ADACONFIGURE: REINFORCEMENT LEARNING-BASED ADAPTIVE CONFIGURATION FOR VIDEO ANALYTICS SERVICES

Anonymous ICME submission

ABSTRACT

Configuration in video analytics defines parameters including frame rate and image resolution for a video analytics pipeline, thus determining the inference accuracy and computational resource consumption. Traditional solutions to select a configuration are either fixed or periodically adjusted using a brute-force search (i.e., periodically trying all configurations and selecting the one with the best performance), and thus suffer either low inference accuracy or high computation cost. To this end, we propose AdaConfigure that dynamically selects video configuration without resource-consuming exploration. First, we design a reinforcement learning-based agent that adaptively changes the configuration according to the video context, by profiling the video stream using a segmentation mechanism that improves video profiling efficiency. Second, we design a reward function that trades off the inference accuracy and computational resource consumption. Experiments show that our approach outperforms the fixed baseline by up to 35% accuracy improvement or equivalently 50% for computation resource saving.

Index Terms— Adaptive Configuration, Reinforcement Learning, Video Analytics Services

1. INTRODUCTION

A video analytics application consists of a *pipeline* of several video processing modules, typically including a decoder, a selective sampling frame application, and a target detector. Such a pipeline always has multiple *knobs*, such as frame rate, resolution, detector model selection and so on. A combination of the knob values is video analytics *configuration*, and the configuration space grows *exponentially* with the number of knobs and their values [1].

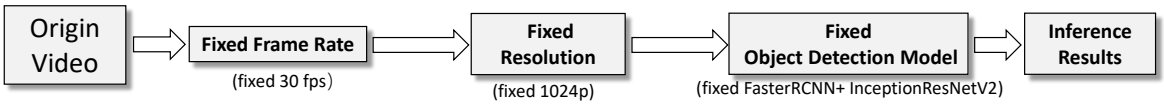
Different configurations directly affect accuracy and resource consumption. The best configuration is the one with the lowest resource demand whose accuracy is over the desired threshold, whcih can optimize the *trade-off* between accuracy and energy consumption. The *best* configuration for video analysis services often varies in minutes or even seconds [1]. As shown in Figure 1(a), if one uses an expensive static configuration (e.g., only profiles the processing pipeline *once* to choose the high frame rate and image resolution) can be precise, but it can also be a huge drain on resource consumption. Similarly, specifying a cheap static configuration (e.g., low resolution and small model) will significantly reduce accuracy. The framework of our solution is shown in Figure 1(b), and we tackle the following design challenges.

Choosing the best configuration is a complicated decision-making problem that is challenging to be solved by rules. The best configuration for video analytics services changes significantly because the exact context of the videos varies over time and across space. For instance, tracking vehicles when traffic moves quickly requires a higher frame rate than when traffic moves slowly. Spatially, the characteristics of video content are different in different locations. For instance, cameras in downtown areas show more cars than the other cameras deployed in the suburbs. It is hard to make an exact rule to choose a configuration for the current context by profiling such complicated video characteristics.

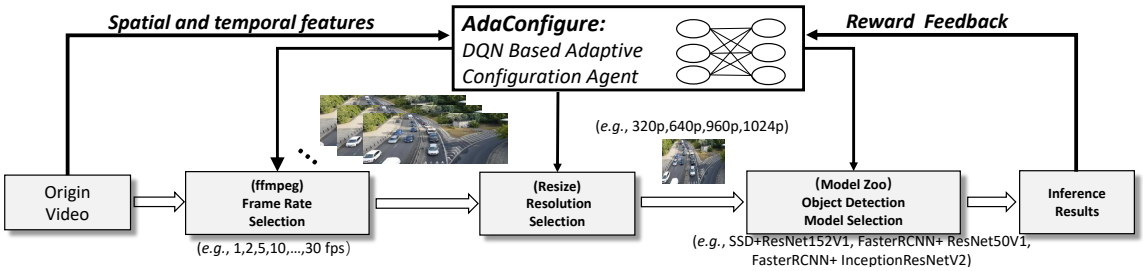
Adaptive configuration would cause a huge extra overhead. The number of possible configurations grows exponentially, and thousands of configurations can be combined with just a few knobs [1]. So exhaustive periodically (e.g., profile once per 4 seconds) configuration to find the best configuration is a highly unrealistic approach because it causes a huge extra overhead, which may exceed the benefits of adaptive configurations. To significantly reduce the resource cost for adaptive configuration requires one solution that automatically chooses a configuration instead of manually trying using various configurations, which is challenging to previous studies including [1, 2], since their approaches pay attention to reduce search space algorithm, and try various configurations in search space to find the best configuration.

In our solution, AdaConfigure can adaptively and automatically select the best configuration according to intrusive dynamics of video context, thus solving this difficult optimal configuration decision problem in a low-cost way. To the best of our knowledge, we are the first to propose an adaptive video configuration solution for such problems. The main contributions of this paper are summarized as follows.

We propose a Deep Q-learning Network-based [3] agent to adaptively pick the best video analytics configuration according to the characteristics of the video stream. For the agent's state, we extract the spatial and temporal features of



(a) Static configuration solution: fixed configuration for video analytics services.



(b) AdaConfigure solution: reinforcement learning-based adaptive configuration framework.

Fig. 1. Comparing to the static solution, our solution can adaptively update the configuration strategy based on the reward feedback.

the video context, so that the agent can adaptively update configuration over time and achieve a superior performance in the multi-camera situation.

We leverage agent's automatic selection characteristic and the video segmentation strategy to reduce profiling cost. In particular, we divide the video into T-second intervals as video chunks and use the agent to choose the best configuration for the first t seconds of the video chunk. It then sticks with the chosen configuration for the rest of the video chunk ($T - t$ seconds) to reduce profiling number. In the evaluation, the profiling cost is about 0.2-2% of the overall video analytics resource consumption.

We design a reward function that considers both inference accuracy and computation resources to assess each configuration's impact. Also, to meet different accuracy-demand services, we leverage the balance factor in the reward function to train different agents. Our evaluation experiments on object detection task show that our approach outperforms baseline: it achieves 10-35% higher accuracy with a similar amount of computation resources or achieves similar accuracy with only 10-50% of the computation resource.

2. RELATED WORKS

2.1. Static Configuration Optimization

Several previous papers have considered optimizing video analytics services by either adjusting the configuration knobs or training specialized NN models. VideoStorm [4] profiles thousands of video analytics queries on live video streams over large clusters, achieving resource-quality tradeoff with multi-dimensional configurations. VideoEdge [5] introduces *dominant demand* to identify the best tradeoff between multiple resources and accuracy, and narrows the search space by identifying a "Pareto ban" of promising configurations. Focus [6] deconstructs video analytics into two phases (i.e., video ingests and video query), achieving an effective and flexible tradeoff of video analytics's latency and accuracy. These algorithms all profile and optimize video analytics only once at the beginning of the video. In their works, video is divided into pictures at a constant frame rate, and the work of video analysis is regarded as a fixed task either. In other words, they do not handle changes in video stream context. But the optimal configurations do change over time because of the complex and changeable video stream context.

2.2. Dynamic Configuration Optimization

Some papers study how to dynamically optimize the configuration for video analytics when the video stream context changes. JCAB [2] jointly optimizes configuration adaption, and bandwidth allocation to address several critical challenges in edge-based video analytics systems, including edge capacity limitation, unknown network variation, intrusive dynamics of video context. Chameleon [1] leverages temporal and spatial correlation to decrease the cost of profiling, and exploits the knob independence to reduce the search space from exponential to linear. Notice these reduce search space algorithms still trying various configurations to find the best configuration. Using different configurations lead to extra expensive profiling costs. To significantly reduce the cost of profiling, we propose an automatic and adaptive configuration algorithm. [7] leverages tabular Q-learning to adapt configuration, but their agent's states only consider inference latency. Our framework pays attention to the characteristics of the video stream and picks the best configuration according to the spatial and temporal features

of the current video stream.

3. REINFORCEMENT LEARNING-BASED ADAPTIVE CONFIGURATION

Figure 2 summarizes how RL can be applied to the adaptive configuration. Briefly, it is a reinforcement learning-based system to train an agent to choose a proper configuration c for one video chunk to inference. We discuss the formulation, agent design, reinforcement learning-based framework, reward feedback in the following subsections. We provide experimental details of all the hyperparameters in Section 4.

3.1. Problem Formulation

To adaptively choose different configuration for video stream, we divides the video into T-second intervals as video chunks, and profiles configurations for each video chunk. Without loss of generality, we denote the object detection service as $\vec{y}_i = M(x_i)$ that provides a predicted result list $\vec{y}_i$ for each input video chunk x_i. It has a baseline output $\vec{y}_{\text{ref}} = M(x_{\text{ref}})$ for each input video chunk $x \in X_{\text{ref}}$ using *reference configuration* (the most expensive configuration). We use this $\vec{y}_{\text{ref}}$ as the ground truth label. For each video chunk x_c that uses a configuration c, the output is $\vec{y}_c = M(x_c)$. Therefore, we have an accuracy metric $\mathcal{A}_c$ by comparing $\vec{y}_{\text{ref}}$ and $\vec{y}_c$. In general, we use the F1 score as the accuracy $\mathcal{A}$ and average GPU processing time per frame as the metric of resource consumption, detailed described in Section 4.2. We also denote the metric of resource consumption as $\hat{s}_{ic}$ that for an input video chunk x_i and a given configuration c. For a reference configuration c_{ref}, the reference resource consumption is $\hat{s}_{\text{ref}}$.

Initially, the agent tries different configurations c to obtain inference results $\vec{y}_c$ from input video chunk x. To obtain object detection results $\{\vec{y}_{\text{ref}}, \vec{y}_c\}$, the agent uses the choosen configuration c and the reference configuration x_{ref}. Comparing the two object detection results $\{\vec{y}_{\text{ref}}, \vec{y}_c\}$ and two resource consumptions $\{\hat{s}_{\text{ref}}, \hat{s}_c\}$, the agent computes the resource consumption ratio $\Delta s = \frac{\hat{s}_c}{\hat{s}_{\text{ref}}}$ and accuracy metric $\mathcal{A}_c$.

3.2. RL Agent Design and Video Characteristic Extraction

The RL agent is expected to give a proper configuration c for minimizing the resource consumption $\hat{s}_c$ while keeping the accuracy $\mathcal{A}$. For the RL agent, the input features are continuous numerical vectors, and the expected output is discrete compression quality level c. Therefore we can use the Deep Q-learning Network [3] as the RL agent. But the naive Deep Q-learning Network can not work well in this task because the state space of reinforcement learning is too large if we directly treat video chunk as the input, making the RL agent extremely difficult to converge.

To address this challenges, we leverage FFmpeg [8] to extract the top k_1 representative images from each chunk and use a pre-trained small neural network to extract the structural characteristics embeddings $\{e_1, e_2, ..., e_{k_1}\}$ of the images as temporal features to reduce the input dimension and accelerate the training procedure. This is a commonly used strategy in training a deep neural network [9, 10]. In this work, we use the early convolution layers of MobileNetV2 [11] as the image feature extractor $\mathcal{E}(\cdot)$ for its efficiency in image recognition. To extract the temporal features of video chunk, we obtain $\{\hat{e}_1, \hat{e}_2, ..., \hat{e}_{k_1-1}\}$ by each embedding subtracting previous embedding. Besides, we record the last k_2 configurations $\{c_1, c_2, ..., c_{k_2}\}$. To solve that vectors of different lengths are not conducive to input to the neural network, we use the fully connected layer to transform the spatial embedding $\{e_1, e_2, ..., e_{k_1}\}$, temporal embedding $\{\hat{e}_1, \hat{e}_2, ..., \hat{e}_{k_1-1}\}$, and recent configuration $\{c_1, c_2, ..., c_{k_2}\}$ to the fixed length vector, similar to the work [12]. We formulate the fixed length vector s as *states* and the configuration c as discrete *actions*.

3.3. Reinforcement Learning-based Framework

In our system, we define k_3 discrete actions to indicate k_3 configuration, and the specific configurations are provided in Section 4.1. We denote the *action-value function* as $Q(s, c; \theta)$ and the optimal compression quality level at time t as $c_t = \text{argmax}_c Q(s, c; \theta)$ where θ indicates the parameters of the Deep Q-learning Network ϕ. In such reinforcement learning formulation, the training phase is to minimize the loss function $L_i(\theta_i) = \mathbb{E}_{s,c\sim\rho(\cdot)}\Big[\big(y_i - Q(s, c; \theta_i)\big)^2\Big]$ that changes at each iteration i where target $y_i = \mathbb{E}_{s'\sim\{\mathcal{X},M\}}\big[r + \gamma \max_{c'} Q(s', c'; \theta_{i-1}) \mid s, c\big]$. Especially, r is the reward feedback, and $\rho(s, c)$ is a probability distribution over state s and the configuration c [3]. When minimizing the distance of *action-value function*'s output $Q(\cdot)$ and target y_i, the *action-value function* $Q(\cdot)$ outputs a more accurate estimation of an action.

In the training phase, the RL agent firstly uses a ϵ-greedy method to take some random trials to observe the environment's reaction and decreases the randomness when training afterward. In iteration t, we input state s_t to neural network ϕ. The RL agent ϕ generates a specific configuration c_t. The framework processes the video chunk x_t using configuration c_t to inference object detection services and obtains reward r_t. Then the framework obtains the next video chunk x_{t+1} and generates the next state s_{t+1}. The framework stores transition (s_t, c_t, r_t, s_{t+1}) in a memory buffer $\mathcal{D}$. All transitions are saved into a memory buffer $\mathcal{D}$, and the agent learns to optimize its *action* by minimizing the loss function L on a mini-batch from $\mathcal{D}$. The training procedure would converge when the agent's randomness

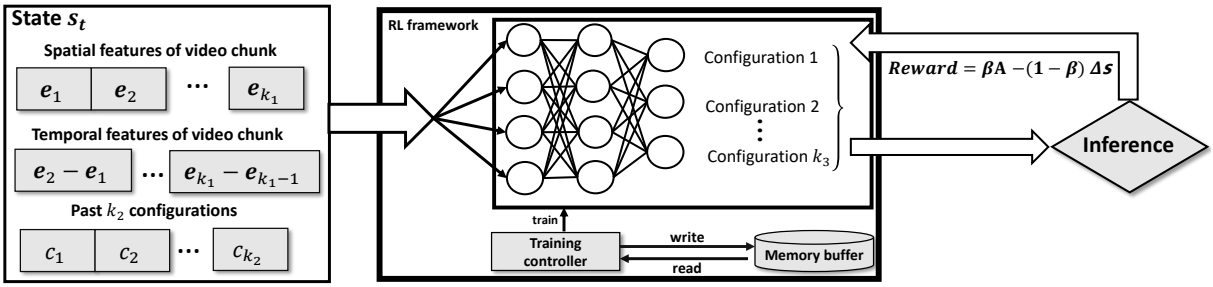


Fig. 2. Applying reinforcement learning to adaptive configuration.

Algorithm 1 Training RL agent ϕ

Initialize action-value function Q with random weights θ, replay memory buffer $\mathcal{D}$ and state s_1
for $t \in 1, 2, \cdots, K$ **do**
  1) Exploration
  With probability ϵ:
    $c_t \leftarrow$ a random valid value
  Otherwise:
    $c_t \leftarrow \text{argmax}_c Q(s_t, c; \theta)$
  2) Reward calcuation
  Process video chunk x_t using configuration c_t to inference
  Obtain $(\vec{y}_{\text{ref}}, \vec{y}_c)$ from the object detection service
  Compute reward $r \leftarrow R(\Delta s, \mathcal{A}_c)$ according to 3.4 Reward Feedback
  3) Gradient descent
  Obtain next video chunk x_{t+1}
  Generate next state s_{t+1}
  $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(s_t, c_t, r_t, s_{t+1}\}$
  Sample a randomly mini-batch of transitions (s_j, c_j, r_j, s_{j+1}) from memory buffer $\mathcal{D}$
  $y_j \leftarrow r_j + \gamma \max_{c'} Q(s_{j+1}, c'; \theta)$
  Perform a gradient descent step on $\big(y_j - Q(s_j, c_j; \theta)\big)^2$ according to [3]
end for

keeps decaying. Finally, the agent's actions are based on its historical "optimal" experiences. The training procedure is presented in Algorithm 1.

3.4. Reward Feedback

In our solution, the agent is trained by the reward feedback. In the above formulation, we define resource consumption rate $\Delta s = \frac{\hat{s}_c}{\hat{s}_{\text{ref}}}$ and accuracy metric $\mathcal{A}_c$ at configuration c. Basically, we want the agent to choose a proper configuration for minimizing the resource consumption while remaining acceptable accuracy. Therefore the overall reward r should be positively correlated with the accuracy $\mathcal{A}$ while negatively with the resource consumption ratio Δs. We introduce a balance factor β to form a linear combination $r = \beta\mathcal{A} - (1-\beta)\Delta s$ as the *reward function* $R(\Delta s, \mathcal{A})$ so that the configuration achieves good accuracy and resource consumption tradeoff.

4. EVALUATION

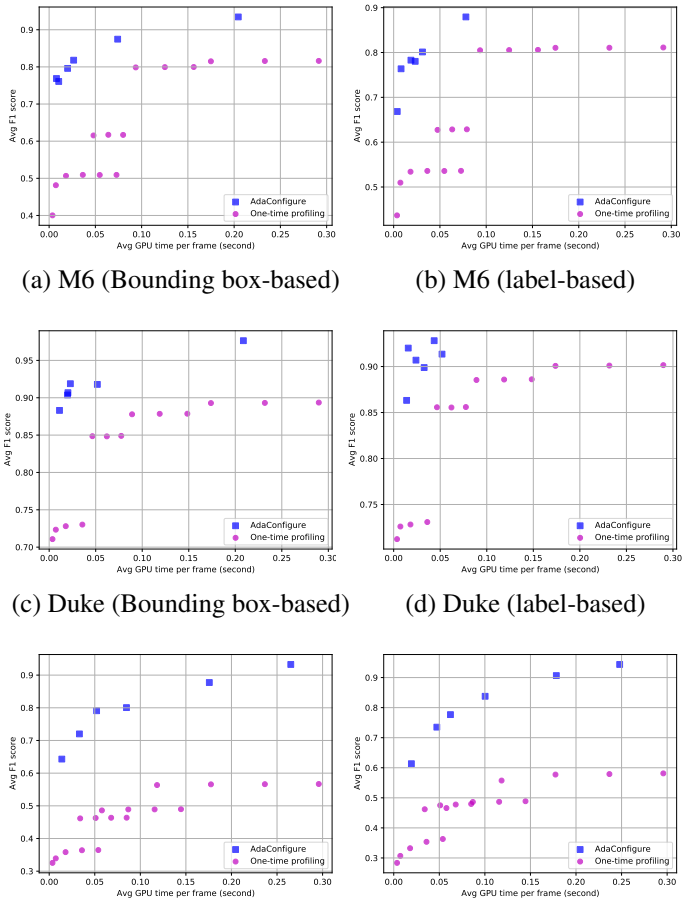
4.1. Experiment Setup

We carry out real-world experiments on the object detection task (basic computer vision services) to verify our solution's performance. We use a desktop PC (Intel Xeon E5-2650 v4 CPU) with four NVIDIA 1080ti graphic card as the server infrastructure, and simulate the environment [13] with three pretrained object detection models in Tensorflow, which are SSD+ResNet152V1, FasterRCNN+ResNet50V1, FasterRCNN+InceptionResNetV2. For each model, two kinds of image resolution, which is 1024×1024 and 640×640, can be picked up. In our experiment, we set the choices space of fps as $\{1, 2, 5, 10, 15, 20, 25, 30\}$. We use FFmpeg to switch the frame rate and image resolution, which decide which frames and in what size they should be fed to the object detection model. The configuration space comprises all possible combinations of values of these knobs, so the environment has 48 configurations in total.

4.2. Datasets and Metric

Most public sources datasets cannot fully satisfy all configuration choices, they provide only a fraction of the requirements for configuration. We try to search keywords (e.g., "highway traffic") on Youtube and download videos that meet the resolution and frame rate requirements. We selecte three datasets: M6, Duke, and Multi-Camera Dataset. M6 is taken from a traffic camera on the longest motorway in Britain. Duke is a video from a fixed camera placed at an intersection, and the traffic flow in the video increases or decreases periodically with the traffic light change. To ensure that AdaConfigure performs better in multi-camera inference, we combine three videos collected from different locations into one video dataset, Multi-Camera Dataset. The exact content of this dataset varies significantly over time and across space.

For metric, we use the F1 score as the accuracy and average GPU processing time per frame as the resource consumption because GPU is the dominant resource for most



(a) M6 (Bounding box-based) (b) M6 (label-based)

(c) Duke (Bounding box-based) (d) Duke (label-based)

(e) Multi-Camera Dataset (Bounding box-based) (f) Multi-Camera Dataset (label-based)

Fig. 3. AdaConfigure (blue) consistently outperforms the baseline of one-time profiling (magenta) across different metrics on different datasets. Each dot represents the results of running each solution.

video processing workloads [1]. F1 score is the harmonic mean of precision and recall, where the precision is true positives divided by the detected objects, and the recall is true positives divided by the ground truth objects. We identify true positives using two conditions, an abounding box-based condition (only check the classified label) and a label-based condition (check the classified label and spatial overlap [14]), consistent with prior work [1, 15]. Both accuracy metrics are useful in real video analytics services and used in our experiments. Besides, to compute the accuracy of a frame that is not sampled by exact configuration, we use the location of objects from the previous sampled frame.

4.3. Experiment Parameters

In the training procedure, we build up to eight independently identical environments for each data set to speed up the training and decrease the data's dependency. For each dataset, we train the agent 5 epochs and 600 steps (5 minutes) per epoch. Some important hyperparameters in our experiments are given in Table 1.

Notation	Value	Notation	Value
ϵ	0.8	k_1	3
γ	0.9	k_2	3
β	0.2,0.3,...,0.7	$T(train)$	1
$t(inference)$	1	$T(inference)$	4

Table 1. Experiment parameters.

4.4. Experiment Results

Figure 3 shows that AdaConfigure consistently outperforms the static configuration baseline (only profiling configurations once) along with resource consumption and two accuracy metrics on different datasets. Each magenta dot represents one static configuration solution (one-time profiling). These static configuration solutions include some fixed expensive configurations and fixed cheap configurations. Each blue dot represents one AdaConfigure configuration solution, which is dependent on the balance factor β in reward function, presented in Section 3.4. The detailed discussion about different AdaConfigure solutions is in Section 4.4.2. Note that AdaConfigure's resource consumption includes both running the best configuration to get inference results and profiling cost of adaptive configuration, which detailed discussed in 4.4.1.

As shown in Figure 3, Adaconfigure achieves a high accuracy of 10-35% with the same amount of computing resources, which benefits from the adaptive selection of relatively expensive configuration when the video content is complex (e.g., traffic congestion, high-speed vehicles). Also, Adaconfigure achieves a 50-90% reduction in resource consumption while achieving almost the same precision as the baseline, which benefits from the adaptive selection of relatively cheap configurations when the video content is simple (e.g., low-traffic flow).

4.4.1. Low Profiling Cost of Adaptive Configuration

In our solution, the video chunk is passed to the AdaConfigure firstly to estimate the configuration, bringing profiling cost (extra resource consumption) to the whole system. The profiling cost, including the resource of extract k_1 embeddings and the cost of agent choosing actions. We test the average time on 30 hours video of Multi-Camera Dataset, and conclude that the average time of extract one embedding is 0.02s, and the average time of agent choosing action is 0.0006s, which can be ignored. We compute the ratio by dividing profiling time into total inference time as metric of profiling cost. The profiling cost is about 0.2-2% of the overall video analytics resource consumption. The concrete ratio depends on the concrete AdaConfigure configuration solutions, such as the solutions listed in Table 2. For instance,

balance factor β	Avg GPU time per frame (second)	Avg F1 score
0.2	0.01384	0.64307
0.3	0.03314	0.72008
0.4	0.05203	0.79096
0.5	0.08479	0.80056
0.6	0.17570	0.87731
0.7	0.26511	0.93262

Table 2. Resource consumption and bounding box-based F1 for different AdaConfigure solutions.

when using the solution of β=0.7, the profiling cost is only 0.2% of overall resource consumption on this solution; when using the solution of β=0.2, the profiling cost is 1.5%.

4.4.2. Different AdaConfigure Solutions to Meet Different Services

In the training phase, when using different balance factors β in the reward function, we would obtain different agents (different adaptive configuration strategies). In general, the β is bigger, indicating the accuracy is more important relatively, and the agent would choose a more expensive configuration. In our experiment, we set β is 0.2,0.3,...,0.7, and the results of different AdaConfigure solutions on Multi-Camera Dataset are listed in Table 2.

Table 2 shows that when the β increases, the resource consumption and the accuracy of the corresponding solution would increase, indicating the AdaConfigure solutions of big β would choose the more expensive configuration to inference. We can leverage this to train proper configuration strategy for different service demands, for example, using big β for high-accuracy demand services and using small β for low-accuracy demand services.

5. CONCLUSION

This paper proposes a reinforcement learning (RL)-based adaptive video analytics configuration framework that can adapt the best configuration to the characteristics of the video stream with much-reduced profiling cost. Comparing to the static configuration baseline, our solution tremendously improves the analytic accuracy while providing low-resource consumption.

6. REFERENCES

[1] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica, "Chameleon: scalable adaptation of video analytics," in *SIGCOMM*, 2018, pp. 253–266.

[2] Can Wang, Sheng Zhang, Yu Chen, Zhuzhong Qian, Jie Wu, and Mingjun Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *INFOCOM*, 2020, pp. 1–10.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *NSDI*, 2017, pp. 377–392.

[5] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *SEC*. IEEE, 2018, pp. 115–131.

[6] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *OSDI*, 2018, pp. 269–286.

[7] Mauricio Fadel Argerich, Bin Cheng, and Jonathan Fürst, "Reinforcement learning based orchestration for elastic services," in *WF-IoT*. IEEE, 2019, pp. 352–357.

[8] FFmpeg, "Ffmpeg," http://ffmpeg.org/, 2000–2018.

[9] Weifeng Ge and Yizhou Yu, "Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning," in *CVPR*, 2017, pp. 1086–1095.

[10] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever, "Improving language understanding by generative pre-training," *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

[11] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[12] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh, "Neural adaptive video streaming with pensieve," in *SIGCOMM*, 2017, pp. 197–210.

[13] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al., "Speed/accuracy trade-offs for modern convolutional object detectors," in *CVPR*, 2017, pp. 7310–7311.

[14] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[15] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia, "Noscope: optimizing neural network queries over video at scale," *arXiv preprint arXiv:1703.02529*, 2017.