) [14], [15], [16] in the Alpha-go game [17], video streaming [18], and job scheduling [19], we propose and design the learning-based Cuttlefish,[1] an intelligent encoder for adaptive video configuration selection, without using any pre-programmed models or specific assumptions.

Cuttlefish starts with no knowledge and gradually learns to make better configuration decisions through reinforcement, in the form of reward signals that reflect user's QoE from past decisions. Cuttlefish depicts its policy as a neural network that maps "raw" observations (e.g., estimated bandwidth, captured velocity and historical configurations) to the configuration decision for the next slot. The neural network incorporates a rich diversity of observations into the configuration policy in a scalable and expressive way. Cuttlefish aims to maximize the accumulative discounted reward rather than a temporary maximum reward, since a current well-performing configuration may not benefit future configurations. Particularly, Cuttlefish trains its policy network using the state-of-the-art asynchronous advantage actor-critic network model (A3C) [15]. After training over numerous episodes, we can adopt the Cuttlefish to make efficient video configuration decisions.

Our major contributions are summarized as follows:

- We identify several subtle factors to adaptive configuration selection in edge-based video analysis applications. First, time-varying bandwidth constrains the encoded fps and resolution, and time-shifted moving velocity limits the encoded fps. Besides, current information may be instructive for future configuration selection. Last but not the least, diverse personalized QoEs usually lead to a latency-accuracy-fluency tradeoff. We combine these factors to make configuration decisions, which has not been revealed in the existing literature.
- We present Cuttlefish, an intelligent system that learns an adaptive configuration policy from past traces. We train the Cuttlefish with the A3C algorithm that takes the current observed state (estimated bandwidth, captured speed of target objects, *et al.*) as input, and outputs the probability distribution of all configurations, from which the video encoder selects an optimal configuration that maximizes the accumulative discounted reward.



(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, 3 or 5 slots.

- We implemented a prototype of Cuttlefish. We simulate the bandwidth over a large corpus of network traces, and deploy YOLOv3 in the servers configured with RTX2080 Ti GPUs. We compare Cuttlefish to several state-of-the-art algorithms using diverse types of AR videos, and Cuttlefish rivals or outperforms these algorithms by improving the average QoE by 18.4-25.8 percent.

2 Observations and Challenges

In this section, we expound some vital observations that motivate us to propose Cuttlefish.

2.1 Latency-Accuracy-Fluency Tradeoff

The AR devices upload real-time video stream to edge cloud for detecting and rendering, in which detecting and uploading processes occupy the majority of the total latency. Generally, the detecting accuracy and video play fluency are positively correlated with the encoded resolution and fps. Users embrace accurate and fluent well-crafted AR videos in real time, however, *high detecting accuracy, high perceived fluency and low completion delay are difficult to meet simultaneously*. A video stream encoded with a higher resolution can get a gratifying accuracy, but assigning a resolution that exceeds the available bandwidth may lead to an unbearable uploading delay when facing a degraded network. Similarly, a video with a higher fps gains desired fluency without stutters, but may be followed with an extra latency due to detecting and uploading more frames. Hence, a well-balanced QoE is urgently-needed to mitigate this tradeoff.

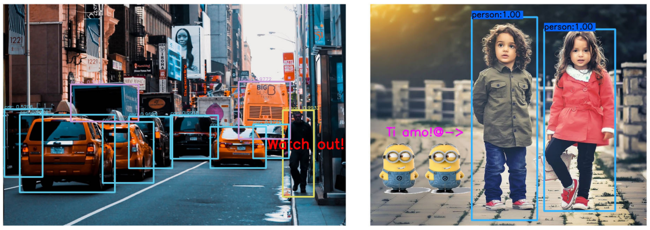
2.2 Variability in Network Bandwidth

Many AR applications are initiated from mobile devices over cellular networks like LTE, which may experience frequent bandwidth fluctuation [20]. To cater to the variable bandwidth, the encoded resolution and fps should be selected adaptively. To illustrate the variability of bandwidth, we depict two ATT-LTE network traces from the Mahimahi [21] project as Fig. 1 shows.

Across the upload and download traces, we made the following observations:

- Periods of extreme low/high throughout are uncommon: only 14.5 percent of the time, the upload

[1] Cuttlefish, sometimes referred to as "chameleons of the sea", can rapidly alter their skin color to camouflage themselves.



(a) Safe driving

(b) Virtual Minions stands next to the boy

Fig. 2. AR video with diverse velocities. (a) Detecting the moving vehicles and pedestrians. (b) Rendering a virtual Minions in a nearly static AR video.

bandwidth is 0 or larger than 10 Mbps, and 14.9 percent for the download bandwidth;

- The bandwidth of the next slot is closely related to the average values of the past several slots: as Fig. 1b shows, for uploading capacity, 76.3 percent slots own less than 20 percent bandwidth variation compared to the previous one slot, and it reaches 89.2 percent when referring to the past five slots;
- The download capacity shares the similar pattern with the upload capacity.

These observations suggest that, the bandwidth fluctuates in a specific range (e.g., [0,28] in Fig. 1) during the whole time scale, but varies less (e.g., [0,5]) in a smaller interval (e.g., from 400 to 410). These observations provide a feasible approach to estimate the bandwidth without future network information, and then guides the resolution selection.

2.3 Time-Shifted Moving Velocity

In real AR scenarios, target objects may not always move fast or keep still as Fig. 2 shows. Fig. 2a depicts the detection of moving vehicles in a dynamic traffic video for pedestrian alert. Apparently, a large encoded fps should be adopted to meet the fluency gap resulting from the dramatic location changes of vehicles and pedestrians over time, but it causes a remarkable rise in both the uploading and detecting latencies. While in Fig. 2b, we render a virtual Minions that stands next to the boy in a relatively static video. In such a scenario, a smaller fps is enough to pledge the fluency. Therefore, not only the bandwidth but also the video content should be taken into consideration when we need to adaptively configure the video for a better QoE.

2.4 Challenges

Intuitively, the video encoding can be viewed as a sequential decision-making process. We select one of the best configurations for the AR video encoder at each time slot. DRL has been widely used for sequential decision-making in an unknown environment, where an agent observes the current state from the environment, selects an action based on the current policy and updates the policy with the feedback (i.e., reward from the environment). Generally, the policy is represented as a neural network trained through numerous trial-and-error interactions with the environment to maximize cumulative reward over time. It seems that we have a practical solution to realize adaptive configuration based on the above three observations. However, it is nontrivial to use DRL in our problem, as indicated by the following knotty challenges:

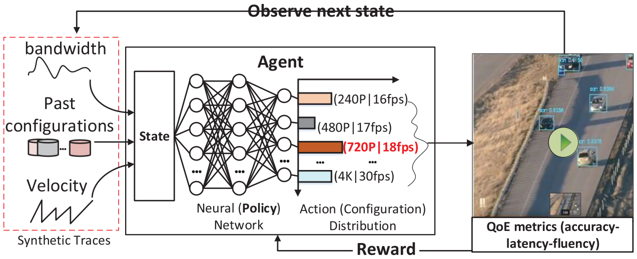


Fig. 3. Offline training. DRL agent pretrains its policy network with past traces collected from interactions with the environment.

- The state, action and reward in DRL are sophisticated. The essence of state, accurate bandwidth and velocity, are difficult to obtain. How to model the estimated real-time bandwidth and capture the moving velocity are unsolved. We try to mitigate latency-accuracy-fluency tradeoff and integrate them into the reward. The reward not only represents the user's real experience for the selected configuration, but also significantly affects the final performance of Cuttlefish. However, the well-crafted reward function is not easy to design.
- Training samples are not readily available. It's impractical to obtain the training data by trial and error in real AR scenarios. How to faithfully model video stream with live AR video player is never trivial.

In the following section, we strive to solve the above challenges and present the design details of Cuttlefish.

3 System Architecture of Cuttlefish

We consider a real-time Augment Reality application with personalized Quality of Experience. The encoded video stream is uploaded to the edge cloud for the object detecting and rendering process, and then sent back for providing the fascinating AR stream for mobile users. To overcome the challenges above, we propose Cuttlefish, a novel system that enables us to adaptively select configuration to achieve a better tradeoff among latency, accuracy and fluency. Cuttlefish leverages a highly representative DRL model rather than a random strategy to choose a valuable configuration. The offline training and online object detecting make up the two fundamental components of Cuttlefish, which is illustrated in Figs. 3 and 4, respectively.

Offline Training. Pure online learning of the policy network from scratch inevitably results in poor policies in the beginning, namely cold start, as DRL typically requires a lot of trials and errors in order to converge to an ideal policy. Thus, the offline training is indispensable to generate a well-designed model to meet the real-time detecting. To overcome the cold start, we collect some expert data to train the policy network through supervised learning. Hence, it can reach better initial parameters compared to the random schemes. As Fig. 3 illustrates, the DRL agent takes the current state observed from the environment including bandwidth, previous frames, moving velocity and previous configurations as inputs, and then generates configuration

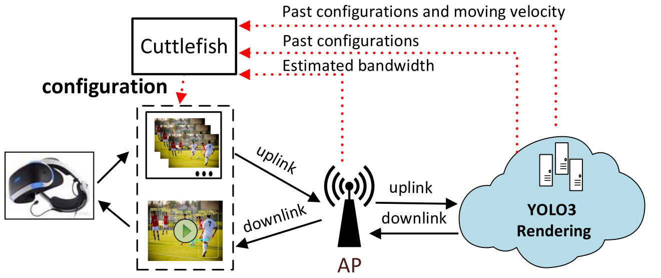


Fig. 4. Online object detecting. Cuttlefish makes adaptive configuration decisions for the real time AR video based on its observed states.

decision. The agent is able to obtain the instant reward, and effectively expand the available trace set for DRL training. Based on the SGD method and several advanced techniques such as the Actor-Critic network and exploration enhancement, the network weights are finally well trained. Note that the DRL model will periodically update its parameters to adapt to the changing environment.

Online Object Detecting. Since the DRL model has been well-trained, we apply them to make configuration decisions for live AR video streams. As Fig. 4 shows, before the AR device encodes the video of the next time slot, it first collects the current state (i.e., estimated bandwidth, captured velocity et al) and extracts the past several configurations. Then, it takes this state as the input of the Actor-Critic network, and outputs the probability distribution of all optional configurations, based on which the AR devices make configuration decisions. Hence, the AR devices encode the video with the selected configuration, upload the AR video to the edge cloud for detecting and rendering, and download the results. During the realtime interaction, Cuttlefish integrates the state, action, instant reward and the next observed state into a quadruple that can be viewed as a newly collected sample, and retrains Cuttlefish's policy periodically to continuously improve the selected configuration over time. In the following section, we show the detailed design.

4 Design Details of Cuttlefish

In this section, we first illustrate the basic learning mechanism, and present the formal definition of our DRL framework. Then, we elaborate on the detailed training methodology.

4.1 Basic Learning Mechanism

DRL is committed to learning an effective policy for the current state from the historical experiences. As depicted in Fig. 3, the RL-agent interacts with the environment, where RL-agent is the brain for making decision, and the environment is a highly abstract that integrates the surrounding information. The RL-agent can observe a small part of the environment, which forms the state. At each time interval 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.

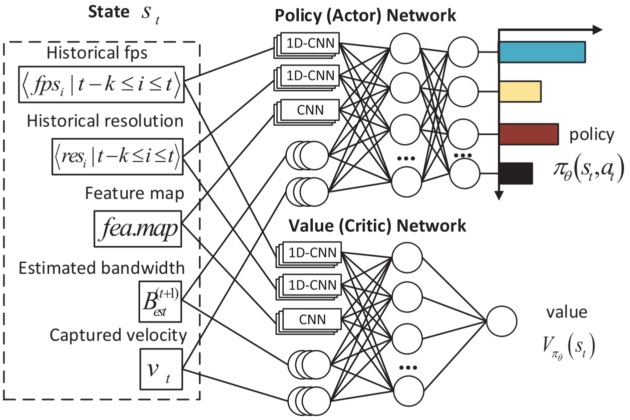


Fig. 5. The neural network architecture of Cuttlefish.

4.2 DRL Framework

We propose the model-free DRL-based Cuttlefish to adaptively generate configurations without any knowledge from the future environment and the state transition probability. The detailed designs and principles are shown in Fig. 5.

4.2.1 State Space

The state is viewed as the observation of a RL-agent (i.e., a MAR device or encoder in making configuration decisions) from the environment. Through continuously learning from historical experience, the RL-agent aims to obtain the comprehensive state that approaches the perspective of the God (i.e., with future and global knowledge). Thus, an exhaustive state is critical to the decision-making efficiency. We take four key elements into consideration, including the followings:

▷ Historical configuration decisions (fps_t, res_t). We divide the total time $\mathcal{T}$ into multiple time slots of equal length. In each slot, we assume the resolution and fps, denoted by res_t and fps_t respectively, are constant. AR video streams do not have subversive changes in two consecutive time slots, so that the past decisions may help in selecting the configuration for the next slot. The number of referenced past configurations used in Cuttlefish, denoted by k shown in Fig. 5, depends on the video contents. For example, if the video is a highly dynamic racing game, one past configuration is enough, while more past configurations may be better in a slightly changeable AR video stream. In practice, choosing an optimal k is not easy, since even a static camera can generate both slightly changeable videos (e.g., midnight) and highly dynamic ones (e.g., rush hours). This is left as our future work, and we believe incorporating it into Cuttlefish can further elevate Cuttlefish.

▷ Estimated bandwidth $B_{est}^{(t+1)}$. The encoder struggles to pick the resolution and fps that perfectly match the available bandwidth, yet lacks the access to gain the future bandwidth. As previously stated, the bandwidth varies around an specific value during solt t, and the more valuable references of past bandwidths it adopts, the more accurate estimation it can get. Thus, we calculate the estimated bandwidth $B_{est}^{(t+1)}$ of slot $t+1$ as the weighted average bandwidth of past k slots, i.e.,

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

where $\omega_i < \omega_j$ if $i < j$, and $\sum \omega_i = 1$.

▷ Average velocity v_t. As we know, AR video stream with target objects of high moving velocities is supposed to be encoded with a higher fps. Similarly, videos with nearly still objects correspond to a lower fps. Assume that the target objects set is $\mathcal{Z} = \{z_1, z_2, \ldots, z_n\}$, and the last configuration is (fps_t, res_t). The encoded video stream is uploaded to the edge cloud for object detecting with the YOLOv3 algorithm, which directly predicts the position of target objects, namely *Bounding Box Prediction*. For $f < fps_t$ and $i \in \mathcal{Z}$, suppose that the *Bounding Box Prediction* set is $(x_i^f, y_i^f, w_i^f, h_i^f, c_i^f)$, which consists of center coordinates of X and Y (x_i^f, y_i^f), height and width (w_i^f, h_i^f), and prediction class c_i^f. Considering the moving trends of target objects are not fixed or regular, we adopt *Manhattan distance* [22] rather than *euclidean distance* to measure the distance that it moves in unit time slot. Thus we define the velocity v_t as the average accumulated distance of all objects moved from the current frame to the next frame during slot t, i.e.,

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

where $\mathcal{Z}$ is the target objects set. Note that some object may disappear at the end of a slot; in this case, we assume its location in the last frame of that slot is in the farthest corner among all four corners from its location in the first frame of the same slot.

▷ Feature map of the latest frame. In a convolution neural network, we'd like to use a network to simulate the characteristics of the visual pathway. Several unseen features, such as the shape edges and color shades, are favorable to decision-making in AR video configuration. Diverse filters are adopted to mine potential knowledge from different perspectives. The detailed design of tuned filters for convolution and pooling are provided in our implementation and evaluation.

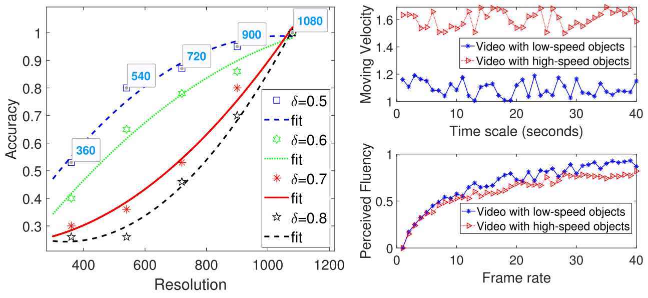
To sum up, we combine the historical configuration decisions, estimated bandwidth of the next slot, and the average velocity of all objects in the past slot into the state space.

4.2.2 Action Space

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 get an instant reward. The policy $\pi_\theta(s, a)$ is the output of *policy network*, whose parameter is set to θ. To improve user's QoE, we aim to make an efficient decision on video configuration. Naturally, we consider two key factors that affect the detecting performance, i.e., the number of frames per slot fps_t and resolution res_t during slot t. We couple these two elements to form the action space, i.e., $a_t = (fps_t, res_t)$.

4.2.3 Reward

The DRL agent is likely to receive an instant reward r_t when applying a_t to state s_t. In practical AR applications, mobile users pursuit high detecting accuracy as well as lower latency and fine fluency, thus we should consider these three metrics in the reward. Suppose that $a_t = (fps_t, res_t)$ during slot t.



(a) Impact of resolution on accuracy (b) Impact of fps on fluency

Fig. 6. Accuracy and fluency. (a) Impact of resolution on accuracy under varying δ. (b) Impact of fps on perceived fluency under diverse velocities.

▷ Latency. As mentioned before, the latency includes uploading delay d_1^t, detecting delay d_2^t, rendering delay d_3^t and downloading delay d_4^t, where d_1^t and d_4^t depend solely upon the available bandwidth, and d_2^t, d_3^t are up to the computing power of edge servers. We make a normalization and denote the total latency d_t of handling the frames at slot t by

$$d_t = \sum_{f=1}^{fps_t} \frac{d_1^t + d_2^t + d_3^t + d_4^t}{d_t^\kappa \, fps_t}, \tag{3}$$

where, d_t^κ is the latency of a single frame with the most expensive resolution (e.g., 1080P in our experiment) at slot t and $d_t \in [0, 1]$. In practice, we calculate d_t^κ as the average latency of each frame using the most expensive resolution.

▷ Detecting accuracy. We adopt F1 score, a harmonic mean of precision and recall, to denote the accuracy. We identify the true positives in the F1 score through a label-based method, which checks if the bounding box has the same label and adequate spatial overlap with the ground truth box [23]. For a specific configuration, we compute accuracy of a single frame by comparing the detected objects with the objects detected by the most expensive configuration. For the frames encoded with configuration (fps_t, res_t) during slot t, the F1 score for frame i is calculated as $F1_i = S_i / S_i^g$, where S_i is the area of the *bounding box* in the ith frame with resolution res_t, and S_i^g is the area of the ground truth box in the ith frame with the most expensive resolution. We define the detecting accuracy c_t at slot t as the fraction of frames whose F1 score $\geq \delta$, e.g.,

$$c_t = \frac{|\{f_i | F1_i \geq \delta, 1 \leq i \leq fps_t\}|}{fps_t}. \tag{4}$$

As Fig. 6a shows, we fix fps, and set diverse δ to see the impact of resolution on accuracy. Through numerous trials, we demonstrate that the accuracy and resolution are positively correlated, which is consistent with our observation. Besides, we find that a much smaller or bigger δ is not likely to obtain a more significant accuracy variation under diverse resolutions. Hence, we select an empirical δ (e.g., 0.7 in Fig. 6a) to reflect this trend in our evaluation.

▷ Fluency. User's perceived fluency is defined as the smooth level of video play. Without loss of generality, configured with a higher fps, the video could gain a better fluency, yet the improvement of fluency will not be significant when the frame rate reaches a certain threshold. As described in Section 2, we observed that the demanded fps towards AR videos with diverse velocities to meet the same fluency is different. For instance in Fig. 2, videos similar to Fig. 2a are proposed to be encoded with a higher fps to reach the same perceived fluency for videos like Fig. 2b. Based on the above observations, we give the formal definition of user's perceived fluency u_t, i.e.,

$$u_t = \frac{v_{max}}{v_t}\log_m(fps_t), \tag{5}$$

where m is the optional maximum fps and v_{max} is the maximum velocity (i.e., the diagonal distance of the frame), $v_t \in [0, v_{max}]$, and $0 < fps_t \leq m$. As illustrated in Fig. 6b, to verify the correctness of Eq. (5) and further explain the impact of fps and moving velocity on user's perceived fluency, we collect two main types of videos from YouTube, including videos with low-speed objects (e.g., pedestrians), and videos with high-speed objects (e.g., cars). Then, we calculate the average moving velocities and the corresponding fluency of these two types of objects in every second based on Eq. (5). Every second on the top half corresponds to a fps on the bottom half. The empirical results also agree with the point we made earlier.

As users may have different preferences on which of the three components is more important, we define the reward r_t of video configuration at slot t by a weighted sum of the aforementioned components, e.g.,

$$r_t = -\alpha_1(d_t - \bar{d}) + \alpha_2(c_t - \bar{c}) + \alpha_3(u_t - \bar{u}), \tag{6}$$

where α_1, α_2 and α_3 are the weight factors to balance the preference to delay, accuracy and fluency, and $\sum_i \alpha_i = 1$. This definition of reward is quite general as it allows us to model varying user preferences on different contributing factors. In practice, to mitigate the diverse fluctuations of these metrics, we set $\bar{d}$, $\bar{c}$ and $\bar{u}$ to the average values of delay, detecting accuracy and perceived fluency respectively, all of which are measured by substantial empirical video traces.

4.3 DRL Model Training Methodology

The configuration space is bounded, but the sophisticated state space seems infinite, thus there are endless (s_t, a_t) pairs. Instead of storing the value of each (s_t, a_t) pair in tabular form, e.g., Q-table, we adopt the state-of-the-art A3C algorithm, which uses a neural network [9] to represent a policy π, and the adjustable parameter of the neural network is referred to as the policy parameter θ. Therefore, we can present the policy as $\pi(a_t|s_t;\theta) \rightarrow [0,1]$, indicating the probability of taking action a_t at state s_t. The objective of DRL is to find a best policy π mapping a state to an action that maximizes the expected accumulative discounted reward as $J(\theta) = \mathbf{E}\left[\sum_{t=t_0}^{t_0+|\mathcal{T}|}\gamma^t r_t\right]$, where t_0 is the current time and $\gamma \in (0,1]$ is a factor to discount the future reward.

4.3.1 Policy Gradient Training

The actor-critic network used by Cuttlefish is trained with *policy gradient method*, whose key idea is to estimate the gradient of the expected total reward by observing the trajectories of executions obtained by following the policy. We highlight the key steps of the algorithm, focusing on the intuition. The policy gradient of $J(\theta)$ with respect to θ, to be used for Policy Network update for slot t, can be calculated as follows [24]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \nabla\theta \log\left(\pi_\theta(s_t, a_t)\right) A^{\pi_\theta}(s_t, a_t)\right], \tag{7}$$

where $A^{\pi_\theta}(s_t, a_t)$ is the advantage function that indicates the gap between the expected accumulative reward when we deterministically select a_t at state s_t following π_θ and the expected reward for actions drawn from policy π_θ. Indeed, the advantage function reflects how much better a current specific action is compared to the "average action" taken based on the policy. Intuitively, we reinforce the actions with positive advantage value $A^{\pi_\theta}(s, a)$, but degrade the actions with negative advantage value $A^{\pi_\theta}(s, a)$.

In particular, the RL-agent extracts a trajectory of configuration decisions and views the empirically computed advantage $A(s_t, a_t)$ as an unbiased estimated $A^{\pi_\theta}(s_t, a_t)$. The update rule of actor network parameter θ follows the policy gradient,

$$\theta \leftarrow \theta + \alpha \sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(s_t, a_t) A(s_t, a_t), \tag{8}$$

where α is the learning rate. The marrow behind this update law is summarized as follows: the gradient direction $\nabla_\theta \log \pi_\theta(s_t, a_t)$ indicates how to change parameter θ to improve $\pi_\theta(s_t, a_t)$ (i.e., the probability of action a_t at state s_t). Eq. (8) goes a step along the gradient descent direction. The specific step size is up to the advantage value $A^{\pi_\theta}(s_t, a_t)$. Hence, the goal of each update is to reinforce actions that empirically have better feedbacks. To compute the advantage value $A(s_t, a_t)$ for a given sample, we need to get the estimated *value function* $v^{\pi_\theta}(s)$, i.e., the total expected reward starting at state s following the policy π_θ. As Fig. 5 shows, the role of *critic network* is to learn an estimated $v^{\pi_\theta}(s)$ from observed rewards. We update the critic network parameters θ_v based on the *Temporal Difference* [25] method,

$$\theta_v \leftarrow \theta_v - \alpha' \sum_t \nabla_{\theta_v}(r_t + \gamma V^{\pi_\theta}(s_{t+1};\theta_v) - V^{\pi_\theta}(s_t;\theta_v))^2, \tag{9}$$

where $V^{\pi_\theta}(s_t;\theta_v)$ is the estimated $v^{\pi_\theta}(s_t)$ that produced by the critic network, and α' is the learning rate. To have a further understanding, we take a specific experience $(s_t, a_t, r_t, s_{t+1})^2$ as an example, we estimate the advantage value $A(s_t, a_t)$ as $r_t + \gamma V^{\pi_\theta}(s_{t+1};\theta_v) - V^{\pi_\theta}(s_t;\theta_v)$. Note that the critic network does nothing to train the actor network other than evaluate the policy of actor network. In actual AR scenarios, only the actor network is involved in making configuration decision.

To realize an adequate exploration for RL agent during training to discover better policies, thereby reducing the risk of falling into suboptimal, we add an *entropy regularization* [15] term to encourage exploration. This practice is significant to help the agent converge to a fine policy. Correspondingly, we modify Eq. (8) to be

[2] 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}.

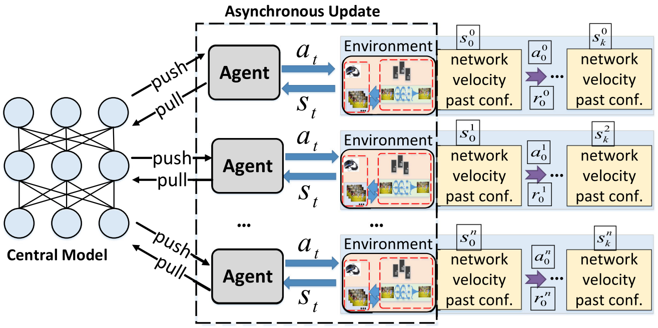


Fig. 7. Parallel training of A3C. A3C adopts multi-thread technology to train the actor-critic network. Each thread, which can be viewed as a RL-agent, trains its own network independently, and interacts with the main network through pull-push mechanism.

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log\pi_\theta(s_t, a_t) A(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(\cdot|s_t)), \tag{10}$$

where β is entropy weight, which set to a large value and decrease over time to allow Cuttlefish to have more opportunity on improving rewards, and $H(\cdot)$ is the policy entropy to encourage exploration by pushing θ in the direction with higher entropy at each time slot.

4.3.2 Parallel Training

To further enhance exploration and speed up training. As shown in Fig. 7, we use a parallel approach to obtain abundant training samples quickly. We start n threads (i.e., agents) at the same time, and adopt diverse environment settings (e.g., diverse network traces and AR videos). Different agents are likely to experience different states and transitions, thus avoiding the correlation. Specifically, each agent continuously collects its samples (i.e., tuple $\{s_t, a_t, r_t, s_{t+1}\}$), and uses the actor-critic algorithm to compute a gradient and perform a gradient descent step as shown in Eqs. (9) and (10), independently. Then, each agent pushes its actor parameters to the central agent, which integrates the parameters, and generates a global actor network. Finally, each agent pulls the global model from central agent, and starts next training episode until the global actor network is convergent. Since the actor-critic network has been well trained, we can take a fast and accurate action based on the action probability distribution for each encoding slot.

5 Implementation and Evaluation

We have implemented Cuttlefish as an intelligent encoder that achieves adaptive configuration for video analysis in AR applications. We first describe the bandwidth simulation in a trace-driven manner. Next, we present Cuttlefish's training settings on the neural network architecture. Finally, we experimentally evaluate Cuttlefish with numerous real live AR videos. Our results answer the following questions:

Question #1. How to verify the convergence of Cuttlefish during training? We track Cuttlefish's policy entropy and accumulative reward across over 2,000 training episodes, and find that the former (resp. later) metric gradually decreases (resp. increases) and finally converges to a non-zero value.

Question #2. How does Cuttlefish perform compared to several carefully-tuned heuristics in term of QoE? We discover that Cuttlefish rivals or outperforms several state-of-the-art schemes, with the average QoE improvements of being 18.4-25.8 percent.

Question #3. Can Cuttlefish's learning generalize to other types of bandwidth traces (e.g., more volatile) or AR videos (e.g., objects move faster)? We find that Cuttlefish is able to maintain a good performance in the face of new network conditions and new videos.

5.1 Trace-Driven Bandwidth and Video Collection

With the aid of local area networks (LANs), the AR devices upload live video to the edge cloud deployed at base stations (BS). However, given the privacy protection, it is impractical to operate the real BSs. Hence, we would like to simulate the LAN that faithfully matches the real scenario. We establish a corpus of network traces by integrating several public datasets or network emulation tool: a broadband dataset provided by the FCC [26] and the tool Mahimahi [21]. The FCC data set consists of over 1 million throughout traces, each of which logs the average throughput over 2,100 seconds at a 5 second granularity. We pick 100 traces for our corpus, each with a duration of 200 seconds, by concatenating randomly sampled traces from "Web browsing" category in the February 2016 collection. The Mahimahi tool generates traces that represent the time-varying capacity of U.S. cellular networks as experienced by a mobile user. Each trace gives a timestamp in milliseconds (from the beginning of the trace) and records the maximum number of 1,500-byte packets it transits at each millisecond. We reformat the throughout trace to match the FCC dataset. Similarly, we generate 100 traces to our corpus, each of which is compatible to the item of FCC traces. During training, unless otherwise noted, we view a trace randomly picked from the corpus as the link bandwidth.

The empirical training datasets of videos, sampled and sifted from the popularized YouTube in an offline method, have the version with the most expensive configuration (i.e., 1080P and 30fps for Cuttlefish). To gain adequate and representative samples, several typical videos, including pedestrians and vehicles, are collected in a large scale across 10 hours, and shares the same length with the bandwidth traces in our corpus. After completing the sampling, we leverage OpenCV [27] to make some preprocessing for the selected videos. We convert the origin video into multiple versions, each of which owns a different resolution. Considering that we make a decision at each time slot, we divide the AR video into multiple chunks of equal length, which is the same as the time slot. Hence, these crafted samples can be adopted to train Cuttlefish's actor-critic network.

5.2 Training Setup

We employ YOLOv3 algorithm for detecting in an edge server (PowerEdge R740, which is configured with NVIDIA GeForce RTX 2080 Ti GPUs). The Actor-Critic networks of Cuttlefish as Fig. 5 shows, implemented using libraries on Pytorch [28] and trained with the A3C algorithm, share the same parameters of the input layer and hidden layer, and output the action distribution and the Q-value. The detailed

TABLE 1
Actor-Critic Network Design of Cuttlefish

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

design of the network architecture is listed in Table 1. We make parallel training of Cuttlefish with multiple workers, each of which gains traces of diverse videos, calculates gradients locally and independently, then pushes the gradient to the central work synchronously, and pulls the aggregated global parameters. We adopt the Adam optimizer to perform gradient descent, with a fixed learning rate of 0.0001, mini-batch size of 32 samples per worker, reward discount factor $\gamma = 0.9$, and entropy weight $\beta = 0.01$. We first verify Cuttlefish's convergence. As illustrated in Fig. 8, a larger policy entropy is set to encourage a deeper exploration in the beginning. While with the increase of training episodes, the policy entropy gradually tends to a smaller value, i.e., the policy network is nearly convergent, and Cuttlefish lays emphasis on utilization toward actions. Note that in the time-varying scenario, to be compatible with the newly generated states, the entropy is not likely to be 0. Concurrent with this increase has been a spiral rise in the accumulative reward. In the initial episodes, as the result of the random policy shows, Cuttlefish performs badly in terms of numerical size and stability. However, through a further exploration, Cuttlefish gains a larger and more steady accumulative reward that fluctuates around the maximum.

Without loss of generality, the available selection range for fps_t and res_t are set to $F = (16, 30)$ and $R = \{480P, 720P, 900P, 1080P\}$, respectively. If only integers are adopted for fps_t and res_t, the total numbers of action can be calculated as $|F| \times |R|$. The bounded action space can greatly lessen the training time. Note that, Cuttlefish can be slightly modified to other ranges of F and R.

5.3 Techniques and Baselines

We utilize several techniques to improve Cuttlefish's utility, including: (1) We adopt sparse optical flow [29] to track the detected objects in the first and last frame of the same slot, considering that there may be multiple objects of the same class; (2) We extract the feature map of the last frame in each slot through the advanced VGG16 [30] directly, rather than a retrained model; (3) We obtain numerous experiences (s_t, a_t, r_t, s_{t+1}) in an offline method, which accelerates the training speed significantly. To further evaluate Cuttlefish's performance, we compare it to the following four schemes:

- None-Adaptation. For each slot, the encoder randomly selects a configuration (i.e., resolution and fps) without consideration on the available bandwidth or velocity.
- Bandwidth Based Adaptation [31]. For each observed state s_t, the encoder first finds out all possible combination of fps and resolution that roughly match the estimated bandwidth B^t_{est}, i.e., $\{(res, fps) | res \times fps \approx B^t_{est}\}$. Then, for each configuration, we calculate its reward using Eq. (6). We choose the optimal one with the maximum reward as the configuration.
- Velocity Based Adaptation. We first set the minimum threshold that the fluency must be satisfied. Then, given the v_{max} and v_t, we can calculate the minimum fps_{min} to meet the threshold according to Eq. (2), i.e., the feasible options are $\{(res, fps) | fps \geq fps_{min}\}$, and we pick the configuration with the maximum reward.
- Velocity Bandwidth Joint Adaptation [32]. The tuned decision is ought to conform to the estimated bandwidth, and meets the fluency threshold. We use $\{(res, fps) | res \times fps \approx B^t_{est}, fps \geq fps_{min}\}$ to record the set of possible configurations. Analogously, we select the most valuable configuration.

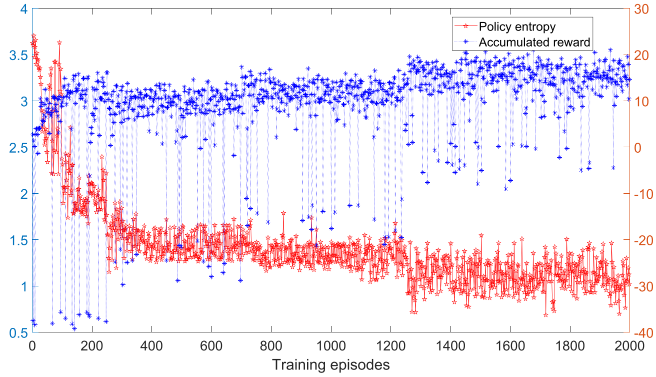


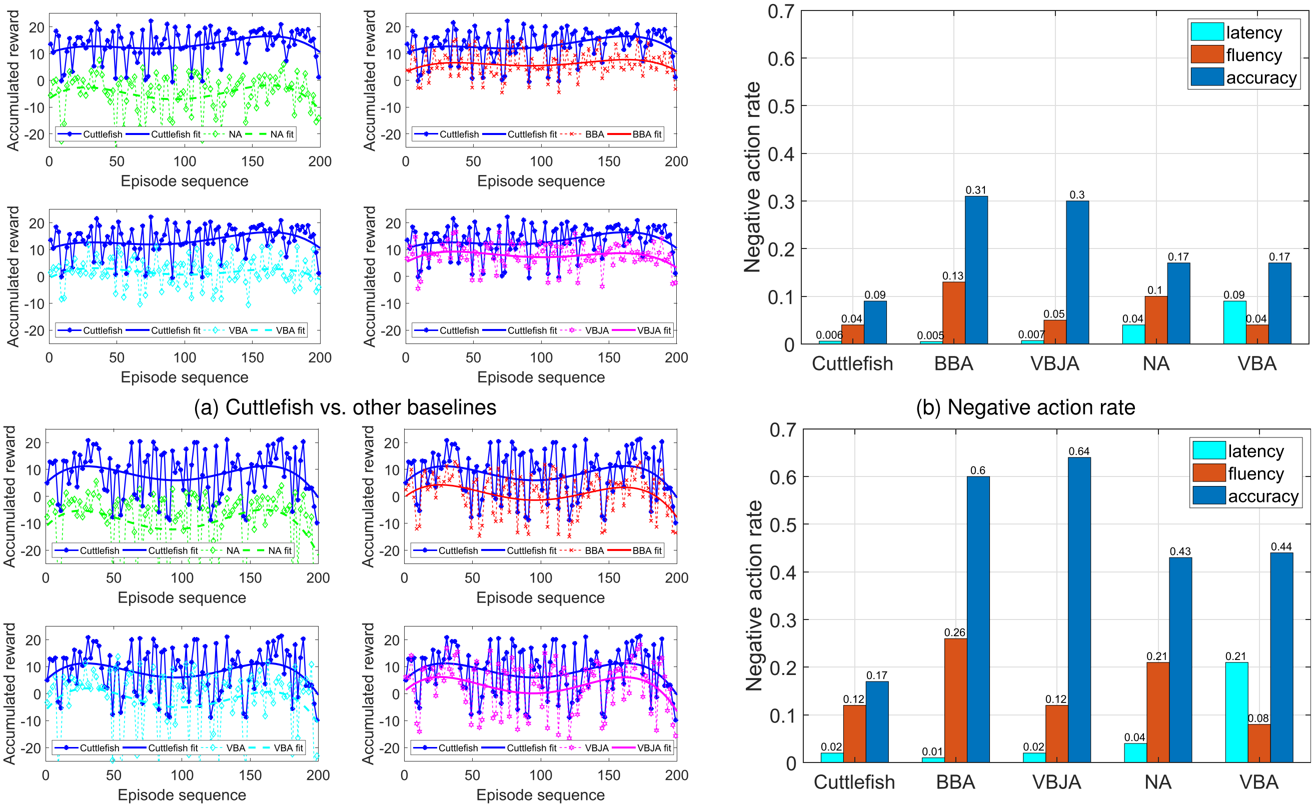
Fig. 8. Policy entropy and accumulative reward over the training episodes.

To simplify the description in the analysis and drawing, we refer to these baselines as NA, BBA, VBA, VBJA, respectively.

5.4 Experimental Results and Analyses

Two typical types of live videos, collected by street fixed cameras for monitoring high-speed cars and on-board mobile cameras for capturing low-speed pedestrians, respectively, are adopted as testing samples to compare Cuttlefish to other baselines. For every type of video we set 200 episodes, each of which contains 200 slots (seconds), i.e., a total of 40 thousand seconds.

In practice, the accumulative reward of a whole episode is the most important metric to evaluate the performance of the proposed model. We first analyze the accumulative reward of Cuttlefish on the videos that mainly consist of pedestrians. As Fig. 9a shows, BBA and VBA take either bandwidth or velocity into consideration, and VBJA emphasizes the instant temporary reward. What's more, to meet the desired fluency, VBA may select a very high fps, which increases the transmission delay, and then decreases the accumulative reward. Hence, their accumulative rewards are inferior to Cuttlefish. Cuttlefish gains a more steady performance enhancement over 40 percent compared to other baselines. We next test Cuttlefish using the car videos, the results of which are shown in Fig. 9c. Compared to the state-of-the-art heuristic VBJA, Cuttlefish still has a significant improvement in the average accumulative reward, which is 18.4 percent. The proposed Cuttlefish, taking network and velocity into consideration, outperforms the



(a) Cuttlefish vs. other baselines

(b) Negative action rate

(c) Cuttlefish vs. other baselines

(d) Negative action rate

Fig. 9. Performance comparisons. We compare Cuttlefish to other baselines in terms of accumulative reward and negative action rate using two testing types of videos (i.e., for capturing pedestrians and cars, respectively), whose experimental results are presented as (a)&(b) and (c)&(d), respectively.

bandwidth-based BBA by roughly 25.8 percent. These inspiring results indicate that Cuttlefish is also capable of processing videos with high-speed targets. It is worth mentioning that Cuttlefish 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, which increases the training difficulty and finally translates into the reward loss.

We are also interested in evaluating how these algorithms perform with respect to latency, accuracy, and fluency, respectively. For latency, we define NAR (short for negative action rate) as the fraction of slots in which the latency (including the uploading, detecting, rendering, and downloading latencies) of a frame is larger than the length of a time slot. For accuracy, the NAR is the fraction of slots in which the accuracy is lower than the threshold (i.e., 0.7). For fluency, the NAR is the fraction of slots in which the perceived fluency is worse than the given value (i.e., 0.7). In our experiments, we simulate the bandwidth with more than 1000 traces, and calculate the NARs across 500 episodes. As Fig. 9b shows, VBA uses a high fps to pursue a good fluency, which leads to a terrible NAR on latency. Similarly, BBA has a large NAR on accuracy. The proposed Cuttlefish, which has the lowest NARs on both latency and accuracy, rivals other baselines, indicating that Cuttlefish can well mitigate the latency-accuracy-fluency tradeoff. When applied to the videos with cars, as shown in Fig. 9d, Cuttlefish still works well, which verifies its generalization ability.

6 Limitations

In this section, we discuss several potential limitations and future research directions.

More Representative State Space. Cuttlefish combines k previous configurations, the estimated bandwidth of the next time slot, and the average velocity of all objects in a slot into the state space. Although Cuttlefish performs well in extensive evaluations, it could generate better configurations if we design specific state spaces for different applications scenarios. For example, the number of previous configurations used in Cuttlefish should depend on specific applications.

Deploying Cuttlefish in Practice. In our current implementation, Cuttlefish runs on the client-side of AR applications. This approach offers several advantages over deployment in edge servers. First, AR clients do not need to send observations to edge servers, which avoids unnecessary information exchange and latency. Second, there is no need to modify edge servers; in other words, this adaptive configuration selection can be transparent to edge servers. Therefore, client-side Cuttlefish can be seen as an overlay on the existing AR applications; whenever there is failure in Cuttlefish, we could disable the configuration selection service and fall back to the default one. This fault recovery mechanism could be invaluable.

7 Related Work

Existing studies on ABR algorithms can be roughly grouped into two classes, i.e., rate-based and buffer-based. Rate-based algorithms [31], [33] first estimate the available network bandwidth based on past several chunk downloads, and then request video chunks at the highest bitrate that matches the bandwidth estimation. For example, Festive [33] predicts throughput in a harmonic mean of the experienced throughput for the past 5 chunk downloads. Apart from the efficient bandwidth utilization and streaming transmission, compared to the traditional ABR problem, our proposed AR adaptive video streaming achieve an efficient object detection.

AR provides helpful information for those things we don't notice or understand, while it's costly in terms of time and computing resources, thus it has to offload the detecting computation to cloud. Existing researches focused on tradeoff between delay and accuracy through intelligent offloading or adaptive configuration. Liu *et al.* [34] observed that RoI changes in the user's view, decoupled the rendering pipeline from the offloading pipeline, and used a fast objecting tracking method locally. Liu *et al.* [35] designed an edge network orchestrator consisting of server assignment and frame resolution selection to mitigate the latency-accuracy trade-off. Jiang *et al.* [36] presented Chameleon to dynamically pick the best configuration for existing NN-based video analytics pipelines. Zhang *et al.* [37] presented AWStream to automatically learns an optimal profile that models accuracy and bandwidth tradeoff. Other mobile AR researches [38], [39] showed their useful insights. In comparison, we explore an adaptive configuration through learning methods from past experiences.

Recently, DRL has achieved promising results in many different domains. Mao *et al.* [18], [19], [40] adopted DRL to adjust streaming rates to cope with unstable network, scheduled Spark jobs with efficient resources usage, and presented Park for researchers to experiment with Reinforcement Learning (RL) for computer systems. Mirhoseini *et al.* [41] used DRL to optimize the operator placement of a TensorFlow computation graph in a single machine. In [42], Xu *et al.* applied DRL for routing path selection in traffic. In [43], [44], [45], [46], the authors considered a multi-user MEC system, and proposed A3C based optimization framework to tackle resource allocation for MEC. Zhang *et al.* [47] proposed ReLeS for Multipath TCP, which supports a real-time packet scheduling. To our best knowledge, Cuttlefish is the first to apply DRL to realize adaptive video configuration.

8 Conclusion

This paper presents Cuttlefish, a deep learning-based system that obtains guaranteed detecting accuracy as well as latency through adaptive configuration. We observed the variability of bandwidth, time-shifted moving velocity of target objects, and similarity among adjacent frames, and all these factors affect the final encoded configuration. Thus, we combine them and propose Cuttlefish with a policy network, which takes the estimated bandwidth, captured velocity and other historical information as input, and output the configuration distribution. We leverage advanced YOLOv3 as the detecting algorithm, and adopt the state-of-the-art A3C model to train Cuttlefish with numerous real traces from YouTube. We compared Cuttlefish to several state-of-the-art bandwidth-based and velocity-based methods. The results shows that Cuttlefish can achieve 18.4-25.8 percent higher QoE. In the follow-up work, we focus on improving Cuttlefish by evaluating more types of videos, and finally implement it in real AR systems.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2017YFB1001801), NSFC (61872175, 61832008), Natural Science Foundation of Jiangsu Province (BK20181252), Jiangsu Key R&D Program (BE2018116), Fundamental Research Funds for the Central Universities (14380060), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and MacIntyre, "Recent advances in augmented reality," *IEEE Comput. Graphics Appl.*, vol. 21, no. 6, pp. 34–47, Nov./Dec. 2001.

[2] G. Westerfield, A. Mitrovic, and M. Billinghurst, "Intelligent augmented reality training for motherboard assembly," *Int. J. Artif. Intell. Educ.*, vol. 25, no. 1, pp. 157–172, 2015.

[3] M. Akçayır and G. Akçayır, "Advantages and challenges associated with augmented reality for education: A systematic review of the literature," *Educ. Res. Rev.*, vol. 20, pp. 1–11, 2017.

[4] Virtual reality and augmented reality device sales to hit 99 million devices in 2021, 2017. [Online]. Available: http://www.capacitymedia.com/Article/3755961/VR-and-AR-device-shipments-to-hit-99m-by-2021

[5] The reality of VR/AR growth, 2017. [Online]. Available: https://techcrunch.com/2017/01/11/the-reality-of-vrar-growth/

[6] Microsoft hololens, 2020. [Online]. Available: https://www.microsoft.com/en-us/hololens/

[7] Magic leap one, 2020. [Online]. Available: https://www.magicleap.com/

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[9] M. Satyanarayanan, "The emergence of edge computing," *IEEE Comput.*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[10] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Gener. Comput. Syst.*, vol. 78, pp. 680–698, 2018.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[12] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 6517–6525.

[13] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv: 1804.02767*[cs.CV]. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2018arXiv180402767R

[14] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015, Art. no. 529.

[15] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[16] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 3207–3222.

[17] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, 2017, Art. no. 354.

[18] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 197–210.

[19] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Interest Group Data Commun.*, 2019, pp. 270–288.

[20] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proc. 10th USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 459–472.

[21] R. Netravali *et al.*, "Mahimahi: Accurate record-and-replay for HTTP," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2015, pp. 417–429.

[22] S. Craw, “Manhattan distance,” in *Encyclopedia of Machine Learning and Data Mining*. Berlin, Germany: Springer, 2017, pp. 790–791.

[23] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.

[24] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Proc. 12th Int. Conf. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[26] Federal communications commission, 2016. [Online]. Available: https://www.fcc.gov/reports-research/reports/measuring-broadband-americ a/raw-data-measuring-broadband-america-2016

[27] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision With the OpenCV Library*. Sebastopol, CA, USA: O’Reilly Media, 2008.

[28] A. Paszke *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.

[29] A. Dosovitskiy *et al.*, “FlowNet: Learning optical flow with convolutional networks,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2758–2766.

[30] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, *arXiv:1409.1556* [cs.CV]. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2014arXiv1409.1556S

[31] Y. Sun *et al.*, “CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction,” in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 272–285.

[32] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, “A control-theoretic approach for dynamic adaptive video streaming over HTTP,” in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 325–338.

[33] J. Jiang, V. Sekar, and H. Zhang, “Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE,” in *Proc. 8th Int. Conf. Emerg. Netw. Experiments Technol.*, 2012, pp. 97–108.

[34] L. Liu, H. Li, and M. Gruteser, “Edge assisted real-time object detection for mobile augmented reality,” in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, Art. no. 25.

[35] Q. Liu, S. Huang, J. Opadere, and T. Han, “An edge network orchestrator for mobile augmented reality,” in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 756–764.

[36] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, “Chameleon: Scalable adaptation of video analytics,” in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 253–266.

[37] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, “AWStream: Adaptive wide-area streaming analytics,” in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 236–252.

[38] W. Zhang, B. Han, and P. Hui, “On the networking challenges of mobile augmented reality,” in *Proc. Workshop Virtual Reality Augmented Reality Netw.*, 2017, pp. 24–29.

[39] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan, “AVR: Augmented vehicular reality,” in *Proc. 16th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2018, pp. 81–95.

[40] H. Mao *et al.*, “Park: An open platform for learning augmented computer systems,” in *Proc. 36th Int. Conf. Mach. Learn. Workshop*, 2019, pp. 2495–2506.

[41] A. Mirhoseini *et al.*, “Device placement optimization with reinforcement learning,” in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2430–2439.

[42] Z. Xu *et al.*, “Experience-driven networking: A deep reinforcement learning based approach,” in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1871–1879.

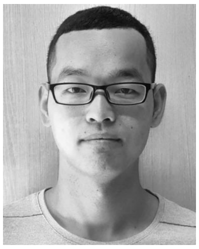
[43] J. Li, H. Gao, T. Lv, and Y. Lu, “Deep reinforcement learning based computation offloading and resource allocation for MEC,” in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2018, pp. 1–6.

[44] Y. He, F. R. Yu, N. Zhao, V. C. Leung, and H. Yin, “Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach,” *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[45] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, “Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[46] C. Zhang, Z. Liu, B. Gu, K. Yamori, and Y. Tanaka, “A deep reinforcement learning based approach for cost-and energy-aware multi-flow mobile data offloading,” *IEEE Trans. Commun.*, vol. E101.B, no. 7, pp. 1625–1634, 2018.

[47] H. Zhang, W. Li, S. Gao, X. Wang, and B. Ye, “ReLeS: A neural adaptive multipath scheduler based on deep reinforcement learning,” in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1648–1656.



Ning Chen received the BS degree from the Chongqing University of Post and Telecommunication, China, in 2018. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Nanjing University, China, under the supervision of Prof. Sheng Zhang. His research interests including edge computing, deep reinforcement learning, and video streaming. His publications include those appeared in IEEE SECON and IEEE ICPADS.

Siyi Quan is currently working toward the undergraduate degree with the Department of Computer Science and Technology, Nanjing University, China. He is a member of the State Key Laboratory 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.

Sheng Zhang (Member, IEEE) received the BS and PhD degrees from Nanjing University, China, in 2008 and 2014, respectively. He is an associate professor with the Department of Computer Science and Technology, Nanjing University, China. He is also a member of the State Key Laboratory for Novel Software Technology. His research interests include cloud computing and edge computing. To date, he has published more than 70 papers, including those appeared in the *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *MobiHoc*, *ICDCS*, *INFOCOM*, *IWQoS*, and *ICPP*. He received the Best Paper Runner-Up Award from IEEE MASS 2012. He is the recipient of the 2015 ACM China Doctoral Dissertation Nomination Award. He is a senior member of CCF.

Zhuzhong Qian (Member, IEEE) received the PhD degree from Nanjing University, China, in 2007. He is an associate professor with the Department of Computer Science and Technology, Nanjing University, China. His current research interests include distributed systems and data center networking. He has published more than 40 papers in referred journals and conferences, including the *IEEE Transactions on Parallel and Distributed Systems*, INFOCOM, and IPDPS.

Yibo Jin (Member, IEEE) received the BS degree from Nanjing University, China, in 2017, where he is currently working toward the PhD degree. He was a visiting student with the Hong Kong Polytechnic University, Hong Kong, in 2017. His research interests include big data analytics, edge computing, and federated learning.

Jie Wu (Fellow, IEEE) is the director of the Center for Networked Computing and Laura H. Carnell professor at Temple University, Philadelphia, Pennsylvania. He also serves as the director of International Affairs at College of Science and Technology. He served as chair of the Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and associate vice provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University, Boca Raton, Florida. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including the *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Service Computing*, *Journal of Parallel and Distributed Computing*, and *Journal of Computer Science and Technology*. He was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society distinguished visitor, ACM distinguished speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). He is a CCF distinguished speaker. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Wenzhong Li (Member, IEEE) received the BS and PhD degrees from Nanjing University, China, both in computer science. He was an Alexander von Humboldt Scholar fellow at the University of Goettingen, Germany. He is currently a full professor with the Department of Computer Science, Nanjing University, China. His research interests include distributed computing, data mining, mobile cloud computing, wireless networks, pervasive computing, and social networks. He has published more than 100 peer-review papers at international conferences and journals, which include INFOCOM, UBICOMP, IJCAI, ACM Multimedia, ICDCS, the *IEEE Communications Magazine*, *IEEE/ACM Transactions on Networking* (ToN), *IEEE Journal on Selected Areas in Communications* (JSAC), *IEEE Transactions on Parallel and Distributed Systems* (TPDS), *IEEE Transactions on Wireless Communications* (TWC), etc. He served as program co-chair of MobiArch 2013 and registration chair of ICNP 2013. He was the TPC member of several international conferences and the reviewer of many journals. He is the principle investigator of three fundings from NSFC, and the co-principle investigator of a China-Europe international research staff exchange program. He is a member of ACM and China Computer Federation (CCF). He was also the winner of the Best Paper Award of ICC 2009 and APNet 2018.

Sanglu Lu (Member, IEEE) received the BS, MS, and PhD degrees from Nanjing University, China, in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor with 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 more than 80 papers in referred journals and conferences in the above areas.