CASVA: Configuration-Adaptive Streaming for Live Video Analytics

Miao Zhang[†], Fangxin Wang[*‡], Jiangchuan Liu[†]
[†] School of Computing Science, Simon Fraser University, Canada
[*] SSE and FNii, The Chinese University of Hong Kong, Shenzhen, China
[‡] Peng Cheng Laboratory, Shenzhen, China
mza94@sfu.ca, wangfangxin@cuhk.edu.cn, jcliu@cs.sfu.ca

***Abstract*—The advent of high-accuracy and resource-intensive deep neural networks (DNNs) has fuelled the development of live video analytics, where camera videos need to be streamed over the network to edge or cloud servers with sufficient computational resources. Although it is promising to strike a balance between available bandwidth and server-side DNN inference accuracy by adjusting video encoding configurations, the influences of fine-grained network and video content dynamics on configuration performance should be addressed. In this paper, we propose CASVA, a Configuration-Adaptive Streaming framework designed for live Video Analytics. The design of CASVA is motivated by our extensive measurements on how video configuration affects its bandwidth requirement and inference accuracy. To handle the complicated dynamics in live video analytics streaming, CASVA trains a deep reinforcement learning model which does not make any assumptions about the environment but learns to make configuration choices through its experiences. A variety of real-world network traces are used to drive the evaluation of CASVA. The results on a multitude of video types and video analytics tasks show the advantages of CASVA over state-of-the-art solutions.**

I. Introduction

Human beings are no longer the only consumers of videos. With the proliferation of *live video analytics* applications [1]–[3], an increasing number of videos are consumed by vision deep neural networks (DNNs) for automated information extraction. Although state-of-the-art DNNs can offer unprecedented accuracy for live video analytics, real-world cameras capturing the videos are not equipped with the matched resources to run these expensive DNNs locally for in-situ analytics [4]. This forces camera videos to be streamed over the network to seek sufficient computational resources. Resource-rich edge or cloud servers have become popular destinations for streaming analytics [1], [3], [5], [6].

Although tremendous video analytics systems have been developed in recent years, few have paid attention to the network streaming optimization problem. For example, systems such as `VideoStorm` [1], `Chameleon` [3], and `Nexus` [7] focus on efficient computational resources management for large-scale streaming analytics in a single data center or cloud, while `Mainstream` [8] focuses on multi-tenant video processing on a fixed-resource edge device. All these systems typically assume that the network resource between cameras and edge or cloud servers is not the bottleneck, and therefore overlook its importance. However, network resources deserve more attention in live video analytics since an increasing number of cameras stream videos over unstable and scarce wireless links and public internet [5].

Several strategies, such as frame filtering [9], frame compression [10], [11], DNN model splitting and approximation [12]–[14], have been proposed to reduce the bandwidth consumption or edge-cloud data transfer overhead in geo-distributed video analytics systems. Unfortunately, these strategies are inclined to be application-specific and only optimize from a particular configuration dimension (e.g., frame rate). Additionally, they tend to use fixed rules to reduce bandwidth consumption instead of adapting to the available bandwidth, which may not achieve strict real-time analytics.

Adapting to the scarce and variable network bandwidth between camera clients and remote servers [5], [9] remains a significant challenge in video analytics streaming. To make things worse, traditional adaptive bitrate streaming (ABR) protocols [15] are not well-suited for live video analytics since they are designed to optimize human-perceived quality of experience (QoE), i.e., enabling humans to watch high-quality videos without interruptions. By contrast, live video analytics pursuit the goal of maximizing DNN-perceived QoE, i.e., maximizing server-side DNN inference accuracy without analysis lags [6]. Unlike picky human viewers, video analytics algorithms can tolerant dropped frames, decayed image quality, or reduced resolution as long as the analytics results are not affected.

With the changed optimization goals, common encoding knobs, such as resolution and frame rate, can play a new role in live video analytics streaming. We refer to a combination of specific knob values as a *configuration*. Recent analytics streaming study [5] has verified the significant influences of configuration choices on bandwidth requirement and inference accuracy. Nevertheless, it relies on the profiling-based method to acquire the relationship between configuration and its performance, which not only incurs prohibitively high profiling costs, but also fails to capture the configuration's performance variation caused by video content dynamics.

In this paper, we present `CASVA`, an efficient Configuration-Adaptive Streaming framework for live Video Analytics.

This research is supported by a Canada NSERC Discovery Grant. Fangxin Wang's work is supported in part by National Natural Science Foundation of China with Grant No. 62102342. Fangxin Wang is the corresponding author.

Video Name	Source	Type	Description
STA1	YouTube Live [16]	stationary traffic camera	A video clip collected on a sunny day
STA2	YouTube Live [17]	stationary traffic camera	A video clip collected on a rainy morning
STA3	YouTube Live [17]	stationary traffic camera	A video clip collected on a sunny morning
DASH1	YouTube [18]	dashcam	Daytime drive in Chicago downtown
DASH2	YouTube [19]	dashcam	Night drive around downtown London

TABLE I: Video dataset used in this paper.

Through continuously configuration tuning, it tries to maximize the server-side DNN inference accuracy while adapting to varying available network bandwidth. In order to motivate the design of CASVA, we conduct extensive measurements with different video analytics models on real-world camera streams. Based on our measurement insights, CASVA trains a deep reinforcement learning (DRL) model to help cameras make adaptive configuration choices. In summary, the contributions of this paper are as follows.

- We take two vision tasks (object detection and semantic segmentation) as case studies to acquire a deep understanding of how encoding configuration affects the DNN-perceived QoE. The results reveal the new opportunities and challenges of configuration adaptation.
- We propose CASVA and design an actor-critic style DRL model to select the configuration for each video segment, which can adapt to the fine-grained network bandwidth and video content dynamics.
- By using a variety of camera videos and network traces, we evaluate the performance of CASVA in a trace-driven way. We compare its performance with two state-of-the-art solutions to show its superiority in improving DNN inference accuracy and reducing upload lags.

The rest of this paper is organized as follows. We first introduce our measurement studies in Section II to motivate the design of CASVA. Then, we describe the configuration-adaptive streaming problem and identify its practical challenges in Section III. We subsequently illustrate the details of our DRL-based solution in Section IV and evaluate its performance with trace-driven experiments in Section V. Related work is discussed in Section VI, and the paper is concluded in Section VII.

II. Measurement and Motivation

A. Measurement Setup

Vision Tasks: In this paper, we illustrate the design ideas of CASVA by two classical vision tasks, *object detection (OD)* [20], [21] and *semantic segmentation (SS)* [22], [23]. They are both basic building blocks of advanced video analytics applications. OD is a good representation of bounding boxes-based vision tasks, while SS is a good representation of pixel-based vision tasks. By default, we employ two popular DNN models to execute the two tasks, i.e., Faster R-CNN [20] for OD and ICNet [23] for SS.

Configuration Knobs: For both OD and SS tasks, we consider three configuration knobs: frame rate (FR) that controls a video from the temporal perspective, resolution (RS) that controls a video from the spatial perspective, and quantization parameter (QP) that controls the image quality. These three knobs are typical video configuration knobs in video analytics systems [5], [24], which can affect both the bitrate and the inference accuracy. Based on these three knobs, we define a *configuration* as a combination of them (FR, QP, RS). In this paper, we consider a variety of FR ({1, 5, 10, 15, 30}), QP ({23, 28, 33, 38, 43}), and RS ({240, 360, 480, 720, 1080}p) choices. There are a total of 125 distinct configurations.

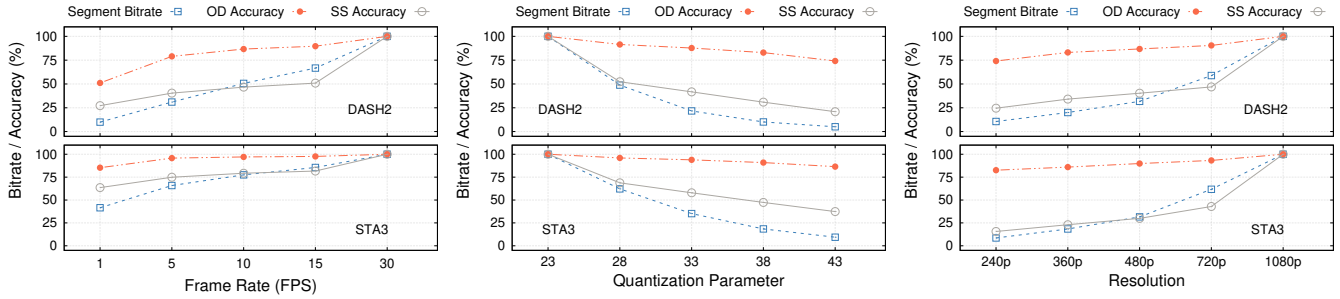
Performance Metrics: For a video segment encoded with a specific configuration, we consider two performance metrics: the minimum bandwidth required for real-time streaming (**bitrate**) and the server-side DNN inference **accuracy**. Concretely, the bitrate is obtained by encoding frames with H.265 and the specified configuration. Unless otherwise noted, the default segment length is 2 seconds, which is the recommended length in various segment-based streaming protocols.

Following the best practices in previous studies [3], [5], [24], instead of human annotations, we regard the DNN outputs of the *golden configuration* (i.e., the most expensive configuration) as the ground truth. In this way, we can minimize the influences caused by the inaccuracy of the DNN model itself and focus on the impacts of encoding configurations. For OD, the accuracy of a configuration is defined as the F1 score of its outputs against the ground-truth results. The true positive is counted if the Intersection over Union (IoU) of the corresponding bounding boxes $\geq 50\%$ and the classified object types match. For SS, we define the accuracy of a configuration as the *Mean of class-wise Intersection over Union (mIoU)* [23] between its outputs and the ground-truth results.

B. Measurement Insights

To identify how video configuration influences video bitrate and server-side DNN inference accuracy, we conduct extensive measurements on a video dataset (shown in TABLE I) collected from real-world cameras. The length of all videos in the dataset is 10 minutes.

We first investigate the influences of decaying one configuration dimension and plot the results in Fig. 1. We find that *different configuration knobs have different impacts on bitrate and accuracy, and such impacts are video-specific and task-specific.* For example, compared with the stationary camera video (STA3 in the figure), reducing FR has more significant impacts on the dashcam video (DASH2 in the figure). Reducing RS leads to dramatic reductions in SS accuracy; in comparison, the impacts of decaying image quality (i.e., increasing the QP value) on SS accuracy are less significant.



(a) Tune FR (RS: 1080p, QP: 23). (b) Tune QP (RS: 1080p, FR: 30). (c) Tune RS (FR: 30, QP: 23).

Fig. 1: The impacts of different configuration knobs on performance. For both video clips, the values of bitrate and accuracy are averaged over all segments and then normalized by that of the golden configuration (FR: 30, QP: 23, RS: 1080p) whose bitrate and accuracy are 100% in all three subfigures.

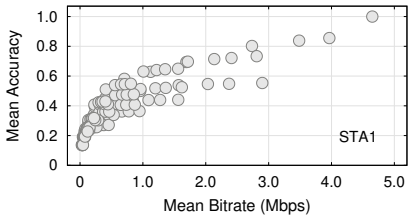


Fig. 2: Performance of all configurations. Values are calculated by averaging across all segments (Task: SS).

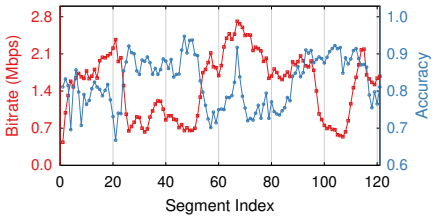


Fig. 3: Performance variations of a single configuration (FR: 10, QP: 28, RS: 720p, Task: OD, Video: DASH1).

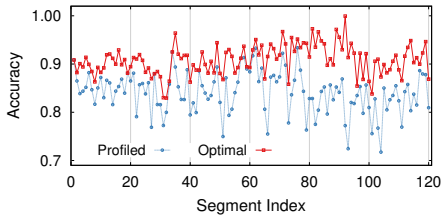


Fig. 4: Performance variations of the profiled and optimal configurations (Task: OD, Video: STA2).

Another interesting observation is that the decreases in image quality and RS dramatically reduce the bitrate with relatively small effects on inference accuracy for the OD task. For instance, raising the QP value of the golden configuration to 43 reduces more than 90% of the bitrate, while the OD accuracy drops less than 26% for both videos. This indicates that, unlike human viewers, DNNs can be less sensitive to image quality. *This also means that live video analytics opens up new opportunities for video streaming, and there is considerable room for further optimization.*

We further adjust all configuration knobs together and examine the bitrate and accuracy distribution of all configurations. Fig. 2 shows the results of analyzing the STA1 video with `ICNet` under various configurations. It can be seen that *higher bitrate does not necessarily lead to higher accuracy, and configurations with similar bitrates can have very different accuracies.* For example, a good configuration (FR: 5, QP: 33, RS: 1080p) reaches a mean accuracy of 0.631 with the mean bitrate of 1.009 Mbps, while streaming the same video with a bad configuration (FR: 10, QP: 23, RS: 480p) requires at least 1.090 Mbps mean bandwidth but only achieves a mean accuracy of 0.439. This indicates that configuration optimization is necessary to achieve the most benefits.

Existing video analytics systems are inclined to ignore the influences of video content dynamics. Yet, it is interesting to quantify the influences of the ever-changing video content on a configuration's performance. An example of this is in Fig. 3, which displays the bitrate and accuracy variations of consecutive video segments encoded with a fixed configuration. As can be seen, video content can affect a configuration's bitrate and accuracy in a fine-grained and highly dynamic manner. This indicates *the relationship between configuration and bitrate (accuracy) is video content-dependent and highly variable.* In addition, we find that the segment bitrate (size) has the potential to be an indicator of video content dynamics. It provides helpful information for predicting the corresponding inference accuracy. For instance, for most cases, an increasing trend in bitrate means increasing video content dynamics, which may lead to configuration performance degradation.

Several video analytics systems [1], [5], [12] rely on one-time offline profiling to obtain resource demands and accuracy of configurations. Further optimization can be performed based on the Pareto frontier [1], [5] of the profiling results. This strategy implicitly assumes that the optimal configuration can maintain its optimality in a certain time window. We conduct the following performance extrapolation experiments to verify the effectiveness of this strategy on our problem.

We first offline acquire the bitrate and accuracy of all configurations on a representative video segment, e.g., the first segment. Assume the available bandwidth during a time window is B. We define the optimal configuration as the most accurate one whose bitrate is not greater than B. We can obtain the *profiled* optimal configuration based on the offline profiling results once B is determined. We then check its performance on subsequent segments. Fig. 4 shows an example where B is 1.5 Mbps. As shown, there exists a considerable performance

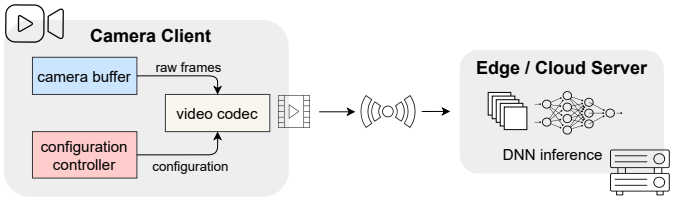


Fig. 5: The framework of configuration-adaptive streaming for video analytics (CASVA).

gap between the *profiled* configuration and the ground-truth optimal configurations. This suggests that profiling-based solutions fail to keep up with the intrinsic dynamics of bandwidth-accuracy trade-off. They may end up with either prolonged latency or lower accuracy. As such, *continually fine-grained configuration adaptation* is necessary.

III. CASVA Design

A. Problem Description

Fig. 5 illustrates our Configuration-Adaptive Streaming framework for live Video Analytics (CASVA). In this framework, frames captured by the camera are cached in a buffer. Then, they are encoded and delivered in segments. Let l denote the duration (in seconds) of a segment. We assume that the camera captures frames in real time and buffers sufficient frames before uploading the next segment, i.e., the upload interval is at least l. With such a design, different segments can be encoded independently with different configurations to allow fined-grained (i.e., at a segment level) control.

The configuration controller in the camera client adjusts the configuration for each segment to achieve both low latency (i.e., preserving data freshness) and high accuracy (i.e., preserving data fidelity). Formally, the goal of the configuration controller is to choose a configuration $\boldsymbol{c}_i$ for each segment i to maximize its server-side inference accuracy $Q(\boldsymbol{c}_i)$ while minimizing its upload lag $\mathcal{L}(\boldsymbol{c}_i)$. The upload lag for a video segment is the time difference between its actual and expected upload time. The upload lag is non-negative, which means if a video segment is uploaded before its expected upload time, we say its upload lag is 0. Assume the start time of a streaming session is 0. In continuous streaming scenarios, the expected upload time of segment i is $i \times l$.

B. Challenges and Motivations

Despite its simplicity, the configuration-adaptive streaming problem has no analytical solutions. We next elaborate on this by discussing several practical challenges.

Low latency and high accuracy are both performance goals pursued by live video analytics streaming. Unfortunately, these two goals are inherently conflicting when the network bandwidth between the camera and the server is dynamic and scarce. As shown in Fig. 2, given the video content, there exists a Pareto frontier. All configurations in this frontier indicate a trade-off between bitrate and accuracy. Higher accuracy only can be achieved at the expense of increased bitrate. In this case, choosing a high-accuracy configuration with a bitrate greater than the available bandwidth may lead to increased lags. Conversely, selecting a low-latency configuration with a bitrate smaller than the bandwidth may sacrifice accuracy. Furthermore, different applications may have different preferences between these two performance goals.

As supported by our measurement results, the DNN inference accuracy $Q(\boldsymbol{c}_i)$ is not fully determined by the configuration $\boldsymbol{c}_i$. It is affected by fine-grained video content dynamics as well. Thus, $Q(\boldsymbol{c}_i)$ cannot be expressed in a closed form of $\boldsymbol{c}_i$. One-time offline profiling does not work since it cannot capture the variations in configuration performance. Frequent online profiling may capture the dynamics, but the camera typically cannot support such profiling due to constrained resources. The server may have sufficient resources, but the prohibitively high profiling costs make such profiling schemes not worthwhile in practice.

To estimate the upload lag $\mathcal{L}(\boldsymbol{c}_i)$, we first need to know the upload delay $u(\boldsymbol{c}_i)$, i.e., how long it takes to upload the video segment. Theoretically, the upload delay can be estimated by $z(\boldsymbol{c}_i)/n(\boldsymbol{c}_i)$, where $z(\boldsymbol{c}_i)$ denotes the video segment size and $n(\boldsymbol{c}_i)$ represents the average network throughput during its uploading. Unfortunately, affected by specific video content, the relationship between $\boldsymbol{c}_i$ and $z(\boldsymbol{c}_i)$ can hardly be modeled by an analytical model. There is no way to obtain the exact value of $z(\boldsymbol{c}_i)$ without actually encoding the segment with configuration $\boldsymbol{c}_i$. However, encoding the video segment with all possible configurations is time-consuming and resource-intensive. On the other hand, the average network throughput $n(\boldsymbol{c}_i)$ is time-variant. It depends on when the camera starts and completes the uploading of segment i and the network conditions during that particular interval. This indicates that $n(\boldsymbol{c}_i)$ is highly dynamic and difficult to be known beforehand. Thus, tuning fine-grained configurations to adapt to dynamic network bandwidth shows significant challenges.

Moreover, in continuous live streaming scenarios, $\mathcal{L}(\boldsymbol{c}_i)$ is not only influenced by segment i's upload time $u(\boldsymbol{c}_i)$, but also the upload time of previous segments. For example, if the upload lag of segment $i-1$ is already 1 second, $\mathcal{L}(\boldsymbol{c}_i)$ will be $max(0, u(\boldsymbol{c}_i) - l + 1)$. This means the configuration choice for a segment can have cascading effects on the camera buffer size, which will further affect the configuration choices of subsequent segments. For instance, streaming a segment with an unnecessarily high-bitrate configuration may increase lags. As a result, subsequent segments may have to be streamed with low-bitrate configurations to maintain freshness, which may sacrifice accuracy. This suggests that the problem is essentially a sequential decision problem.

IV. Solutions with Deep Reinforcement Learning

Deep reinforcement learning (DRL) has achieved great success in solving sequential decision problems with dynamic environments. Our problem is within this realm and can be potentially solved by DRL-based solutions. In this section, we show how CASVA addresses the aforementioned challenges of the configuration-adaptive streaming problem, i.e., how

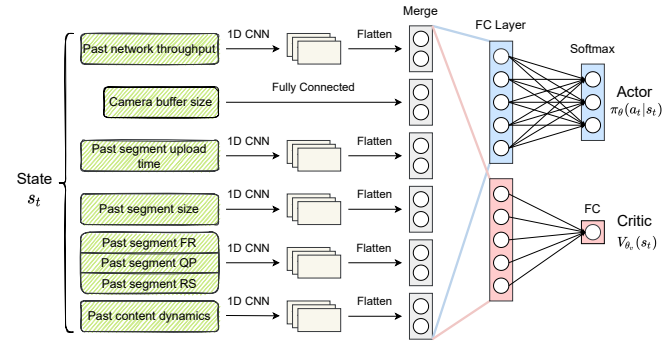


Fig. 6: The NN architecture of CASVA.

to convert this problem into a learning task and devise a DRL-based solution. Different from heuristic solutions that rely on simple models or offline and online profiling, DRL learns to make the most beneficial configuration decisions through the interactions between an *agent* and its *environment*. Specifically, it involves the following key design elements.

A. States: Modeling Environment Dynamics

In DRL, state s_t is a set of observable variables that describe the situations of the environment at the time step t. For our configuration adaptive streaming problem, the state s_t is defined as

$$s_t = (\vec{n_t}, \vec{u_t}, b_t, \vec{f_t}, \vec{q_t}, \vec{r_t}, \vec{z_t}, \vec{d_t}) \tag{1}$$

where $\vec{n_t} = \{n_{t-k}, \cdots, n_{t-1}\}$ is the vector of average network throughput measurements for uploading the past k video segments, which represents recent network conditions. $\vec{u_t} = \{u_{t-k}, \cdots, u_{t-1}\}$ is the upload delay vector of the past k video segments, which is determined by the corresponding segment sizes and network conditions. It has cascading influences on the buffer size. b_t is the buffer size when the camera starts to upload the segment t. $\vec{f_t}$, $\vec{q_t}$, and $\vec{r_t}$ represent the selected FR, QP and RS vectors for the past k video segments, respectively. They provide the configuration information of the past k video segments. Instead of encoding their combinations as a single configuration vector, we choose to encode them independently since they have distinct impacts on the performance goals. $\vec{z_t} = \{z_{t-k}, \cdots, z_{t-1}\}$ is the segment size vector for the past k segments, which is controlled by the configuration choices of the corresponding segments while affecting their upload delays and the camera buffer size.

To quantify the video content dynamics, we introduce the video content dynamics vector $\vec{d_t} = \{d_{t-k}, \cdots, d_{t-1}\}$. Motivated by our measurement insights, the video segment size can be an effective factor in indicating video content dynamics. Therefore, we define the content dynamics d_{t-1} as $(z_t^{t-1} - z_{t-1})/z_{t-1}$, where z_{t-1} is the size of segment $t-1$, and z_t^{t-1} denotes the size of segment t encoded with the configuration used by segment $t-1$. To obtain the value of z_t^{t-1}, the camera needs to additionally encode segment t with segment $t-1$'s selected configuration. Fortunately, this can be done very fast with negligible costs.

B. Policy: Adapting Configuration to Varying Environment

For each input state s_t, the agent determines which action a_t to take, i.e., encoding segment t with which configuration. All possible FR, QP, and RS value combinations constitute the action space. A policy π defines the mapping from states to actions, specifying the probability distribution of all possible actions under a particular state. In our problem, the state space is high-dimensional and infinite since it contains variables that can obtain continuous real values (e.g., network throughput). Neural networks (NNs) are powerful function approximators with high generalization abilities, which are especially useful when handling large state or action spaces. As such, we use a NN to represent the policy.

The *actor* component in Fig. 6 depicts the NN architecture of the policy. For the time-series input type (e.g., past segment upload delays), we utilize 1D CNN layers to extract the underlying features along the time dimension efficiently. Meanwhile, we leverage fully connected layers to process the scalar input type (e.g., current buffer size). Then, all processing results are flattened if necessary and concatenated to a new layer, which is further fed to a fully connected layer to learn the complex relationships between the features. The outputs are finally fed to one softmax layer to calculate the probability distribution of actions. By adjusting the parameters θ of the neural network, the policy π_θ can be optimized accordingly.

C. Policy Optimization: Maximizing DNN-perceived QoE

Optimization goal: At each time step t, by feeding the state s_t to the NN, the agent can choose an action a_t based on the output probability distribution. After applying the chosen action a_t, the agent will receive a scalar reward r_t from the environment while transiting to the next state s_{t+1}. For our problem, the immediate reward r_t for uploading segment t includes three parts (details in Section V-A), and it reflects the quality of the action choice a_t. The goal of DRL is to maximize the expected return $\sum_{t'=0}^{\infty} \gamma^{t'} r_{t+t'}$, where $\gamma \in (0, 1]$ is a factor to discount the future rewards. This is consistent with the goal of configuration controller in Fig. 5, i.e., when encoding each segment, choosing the configuration that can maximize the long-term cumulative DNN-perceived QoE during a streaming session.

Policy gradient training: We choose *policy gradient methods* to train the agent since they do not suffer from the complexity arising from continuous and uncertain states in our problem. In policy gradient methods, the agent directly learns a parameterized policy π_θ based on the gradient of the expected return with respect to the policy parameters θ. The gradient can be typically estimated by $g_t = \mathbb{E}_t[\nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)]$, where $A^{\pi_\theta}(s_t, a_t)$ is the *advantage* function. It represents for state s_t, the advantages of selecting a specific action a_t over the average action taken based on current policy π_θ. By following the popular T-step update method [25], [26], an estimate $\hat{A}_t$ ($t \in [0, T]$) of $A^{\pi_\theta}(s_t, a_t)$ can be calculated by

$$\hat{A}_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t} V_{\theta_v}(s_T) - V_{\theta_v}(s_t) \tag{2}$$

where $V_{\theta_v}(s_t)$ is an estimate of the *value function*, which maps the state s_t to a *state value* that represents the accumulated return the agent can expect to be in s_t by following the policy π_θ. We also use a NN with one linear output layer, corresponding to the *critic* component in Fig. 6, to approximate the value function. The actor and critic have the same input, and following the common practice in [25], we allow them to share the same backbone layers except for the last hidden and output layers. The critic merely helps to train the actor by critiquing the actor's action choices based on the value function [27].

Unfortunately, in practice, the policy gradient estimator g_t can lead to destructively large policy updates and poor sample efficiency [26]. To overcome these challenges, we resort to a dual-clipped *Proximal Policy Optimization (PPO)* [28] method, which instead optimizes the following surrogate objective that can be considered as a first-order approximation of the expected return.

$$\begin{aligned} L_t^{policy}(\theta) &= \mathbb{I}(\hat{A}_t < 0) max(c\hat{A}_t, L_t^p(\theta)) + \mathbb{I}(\hat{A}_t \geq 0) L_t^p(\theta) \\ L_t^p(\theta) &= \hat{\mathbb{E}}_t[min(R_t(\theta)\hat{A}_t,\ clip(R_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \\ R_t(\theta) &= \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \end{aligned} \tag{3}$$

where $\mathbb{I}(\cdot)$ is a binary indicator function. $L_t^p(\theta)$ is the clipped surrogate objective of the vanilla PPO [26]. $R_t(\theta)$ is the probability ratio of the new and old policy. All *min*, *max* and *clip* operators are used to control the step size of the policy update, avoiding the new policy too far away from the old policy. $c > 1$ and $\epsilon > 0$ are hyper-parameters that control the bounds. By default, we set $c = 3$ and $\epsilon = 0.2$ to align with the previous work [28]. The policy optimization objective can be further augmented by a entropy bonus $H(\pi_\theta(\cdot|s_t))$ to encourage exploration [26]. Combining these two terms, during the training process, the policy parameter θ can be updated as

$$\theta \leftarrow \theta + \eta \sum_t \nabla_\theta L_t^{policy}(\theta) + \beta \nabla_\theta H(\pi_\theta(\cdot|s_t)) \tag{4}$$

where η is the learning rate of the actor and β controls the strength of entropy regularization.

Let η' denote the learning rate of the critic. The optimization objective of the critic is to reduce the difference between the predicted and actual state values. It means the critic needs to minimize a squared-error loss of the difference, which can be estimated by the square of $\hat{A}_t$. Therefore, during the training process, the parameter θ_v of the critic can be updated as

$$\theta_v \leftarrow \theta_v - \eta' \sum_t \nabla_{\theta_v} (\hat{A}_t)^2 \tag{5}$$

Training methodology: We implement a distributed version of the dual-clipped PPO to accelerate the training speed and enhance the training performance. Following the parallel training idea in [15], the agent spawns a multitude of workers to collect experiences (i.e., {state, action, reward} tuples) under different parallel environments (i.e., different network traces and camera videos). Every T steps, the collected experiences are sent to the agent, where all of them will be aggregated to update the policy gradient. Subsequently, the updated policy parameters are synchronized by all workers, and a new experience collection iteration starts.

Once the training is complete, CASVA's configuration controller will apply the generated policy to select the configuration for each video segment during video streaming sessions. It should be noted that when applying the trained policy for streaming, no feedback from the server is required for the state transitions, as all variables in the state can be observed solely from the camera client side. This prevents the configuration controller from waiting for server feedback and is beneficial to latency-sensitive live streaming.

V. Performance Evaluation

A. Methodology

Streaming modes: We consider two upload modes. The first mode is *latency-first*. In this mode, the camera client always uploads the up-to-date ready segment. For example, after segment i has been uploaded, the camera skips segment $i+1$ and directly uploads segment $i+2$ if it is the newest ready segment. This strategy is to prevent the lags from accumulating and allow the server-side analytics to catch up with the live scenario for latency-first applications. For this mode, the reward r_t for uploading segment t can be calculated as follows.

$$r_t = \alpha_1 Q_t - \alpha_2\ max(u_t - l,\ 0)\ /\ l - \alpha_3 M_t \tag{6}$$

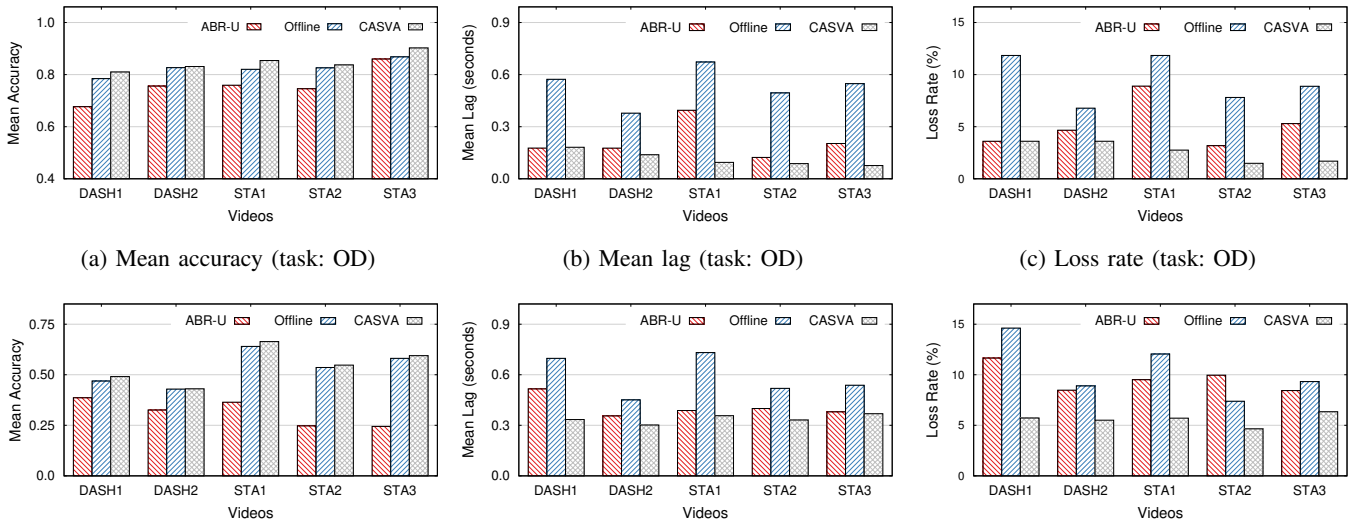
where Q_t is the server-side DNN inference accuracy of segment t, which is used to encourage high-accuracy configuration choices. $u_t = z_t\ /\ n_t$ is the upload delay of segment t, and the second term will penalize configuration choices introducing additional lags. M_t is the number of dropped (unuploaded) segments caused by the upload of segment t, and the third term will further penalize configuration choices leading to segment drops. α_1, α_2, and α_3 are positive coefficients to make a trade-off between these distinct objectives. We set $\alpha_1 = 5$, $\alpha_2 = 1$, and $\alpha_3 = 1$ in this mode through random search.

The second mode is *delivery-first*. In this mode, all segments will be uploaded. As such, the segment lag can be accumulated to a very large value in poor network conditions. This mode can be used when reliable delivery is the first concern. The immediate reward signal r_t in this case is defined as

$$\begin{aligned} r_t = \alpha_1 Q_t &- \alpha_2\ max(u_t - l,\ 0)\ /\ l \\ &+ \alpha_3\ \mathbb{I}(b_{t+1} < b_t)(b_{t+1} - b_t)\ /\ l \end{aligned} \tag{7}$$

where the first two terms are the same as those in the *latency-first* mode. $\mathbb{I}$ is a indicator function that gets 1 when $b_{t+1} < b_t$. Otherwise, it gets 0. b_t represents the camera buffer size when starting to upload segment t, which is used to encourage the agent to choose configurations that are able to catch up with the lags. In this mode, we set $\alpha_1 = 2$, $\alpha_2 = 3$, and $\alpha_3 = 1$ through random search.

Network trace datasets: In order to evaluate the performance of CASVA under real-world network conditions, we consider



(a) Mean accuracy (task: OD) (b) Mean lag (task: OD) (c) Loss rate (task: OD)

(d) Mean accuracy (task: SS) (e) Mean lag (task: SS) (f) Loss rate (task: SS)

Fig. 7: The performance of different methods on various videos in *latency-first* mode (network traces: 4G/LTE).

the following two public datasets, which represent the typical network conditions of networked cameras.

- A fixed broadband dataset provided by FCC [29]. We randomly select $1,500$ traces from the January 2021 collection, and each trace spans 150 seconds. To match the network throughput with the video bitrates, only traces with a minimum throughput above 0.02 Mbps and an average throughput less than 6 Mbps are considered.
- A 4G/LTE bandwidth dataset [30]. Since the average bandwidth throughputs of all traces in the dataset excess the maximum video bitrate, we scale all traces to $(0.02, 6)$ Mbps while maintaining their variances.

For each dataset, we generate a training set by selecting 80% of all network traces at random, and the remaining 20% is used as the test set. In total, more than 13 hours of network traces are used for testing.

Video dataset and vision tasks: We evaluate all videos in Table I. For each video, we regard the first 80% segments as the training set, and the remaining 20% segments are used for testing. Meanwhile, we evaluate two typical video analytics tasks, OD and SS. Specifically, we will show the results of the `Faster R-CNN` model for OD and the results of the `ICNet` model for SS.

Experiment setup: We implement a trace-driven simulator to stream videos segment by segment from a camera client to a server under different network traces, and the average RTT between the camera and the server is 80 ms. The DRL agent is implemented based on a deep learning framework `PyTorch` [31], and its hyper-parameters are tuned empirically.

At each time step t, we consider the past 8 video segments, i.e., k is set to 8. The number of filters for all 1D CNN layers is 128, and the stride is 1. $\vec{n}_t$, $\vec{u}_t$, $\vec{z}_t$, and $\vec{d}_t$ each feed into a 1D CNN layer whose input channel is 1 and kernel size is 4. The 1D CNN layer for past configuration choices has 3 input channels ($\vec{f}_t$, $\vec{q}_t$, and $\vec{r}_t$), and its kernel size is 1. There are 128 neurons in the each fully connected layer. The actor's output dimension is 125, and the critic's output dimension is 1. By default, the learning rates of the actor and critic are both $1e^{-4}$, and T is 59. The entropy coefficient β is originally set as 0.5 and will gradually decay in the training process.

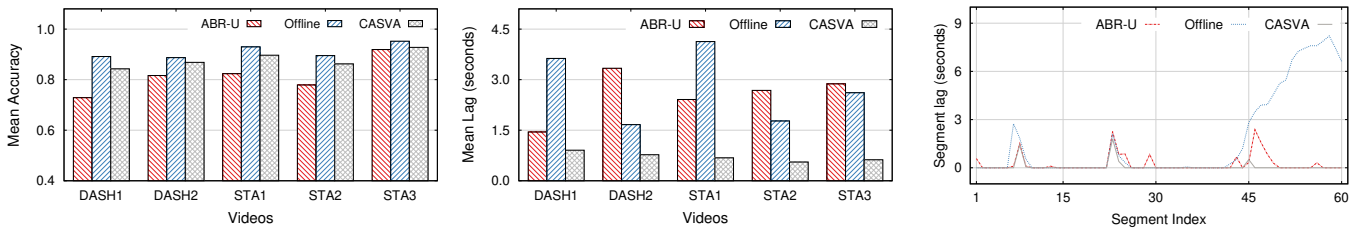
Evaluation metrics: The evaluation metrics are as follows.

- *Mean accuracy.* It is the mean of the server-side DNN inference accuracies of all test video segments. For unuploaded segments, we count the accuracy as 0.
- *Mean lag.* It is the mean of the upload lags of all test video segments.
- *Loss rate.* It is the ratio of unuploaded test video segments to all test video segments.

For the *latency-first* mode, we consider *mean accuracy*, *mean lag* and *loss rate*. For the *delivery-first* mode, we only consider *mean accuracy* and *mean lag*.

Baselines: We compare `CASVA` with the following baselines, which collectively represent the state-of-the-art streaming methods for live video analytics.

1) *ABR-U*: An upload version of `Pensieve` [15]. We implement an upload version of the RL-based adaptive bitrate streaming algorithm for live video analytics. With this method, each video segment will be encoded into one of five candidate bitrates corresponding to five video resolutions ({240, 360, 480, 720, 1080}p). Unlike human viewers, video analytics tasks are insensitive to video quality smoothness. As a result, we also modify the human-perceived QoE metrics in [15] to make this baseline more suitable for video analytics. Specifically, we redesign the immediate reward to optimize bitrate, lag, and loss rate. We leverage this baseline to show the performance of transferring existing ABR algorithms to live video analytics.



(a) Mean accuracy comparison (b) Mean lag comparison (c) Lag variations over time

Fig. 8: The performance of different methods on various videos in *delivery-first* mode (task: OD; network traces: 4G/LTE).

2) *Offline*: This is a popular method to quantify the relationship between resource demands (or accuracy) and configurations [1]. We obtain a profile of {configuration, mean bitrate, mean accruacy} tuples for all configurations by offline profiling all training video segments. The network throughput is estimated by calculating the harmonic mean of the network throughput measures for the past 8 video segments, as described in [15]. Then, based on the profiling results obtained offline, this method selects the most accurate configuration with a mean bitrate not greater than the currently estimated network throughput for each test segment.

B. Evaluation Results

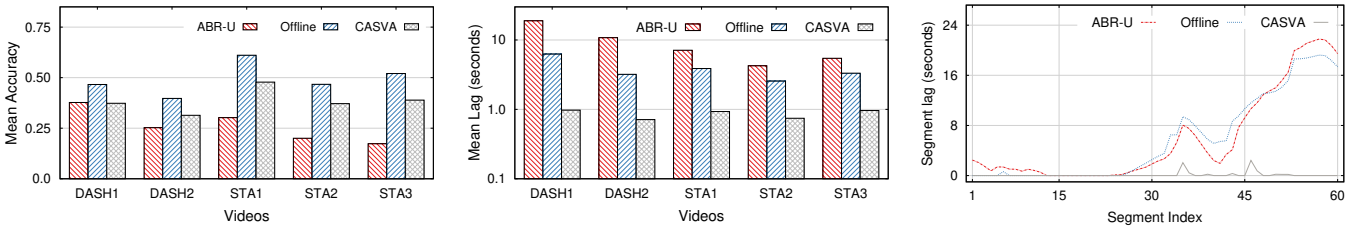
Fig. 7 compares the performance of streaming various videos with different methods for different video analytics tasks in the *latency-first* streaming mode. It can be seen that compared with the *ABR-U* method, the *Offline* method achieves higher mean accuracies but also leads to increased mean lags and segment loss rates. One possible reason is that its network throughput estimation strategy is too simple to handle frequent network throughput fluctuations. Aggressively choosing the most accurate configuration under the constraint of inaccurately estimated network throughput can be vulnerable to the negative influences of configuration performance drifts caused by video content. In contrast, by consciously adapting to the dynamic network conditions, *ABR-U* tends to have a shorter mean lag and lower loss rate. Nevertheless, it suffers significant mean accuracy drops.

CASVA overcomes the deficits of both baselines and reaches the highest mean accuracy on all videos for both tasks. Specifically, it reduces the *Offline*'s mean lag by up to 85.96% and loss rate by up to 76.74%. Meanwhile, it outperforms *ABR-U* on all evaluation metrics in almost all cases. We also note that CASVA shows more stable mean lag and loss rate values on different videos compared with the baselines. For the mean accuracy, dashcam videos have lower values than stationary camera videos. This is because the dashcam videos have larger bitrates than the stationary camera videos for the same configuration. As a result, the dashcam videos must sacrifice higher accuracy to achieve similar lags and loss rates under the same network conditions.

Fig. 8a and Fig. 8b show the mean accuracy and mean lag that each method achieves on various videos in the *delivery-first* streaming mode. Since all video segments have to be uploaded in this mode, the mean lags of all methods rise compared with their *latency-first* counterparts (shown in Fig. 7b). The *Offline* method reaches the highest mean accuracy on all videos at the expense of significant lags. For instance, the mean segment lags of video DASH1, STA1, and STA3 are even longer than a segment duration (2 seconds), falling in an unacceptable range for live video analytics. *ABR-U* also experiences considerable lag increases. It struggles to make a good trade-off between bitrate and lag but ends up with the lowest analysis accuracy. This shows that the coarse-grained bitrate control is inadequate for live video analytics scenarios. Different from these two baselines, CASVA achieves sub-second mean lags on all videos with relatively high accuracies. This indicates that it can adapt to all underlying dynamics to balance analysis accuracy and lags.

As a supplement, Fig. 8c reveals how distinct methods deal with lags when streaming the video DASH1 for object detection under a 4G/LTE network trace. As shown, from segment 42, the segment lag of the *Offline* method accumulates to about 8 seconds. This method is unaware of the increasing lags. It only selects configurations based on indirectly sensed network bandwidth variations; it does not make up for the previously made bad decisions. *ABR-U* can monitor and mitigate the lag, but the coarse-grained bitrate control tends to cause significant variances. By contrast, CASVA addresses this issue gracefully. It cautiously explores the room for accuracy improvement under the bandwidth constraint with the fine-grained configuration control. Once the lag accumulates, it quickly alleviates it by configuration adaptation.

Fig. 9 further illustrates the *delivery-first* mode results when steaming videos under the FCC network traces for the semantic segmentation task. The *Offline* method maintains its leading position in mean accuracy. However, all mean lags of this method exceed a segment duration. It should be noted that the Y-axis of Fig. 9b is in log-scale, and CASVA achieves sub-second mean lags for all videos. By comparison, *ABR-U* shows abnormal large mean lags. We examine its running logs and find that this is due to the poor network conditions involved in the FCC network traces. A considerable part of the network traces cannot even support the real-time streaming of 240p videos. For example, Fig. 9c demonstrates the lag variations when streaming the DASH1 video under an FCC



(a) Mean accuracy comparison (b) Mean lag comparison (c) Lag variations over time

Fig. 9: The performance of different methods on various videos in *delivery-first* mode (task: SS; network traces: FCC).

network trace. The average network throughput of this trace is lower than the average bitrate of the video. Under this network trace, *ABR-U* always selects the lowest bitrate but still cannot prevent the lag from accumulating. Thanks to the flexibility of fine-grained configuration choices, CASVA easily gets out the tight corners and streams the videos in real time.

VI. Related Work

Configuration Optimization for Live Video Analytics. To meet the tremendous resource demands of high-accuracy live video analytics, streaming camera videos over the network to a resource-rich data center or cloud for processing becomes popular. Optimizing configurations to balance between computational resources and inference accuracy is a prevalent technique used by existing systems [1], [3]. For example, VideoStorm [1] profiles the computational resource demand and accuracy of different configuration knob combinations for each live video query offline. It then adjusts knobs for large-scale concurrent queries according to their quality and lag goals online. To adapt to the ever-changing resource demands caused by video content dynamics, Chameleon [3] dynamically tunes the configuration knobs by frequent profiling and amortizes the profiling costs over time and across multiple videos based on the underlying temporal and spatial correlation. These systems typically assume that videos are streamed to the data center over dedicated network links. Hence, they focus on computational resources optimization and ignore the influences of network variations. CASVA instead addresses the challenges of streaming videos over scarce and dynamic network conditions, e.g., streaming videos collected by moving cars to powerful backends for live video analytics.

AWStream [5] is a system for wide-area streaming analytics, which adjusts the application data rate to match available bandwidth while maximizing the achievable accuracy. Unfortunately, like the *Offline* baseline, it harnesses a simple method to estimate the network throughput and profiling to acquire the relationship between bandwidth and accuracy under different configurations. It is thus difficult to capture the fine-grained dynamics in complex network conditions and video content. DDS [24] provides a server-driven video streaming method for DNN inference, where low-quality video segments are continuously sent to the server for inference, and unsatisfactory inference outputs can drive the camera to resend high-quality content to compensate for accuracy. In contrast, CASVA is specifically designed for live streaming scenarios, so a client-driven design framework is chosen to avoid the delay of waiting for feedback from the server.

RL in Video Streaming. Recent years have witnessed the successful applications of RL in building video streaming systems. Several on-demand video streaming systems attempt to maximize human-perceived QoE while adapting to the underlying network conditions [15], [32], [33]. For instance, Pensieve trains a DRL model to learn ABR algorithms to adapt to various environments and QoE metrics based on experiences collected by the client video players. The following work, Comyco [33], further improves the sample efficiency by training the policy under the guidance of expert trajectories. Additionally, RL-based methods also gain popularity in other video streaming scenarios. For instance, in order to improve QoE of live viewers, Rldish [34] relies on RL to dynamically select initial video segment of HTTP live streaming at the edge content delivery network servers. iView [35] employs multimodal learning and DRL for user QoE improvements in multi-viewpoint 360° interactive video streaming. Unlike these systems, to the best of our knowledge, CASVA is the first video analytics streaming framework that exploits DRL to maximize DNN-perceived QoE while adapting to video content and network dynamics.

VII. Conclusion

Streaming camera videos for video analytics applications rather than human viewers has become an important topic. This paper starts with extensive measurements on real-world camera videos to investigate the impacts of video configurations on bandwidth requirement and analytics accuracy. We identified the potentials of configuration adaptation in live video analytics streaming and revealed the fundamental challenges brought by the intrinsic dynamics in network conditions and video content. We accordingly proposed CASVA, a configuration-adaptive streaming framework for live video analytics. Instead of relying on fixed heuristics or inaccurate performance models, CASVA employs DRL to learn an efficient policy in complicated streaming environments. Our trace-driven evaluations with various network traces and videos demonstrated the superiority of CASVA.

REFERENCES

[1] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proceeding of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[2] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proceedings of the 37th IEEE Conference on Computer Communications (INFOCOM)*, 2018.

[3] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018, pp. 253–266.

[4] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the 2020 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2020, pp. 359–376.

[5] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018, pp. 236–252.

[6] C. Pakha, A. Chowdhery, and J. Jiang, "Reinventing video streaming for distributed vision analytics," in *Proceedings of the 10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2018.

[7] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: A gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019, pp. 322–337.

[8] A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger, "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *2018 USENIX Annual Technical Conference (ATC)*, 2018, pp. 29–42.

[9] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," in *Proceedings of the 2nd SysML Conference (SysML)*, 2019.

[10] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the edge-cloud barrier for real-time advanced vision analytics," in *Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2019.

[11] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," in *Proceedings of the 40th IEEE Conference on Computer Communications (INFOCOM)*, 2021, pp. 10–13.

[12] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016, pp. 123–136.

[13] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary," in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo)*, 2019, pp. 27–32.

[14] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, "Distilled split deep neural networks for edge-assisted real-time systems," in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo)*, 2019, pp. 21–26.

[15] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2017, pp. 197–210.

[16] Y. Live, "Jackson hole wyoming usa town square live cam - seejh.com," https://www.youtube.com/watch?v=1EiC9bvVGnk, 2021, [Online; accessed 28-June-2021].

[17] ——, "La grange, kentucky usa - virtual railfan live," https://www.youtube.com/watch?v=WsYtosQta5Y, 2021, [Online; accessed 28-June-2021].

[18] YouTube, "Driving downtown - chicago 4k - usa," https://www.youtube.com/watch?v=kOMWAnxKq58, 2021, [Online; accessed 28-June-2021].

[19] ——, "London 4k - night drive - uk," https://www.youtube.com/watch?v=Ujyu8foke60, 2021, [Online; accessed 28-June-2021].

[20] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[21] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[22] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[23] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "Icnet for real-time semantic segmentation on high-resolution images," in *Proceedings of the 15th European Conference on Computer Vision (ECCV)*, 2018, pp. 405–420.

[24] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the 2020 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2020, pp. 557–570.

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33nd International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction, second edition*. MIT press, 2018.

[28] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang *et al.*, "Mastering complex control in moba games with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 34, no. 04, 2020, pp. 6672–6679.

[29] FCC, "Measuring broadband raw data releases - fixed," https://www.fcc.gov/oet/mba/raw-data-releases, 2021, [Online; accessed 28-June-2021].

[30] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[31] "Pytorch: An open source machine learning framework that accelerates the path from research prototyping to production deployment," https://pytorch.org/, [Online; accessed 28-June-2021].

[32] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware internet video delivery," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 645–661.

[33] T. Huang, C. Zhou, R. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," in *Proceedings of the 27th ACM International Conference on Multimedia (MM)*, 2019, pp. 429–437.

[34] H. Wang, K. Wu, J. Wang, and G. Tang, "Rldish: Edge-assisted qoe optimization of http live streaming with reinforcement learning," in *Proceedings of the 39th IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 706–715.

[35] H. Pang, C. Zhang, F. Wang, J. Liu, and L. Sun, "Towards low latency multi-viewpoint 360° interactive video: A multimodal deep reinforcement learning approach," in *Proceedings of the 38th IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 991–999.