Batch Adaptative Streaming for Video Analytics

Lei Zhang[†], Yuqing Zhang[†], Ximing Wu[†], Fangxin Wang[‡‖], Laizhong Cui[†*], Zhi Wang[§‖], Jiangchuan Liu[††]
[†]Shenzhen University, Shenzhen, China
[‡]The Chinese University of Hong Kong, Shenzhen, China
[§]Tsinghua Shenzhen International Graduate School, Shenzhen, China
[‖]Peng Cheng Laboratory, Shenzhen, China
[††]Simon Fraser University, Burnaby, Canada
leizhang@szu.edu.cn, zhangyuqing20@email.szu.edu.cn, wuximing2019@email.szu.edu.cn,
wangfangxin@cuhk.edu.cn, cuilz@szu.edu.cn, wangzhi@sz.tsinghua.edu.cn, jcliu@cs.sfu.ca

***Abstract*—Video streaming plays a critical role in the video analytics pipeline and thus its adaptation scheme has been a focus of optimization. As machine learning algorithms have become main consumers of video contents, the streaming adaptation decision should be made to optimize their inference performance. Existing video streaming adaptation schemes for video analytics are usually designed to adapt to bandwidth and content variations separately, which fail to consider the coordination between transmission and computation. Given the nature of batch transmission in video streaming and batch processing in deep learning-based inference, we observe that the choices of the batch sizes directly affects the bandwidth efficiency, the response delay and the accuracy of the deep learning inference in video analytics. In this work, we investigate the effect of the batch size in transmission and processing, formulate the optimal batch size adaptation problem, and further develop the deep reinforcement learning-based solution. Practical issues are further addressed for Implementation. Extensive simulations are conducted for performance evaluation, whose results demonstrate the superiority of our proposed batch adaptive streaming approach over the baseline streaming approaches.**

***Index Terms*—video analytics, adaptive streaming, batch, machine learning**

I. Introduction

With the advances of deep neural network (DNN)-based learning techniques in recent years, complicated computer vision tasks, e.g., highly accurate object detection and recognition, can now be accomplished. Various video analytics applications including surveillance [1], traffic control [2], factory monitoring [3], and face authentication [4] have driven the pervasive deployment of video cameras, whose number and coverage are still rapidly expanding. A typical pipeline for video analytics is that, video contents are generated at the source camera, streamed to the cloud/edge server, and processed there by the machine learning (ML) algorithm for the computer vision (CV) tasks. Such video analytics applications usually run in long durations and thus demand significant amounts of network resources.

The Internet video streaming has been a key optimization target, which must balance between maximizing application-level quality and adapting to limited network resources. Conventionally, the video source naturally knows which version of frames can provide better QoS for human users–a smooth playback with higher resolutions. However, for video analytics applications, the quality-of-service (QoS) should be the inference accuracy of the ML algorithm, which depends on the video contents. This makes a huge difference from traditional video services for human users. When ML algorithms become the video consumers, only the server can tell the QoS, while the video source does not know which transmission configuration is actually better. Adaptation relying on the source side estimation could lead sub-optimal performance. Therefore, the adaption decision should be made at the server side.

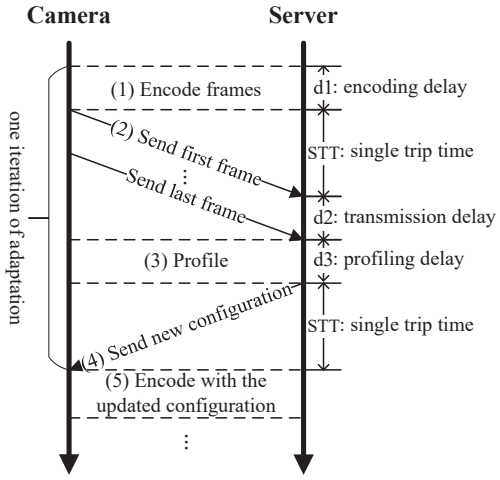


Fig. 1. The workflow of a typical server-driven streaming adaptation

The server-driven adaptation approach is proposed and proved to work better than the camera-side adaptation heuristics for machine-centric video streaming [5]. Fig. 1 illustrates the basic workflow of server-driven adaptation. Given a configuration containing several "knobs" (e.g, resolution, frame rate, region of interest), the source camera (1) encodes the raw frames accordingly and (2) sends the encoded video chunk to the server. The server (3) profiles the received chunk to select a configuration update with the best bandwidth-accuracy trade-off and (4) sends the adaptation decision back to the source camera, which (5) applies the updated configuration at the earliest feasible time and carries on the above process

*Corresponding author

iteratively. As the adaptation is based on server side profiling, it is possible to adapt to both network and video contents, although unfortunately the two goals are achieved separately in existing solutions.

An intrinsic feature of such machine-centric video streaming, which is omitted in literature, is batch transmission and batch processing–the video contents are transmitted and processed in batches, e.g., the group of pictures (GOPs) for encoding [6], the video chunks for streaming [7], the input frames for CV algorithms [8]. We observe that the selection of the batch size affects not only the response time but also the inference accuracy, the two key performance metrics for video analytics. From the network perspective, an appropriate batch size would allow the source camera to encode video frames with a high compression ratio. The frames are encoded in batches so that the correlation between them can be leveraged to reduce the redundant pixels as much as possible. From the video content perspective, an appropriate batch size would allow the server to make quality adaptation decisions without excessively increasing the response delay. Increasing batch size will directly lead to longer delays as shown in Fig. 1 as more data is transferred and processed each time, while a too small batch size cannot yield good configurations as the profiling samples are not enough. Therefore, choosing an appropriate batch size is a crucial task for video analytics.

Unlike existing adaptation approaches with fixed batch size, in this work, we for the first time propose the batch adaptation for video streaming in video analytics applications. We analyze the impacts of batching in transmission and processing on various aspects and show that adjusting the batch size can help adapt to both network and video contents, especially the latter as video contents are constantly changing and need to be closely profiled. We further formulate the batch adaptation optimization problem and develop a deep reinforcement learning (DRL) based solution to maximize the overall inference accuracy. To make our solution feasible for deployment, we address key issues towards a practical implementation.

The rest of this paper is organized as follows. Section II presents the background about batching and our motivation for this work. We formulate the optimization problem for batch adaptive streaming for video analytics in Section III, and propose a DRL-based batch-size adaptation solution in Section IV. The key implementation issues are further discussed in Section V. We evaluate the performance of our solution and compare it with other baselines in Section VI, and finally conclude the paper in Section VIII.

II. Background and Motivation

A. Batch Transmission and Batch Processing

Networking systems transfer and process data in batches to improve efficiency. Video analytics systems naturally take this batch transmission and batch processing approach. At the video source side, the media codec compresses frames in batches (i.e., GOPs) to reduce data redundancies, which can be transferred using less bandwidth. As the frames are encoded and transferred in batches, the reconstruction requires the information from multiple frames, e.g., decoding P-frames and B-frames needs I-frames.

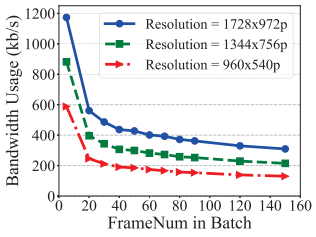


Fig. 2. Batching effect on transmission efficiency

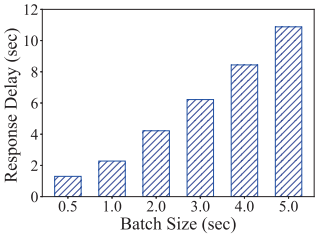


Fig. 3. Batching effect on response delay

At the server side, the reason for batch processing is twofold. First, given today's highly advanced GPUs, analyzing one frame at a time cannot fully utilize the hardware resources, e.g., GPU's computing capability and GPU memory. Therefore, ML algorithms for video analytics can take multiple frames as a batch input to improve throughput for inference [9]. Second, as the server is responsible for making adaptation decisions, it needs to profile a reasonably-sized batch of frames to select a quality configuration. Too few samples could lead to poor selections since the limited video contents are not representative.

Taking object detection (using YOLOv3 [10] as the detector model) as an example, we discuss the batching impact on various factors for video analytics applications and present our motivations in the following subsections.

B. Impact of Batch Size on Bandwidth

The size of a frame is mostly decided by the configurable knobs such as resolution in pixels and which part of the frame to encode. Transferring multiple frames in a batch after encoding can reduce the bandwidth requirement. This data compression technique is highly optimized by today's media codecs and is still one of their working targets. We set the same frame rate and vary the number of frames that are encoded in a batch as a GOP, and then check the bandwidth requirement. Fig. 2 shows that when there are more frames to be encoded in a batch as a GOP, the bandwidth consumption rapidly deceases at first (from 1 frame/batch to 50 frames/batch) and then slowly goes down afterwards. This result demonstrates the batching effect on bandwidth requirement, which implies that adjusting batch size (with fixed frame rate setting) can indeed help to adapt to network to a certain degree.

C. Impact of Batch Size on Response Delay

Another direct impact of increasing the batch size is that each batch takes longer to be encoded, transferred, and profiled. Therefore, a larger batch size leads to a longer response delay–time for one iteration of adaptation in Fig. 1. We change the batch size from 0.5s to 5s and collect the response delays following the server-driven adaption process. It is not surprising to see the response delay increases with the batch size as shown in Fig. 3. It is worth noting that here we use fixed

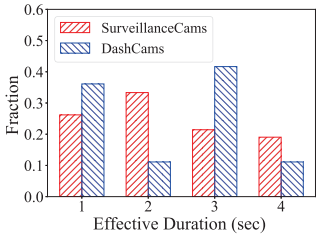


Fig. 4. Distribution of Effective Duration

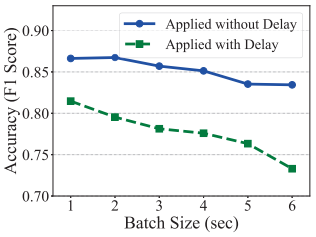


Fig. 5. Accuracy with different batch size

bandwidth, while the bandwidth varies constantly in practice, which may change the numbers but the growing trend holds.

D. Impact of Batch Size on Inference Accuracy

In the server-driven adaptation, the adaptation decision is made by profiling a batch and finding its best configuration. As the most important performance metric, inference accuracy is also influenced by batch size from two aspects. With different batch sizes, the video is profiled at different paces, which will likely generate distinct configuration suggestions with different accuracy performance. Moreover, as batch size directly affects response delay, which in turn changes the time for the suggested configuration to take effect. The configuration can be outdated if it takes too long to apply.

1) Varying Configuration for Uncertain Video Contents: The best configuration and its effective duration change with the video contents. The uncertainty and the dynamics of video contents require the batch's profiling to be appropriate and responsive. For different test videos, we analyze the best configuration and count the occurrences for how long it lasts with the granularity of 1 second. Fig. 4 shows different distributions for the effective duration. The result suggests that the best configuration does not last very long and different videos have different dynamics of the contents. As the features of the contents change in different videos and different parts of a video, a fixed batch size is sub-optimal for profiling. Varying batch size properly can help closely monitoring and profiling the video at the right pace to obtain quality configurations.

2) Delay for Configuration to Take Effect: Batch size affects not only the configuration obtained from profiling but also the time when it becomes effective. Fig. 5 plots the F1 scores of the inference when the suggested configuration from profiling is applied with and without the response delay. Even if the adaptation decision is applied immediately, the inference performance varies for different batch sizes (blue solid line), which is accorded with the previous observation. When the suggested configuration takes effect with longer delays, the inference accuracy drops significantly (green dashed line) because the profiling result becomes less timely and even outdated for the dynamic video contents.

To sum up, we have presented the implications of batching. On one hand, adjusting batch size in video analytics applications can trade-off between various performance metrics to better fit network and video contents. On the other hand, it is not trivial to evaluate the comprehensive impacts of changing the batch size, and hence it remains a difficult task to strike the optimal balance between maximizing application-level quality and adapting to limited network resources.

III. Problem Formulation

A. System Model

For a video analytics application, the video contents are organized and transmitted in N batches. Note that N can be infinite for continuous video analytics. For each batch i, the transmission batch size is $batch\_t_i$, which means this video segment is of $batch\_t_i$ second length. The server profile the first $batch\_p_i$ second of batch i to make the adaptation decision. The profiling can be done based on a part of or the whole batch, which implies $batch\_p_i \leq batch\_t_i$. Let the transmission configuration applied to batch i be $c_i = \{c_i^r, c_i^f, c_i^b\}$, where c_i^r denotes the setting for resolution in pixels, c_i^f denotes the setting for frame rate, and c_i^b denotes the setting for region of interest (ROI) blocks in the frame. Therefore, batch i has $c_i^f \cdot batch\_t_i$ frames to encode at the source and $c_i^f \cdot batch\_p_i$ frames to profile at the server. Assume that function $g(c_i, batch\_t_i)$ and $f(c_i, batch\_t_i)$ calculate the encoding time and the amount of transmitted data for a batch, respectively, given its configuration c_i and batch size $batch\_t_i$. Note that $g(\cdot)$ and $f(\cdot)$ can be obtained by measuring and fitting the multimedia processing process, as both the encoding time and the data volume after encoding directly depend on the factors including frame resolution c_i^r, ROI areas c_i^b, and number of frames $c_i^f \cdot batch\_t_i$. As the computing capability and the available bandwidth may vary with time and the applied configuration, we further assume that the server has a computing capability of C_i frames per second for batch i, and the available bandwidth for transmitting batch i is B_i.

B. Delay Analysis

Since the server driven adaptation relies on a feedback control logic, it is important to analyze the delay components. As shown in Fig. 1, the response delay is defined as the duration from the source camera preparing a batch with a configuration to it receiving the updated configuration applied to a new batch, which mainly consists of four parts: the time for encoding, the time for transmission, the time for profiling, and the round trip time (RTT).

Given the system model described above, we next calculated the delay components that are affected by the selections of $batch\_t_i$ and $batch\_p_i$. The encoding delay d_i^1 for batch i can be decided as

$$d_i^1 = g(c_i, batch\_t_i). \tag{1}$$

The transmission delay d_i^2 for batch i can be calculated as

$$d_i^2 = f(c_i, batch\_t_i)/B_i. \tag{2}$$

Note that the bandwidth requirement for transmitting batch i depends on both the configuration and the batch size. To profile the received frames, the server needs to run the CV algorithm to evaluate the inference performance of different

configurations, which involves a certain amount of computation. The processing delay d_i^3 for profiling batch i is calculated as

$$d_i^3 = c_i^f \cdot batch\_p_i / C_i. \tag{3}$$

Therefore, we have the total response delay for batch i as

$$d_i = d_i^1 + d_i^2 + d_i^3 + RTT. \tag{4}$$

C. Optimization Problem

Denote $\mathbb{C} = \text{Profile}(c_i^f, batch\_p_i)$ as the candidate set of configurations based on the profiling result from the input $c_i^f \cdot batch\_p_i$ frames of batch i. Note that c_i^f and $batch\_p_i$ can be uniquely defined the profiling window, and $\mathbb{C}$ is a Pareto-optimal set for the bandwidth-accuracy trade-off, in which any configuration cannot find an alternative that requires less bandwidth and offers a higher inference accuracy. Let c' be the best configuration in $\mathbb{C}$ with a corresponding $batch\_t_i$ given the bandwidth constraint. As c' is updated to the source camera to be applied on the next batch, we have $c_{i+1} = c'$.

We further assume $\text{Accuracy}(c_i)$ to be the function mapping the inference accuracy with the applied configuration c_i. It is worth noting that the inference accuracy of batch $i+1$ depends on the configuration that is applied to batch $i+1$ but was decided a while before by profiling batch i. For the first batch, an initial configuration c_1 is provided directly as a default setting on the source camera.

Our objective is to select the best $batch\_p_i$ and $batch\_t_i$ so that the overall accuracy is maximized:

$$\sum_{i \in [1,N]} \text{Accuracy}(c_i) * batch\_t_i, \tag{5}$$

subject to

$$batch\_p_i \leq batch\_t_i, \tag{6}$$

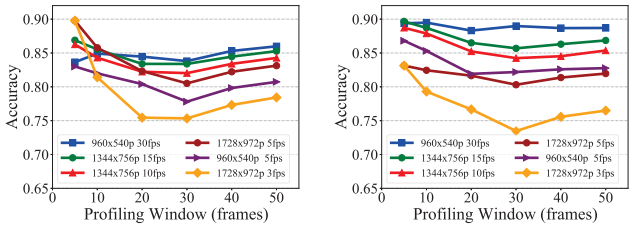
$$d_i \leq batch\_t_i, \tag{7}$$

$$\mathbb{C}_i = \text{Profile}(c_{i-1}^f, batch\_p_{i-1}) \tag{8}$$

$$c_i = \text{best}(\mathbb{C}_i) \tag{9}$$

$$\sum_{i \in [1,N]} batch\_t_i = L_{video} \tag{10}$$

Eq. 6 implies that the profiling is conducted within the current batch. Eq. 7 guarantees that the video analytics pipeline works fluently and does not have any bottleneck. The components and the detailed computation of d_i can be referred to Eq. 1, Eq. 2, Eq. 3, and Eq. 4. This constraint does not allow any accumulated latency to build up so that our model can work in scenarios such as live video analytics, where the analytics requests arrive constantly and need to be done in time without choking the pipeline for later processing. Eq. 7 also implies the combination of the configuration and the batch size should satisfy the bandwidth requirement. Eq. 8 and Eq. 9 indicate that we select the Pareto-optimal set $\mathbb{C}_i$ from the last step and then select the best configuration to apply in the current step. Eq. 10 guarantees that the total length of all the batches should be equal to the video length L_{video}, no matter how we select the batch size at each step. Note that L_{video} can be an arbitrary boundary for live video analytics within which we consider maximizing the inference accuracy.



(a) Video 1 (SurveillanceCams) (b) Video 2 (DashCams)

Fig. 6. Screenshots of two browsing sessions with the same timestamp

This problem is difficult to solve because 1) multiple decisions need to be made in large search spaces, 2) the decision made based on the current batch affects the inference performance of the next batch, and 3) $\text{Accuracy}(c_i)$, the key function for the optimization objective, is unknown and can hardly be obtained analytically.

IV. DRL-based Batch Adaptation

To deal with the hardness of the formulated problem, we propose a two-step solution, which selects $batch\_p_i$ and $batch\_t_i$ for each batch individually. In the first step, we conduct an incremental profiling that gradually increases $batch\_p_i$ until a quality set of configurations are generated. After this profiling step, we leverage the nice characteristic that, given the current conditions (state), the controller (agent) needs to make a choice of $batch\_t_i$ (action) whose benefit (reward) will be recognized in the process of streaming and processing (environment). As it naturally falls into the scope of DRL, we take advantage of this advanced technique to smartly select $batch\_t_i$ in the second step of our solution.

A. Profiling with Early Quit

The goal of profiling is to generate $\mathbb{C}$, the Pareto-optimal set of configurations for the batch. Intuitively, we can always set $batch\_p_i$=$batch\_t_i$ and profile the whole batch to get the final $\mathbb{C}$, which however may introduce unnecessary overhead. Hence, it is better to keep $batch\_p_i$ as small as possible while we can still obtain the same (or a close-enough) $\mathbb{C}$. A good setting of $batch\_p_i$ will not make suggestions based on too few frames, but can allow the profiling process to "sense" the general content features of the whole batch. We vary $batch\_p_i$ to increase the profiling window and for each profiling window we track the highest accuracy can be achieved by assuming that the profiling result is applied to the whole current batch. We observe that the overall accuracy on the batch increases rapidly at first and then keeps relatively stable (see the detailed result in Fig. 8 and more illustrations later). Therefore, we infer that the optimal configuration can likely be found at an early stage before the profiling window gets very large.

Given the above observation, we design the first step of our solution to work as follows. In general, when a batch

of encoded frames is received, we profile from the start and gradually increase $batch\_p_i$ with a small step size. This profiling process should stop at the earliest possible time. The key question is when to quit early. For each $batch\_p_i$, we find the corresponding $\mathbb{C}$ and keep a record of the historical $\mathbb{C}$s. Given the current bandwidth estimation, once the top-k configurations and their rankings become stable, the profiling window stops increasing. Fig. 6 plots the top-k configurations (for c_i^r and c_i^f) and their accuracy (calculated within the current profiling window rather than the whole batch to simulate actual online processing) curves as $batch\_p_i$ increases for the two test videos in different types. The results show that the sequence of the top-k configurations remain the same after the profiling window reaches a certain point. For example, after 30 frames for video 1 and 10 frames for video 2, the top-3 configurations are observed in order of blue, green, and red. When this ranking is found to repeat later, the profiling terminates early. With the selected $batch\_p_i$, we are now able to obtain the corresponding Pareto-optimal set of the configurations. The enhancement details for efficient runtime execution will be discussed in Section V-C.

B. DRL Model for Batch Size Selection

Upon obtaining $batch\_p_i$ and $\mathbb{C}$, we next discuss how to select the best batch size $batch\_t_i$ so that the current accuracy can be maximized. The batch size selection problem can be viewed as a sequential process where the previous and current selections will have an impact on the subsequent states, which calls for careful decisions at every step. Deep reinforcement learning (DRL) has been widely explored in recent years and demonstrates powerful capabilities in mining such hidden characteristics and making proper decisions. We advocate that DRL is most appropriate in our context and thus propose a DRL-based algorithm for batch allocation. The detailed DRL model design is described as follows.

State Space. In our context, the final decided configuration c_i will be applied to time step i, which together with the $batch\_t_i$ decided at the last step will affect the current accuracy. Our target of DRL model is to decide the most proper batch size at every time step. Intrinsically, the selection of transmission batch size depends on the video content features and the available bandwidth conditions, which are usually highly dynamic and sequentially dependent. From the video content perspective, a representative example is that compared to stable scenes, dynamic scenes usually need more frequent profiling, i.e., smaller transmission batch size, to adapt the fast-changing video features. From the bandwidth perspective, the available bandwidth for the next period also affects the configuration selection and transmission batch size selection, e.g., lower available bandwidth usually requires higher compression rate or larger batch size. Therefore, we consider including these metrics in the state space for future batch selection. The state space can be represented as $s_i = \{\overrightarrow{c_i}, \overrightarrow{B_i}, \overrightarrow{batch\_t_i}\}$. Here $\overrightarrow{c_i}$ indicates the past k configurations; $\overrightarrow{B_i}$ indicates the past k available bandwidth; $\overrightarrow{batch\_t_i}$ indicates the past k transmission batch selections. Here we consider the past consecutive k samples in a state since all the metrics are changing gradually and the changing patterns therein will help the DRL agent better capture the hidden characteristics. The default value of k can be from 4 to 8.

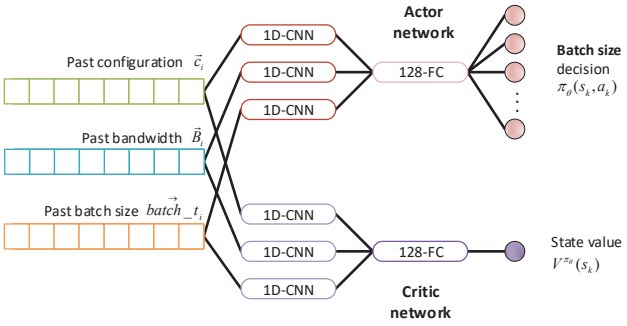


Fig. 7. Design of the actor-critic network

Action Space and Policy Network. With the state s_i at every step i, the DRL agent will take an action a_i that corresponds to the next selected transmission batch size $batch\_t_{i+1}$ to maximize the further accuracy. Originally, the batch size selection is a continuous variable within a reasonable range. In practical model setting, however, two very close batch size will not lead to obvious configuration difference. Thus, we can actually simplify this problem by discretizing this variable with certain granularity, e.g., using every 0.1 second as a gap. In reinforcement learning, our target is to obtain a policy to conduct the action selection. In our DRL model, we design a deep neural network as the policy network, which at every time step i will make the proper $batch\_t_{i+1}$ (i.e., a_i) selection once be well trained.

Reward Setting. In the DRL model, each time after the agent takes an action, a corresponding reward from the environment will be received. The agent will be trained towards the objective of maximizing the accumulated rewards. Thus, we design the model reward to align with our optimization objective, i.e., maximizing the inference accuracy throughout the entire video analysis process. The reward at step i is then represented as $r_i = batch\_t_i * Accuracy(c_i)$.

Learning Architecture. We use an actor-critic architecture [11] for DRL model training. Actor-critic architecture consists of two components, an actor network that outputs the probability distribution of the possible $batch\_t_i$ for action selection, and a critic network that learns an estimate of the expected total reward and guides the update direction of the actor network. The network design of our DRL model is illustrated in Fig. 7.

Besides, we find that using traditional policy gradient methods in our model context is not enough, which can easily experience quite large policy update, leading to unstable convergence and poor data efficiency. To address this problem, we integrate a clipped surrogate objective inspired by Proximal Policy Optimization Algorithms (PPO) [12] in our policy update process to stabilize the policy and further achieve more robust performance. The updating steps for the actor and critic

networks are as follows:

a) Critic network: Denote the parameters in the critic network and actor network as θ_v, θ_a respectively. Then the estimation of state value $V^{\pi_\theta}(s_k)$ under policy π_θ is the output of critic network. We update the critic network by mean squared error as follows:

$$\theta_v \leftarrow \theta_v - \beta \sum_k \nabla_{\theta_v} (R_k^{(n)} - V^{\pi_\theta}(s_k))^2 \tag{11}$$

The n-step estimated return is defined as:

$$R_k^{(n)} = \sum_{t=0}^{n-1} \gamma^t r_{k+t} + \gamma^n V^{\pi_\theta}(s_{k+n}) \tag{12}$$

where γ is the discount factor.

b) Actor network: The actor network is updated by the clipped surrogate objective L^{CLIP} as follows:

$$\theta_a \leftarrow \theta_a + \alpha L^{CLIP} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{13}$$

where α is the learning rate. The surrogate objective will be modified by clipping the probability ratio as follows:

$$L^{CLIP} = \min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{14}$$

where the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and $\hat{A}_t$ is the advantage value which is equal to subtract $R_k^{(n)}$ and $V^{\pi_\theta}(s_k)$.

V. Enhancements for Implementation

In this section, we discuss the enhancements for our DRL-based batch adaptive solution to address important issues towards a practical implementation.

A. Offline Profiling and Pre-training

Before applying our DRL-based solution, we need to extract the relationship between encoding time and configuration (function $g(\cdot)$) and the relationship between frame size and configuration (function $f(\cdot)$), respectively. Fortunately, there are researches conducted in the area of rate-distortion-complexity analysis for video encoding, of which we can take advantage. For instance, the bitrate of encoded frames can be estimated through modeling [13] or learning [14]. With today's highly-optimized encoders, video encoding is usually fast enough and thus the encoding time is not a concern, which can also be carefully managed [15]. As the encoding features are usually shared across video contents and encoders, we can reduce the difficulty of acquiring $g(\cdot)$ and $f(\cdot)$ by offline profiling the similar types of existing videos.

DRL algorithms are known to be data inefficient [16]. Integrating prior knowledge into DRL algorithms is an important way to improve learning efficiency since it helps to build useful representations [17]. To pre-train our DRL model, we build a simulator that faithfully models the dynamics of video streaming in video analytics applications given the video frames for inference and the input network traces. The simulator contains an object detector running the DNN model on the batch of frames that transcoded from the original video frames using the selected configuration. A network controller responsible for applying the transmission delays based on the given bandwidth is also maintained in the simulator. The simulation environment can provide our DRL model a considerable amount of video analytics experience.

B. Accuracy Indicator at Runtime

During the pre-train in the simulator, the inference accuracy can be directly obtained by comparing the object detection boxes in the transferred frames and those in the raw frames. This ground truth about inference performance can help accelerate the initial learning and lead to a reasonable policy quickly. However, when the video analytics application proceeds online, the server usually does not have raw frames and thus needs other metrics to evaluate the inference performance.

We aim to adjust the batch size at runtime in the streaming for video analytics, and thus cannot rely on the previous offline evaluations. We notice that for most object detectors such as YOLO, the confidence value can be output along with the bounding box. To this end, we use the cumulative confidence (CC) [18] as an instantaneous accuracy indicator, which is the sum of the confidence values for all the recognized objects. The feedback for inference performance can be instantly acquired at runtime on the server for each processed frame.

C. Accelerating Configuration Search

Profiling could be resource-consuming for periodic exhaustive search for the optimal configurations [19]. In our model, the best configuration can be found once the following three factors are fixed: profiling window, batch size, and available bandwidth, which can be quite dynamic at runtime and hence potentially lead to a huge search space. Our solution requires profiling for two different tasks: detecting the best profiling window and obtaining the Pareto-optimal set of configurations. We reduce the search steps by taking advantage of two facts observed in previous studies [19], [20]: (1) the top-k configurations are stable over a short-period duration for similar videos; (2) the relationship between two configurations follows the rule that higher accuracy demands more resources (knobs with higher resource demands can achieve higher accuracies).

For the first task, detecting the profiling window, we let the batch size and the available bandwidth be fixed as the latest setting or estimation. As discussed before, we keep tracking the top-k configurations and their rankings, and stop increasing the profiling window when both of the results become stable over the last several steps. For the second task, obtaining the Pareto-optimal set, we let the batch size and the profiling window be fixed as the latest settings. We take the current bandwidth estimation as a base point and vary the available bandwidth to gradually increase and decrease within a certain range r, a parameter reflecting the bandwidth dynamics. Given the base bandwidth, we can have a best configuration of this case as the root. As the configurations change "monotonically" in the accuracy-resource trade-off, we can search from the root in the direction that accords to the bandwidth change. By applying the above techniques, we are able to accelerate our profiling process significantly.

VI. Performance Evaluation

In this section, we conduct extensive experiments to evaluate our DRL-based batch-size adaptation approach and compare its performance with that of the other baselines.

A. Evaluation Setup

1) Data Sets: For the test videos, we collect three types of videos from YouTube: surveillance for crossroad, dash cameras, and mall cameras. Each type contains four test videos, whose detailed IDs are listed in Table I. To emulate the real-world network conditions and the bandwidth fluctuation, we replay the bandwidth traces from a public 4G/LTE dataset [21]. Since the bandwidth in some traces is quite high, we linearly scale each trace to emulate different network conditions.

2) Settings: We use YOLOv3, one of the state-of-the-art object detection models, as the object detector running a server with Intel(R) Xeon(R) Gold 6152 CPU @ 2.10GHz and Tesla T4 GPU x 4. As the source videos have the frame rate of 30fps, we set the granularity for searching batch size as 10 frames/30fps=0.33s. We check the top-3 configurations to decide the profiling window. For the DRL training, we use the following settings: the number of past samples $k = 5$, the actor network's learning rate $\alpha = 0.0002$, the critic network's learning rate $\beta = 0.0005$, the discount factor $\gamma = 0.999$, and the clipped parameter $\epsilon = 0.2$.

3) Methods for Comparison: We implement two baseline adaptive streaming approaches for video analytics applications. **Chameleon** [19] adopts a video streaming adaptation by periodically profiling the beginning part of each batch to output the configuration update. **AWStream** [20] uses a streaming approach that fully relies on server side offline profiling to examine the accuracy-resource trade-off and applies the proper configuration under the available bandwidth. Unlike our method, both baselines work with the fixed batch size and cannot adjust batch size during the video streaming.

4) Performance Metrics:

Accuracy. We calculate the accuracy of a single transferred frame by comparing the detected objects with the objects detected by the original (highest quality) frame. The F1 score, which is the harmonic mean of precision and recall, can be obtained by checking whether each object is detected as expected in raw frames.

Normalized Bandwidth Consumption. We track the bandwidth consumed by each streaming approach, and then normalize the result over the bandwidth cost for transferring the encoded raw frames (with GOP=30) in the highest quality.

Response Delay. We calculate the response delay for each streaming approach following the definition in SectionIII-B.

B. Effectiveness of Profiling with Early Quit

We first validate our approach of profiling with early quit. We conduct the profiling process in which the profiling window gradually increases with the batch size fixed at 2s (60 frames). For each profiling window, we check the best configuration and calculate the corresponding response delay. Fig. 8 shows the profiling process which early quit at different steps, each curve plots the inference performance of the configuration output from corresponding profiling window, assuming it is applied to the current batch without delay. We can see that the quality of the configuration suggestion improves first and then does not change as the profiling window increases. We check the change of the response delay in Fig. 9, which suggests that larger profiling windows will contribute to longer response delays. Finally, Fig. 10 shows the actual inference accuracy when each configuration output takes effect after the corresponding delay. The accuracy for each profiling process first increases as the suggested configuration is getting better, and then it keeps stable and almost drops at the end since the response delay is so large that the configuration is outdated when it is applied, it may also go up after a larger response delay since the suggested configuration still performs well (the purple line marked with diamonds). The results imply that we can and more importantly we should early terminate the profiling at the right time to achieve the best performance with the least overhead.

C. Accuracy–Bandwidth Trade-off

We next examine the accuracy-bandwidth trade-off that can be achieved by different approaches. As the baselines cannot adjust batch size, we execute each streaming approach multiple times with fixed batch sizes of 1s, 2s, 3s, and 4s, respectively. We plot the accuracy and the normalized bandwidth consumption for the three types of videos in Fig. 11, Fig. 12, and Fig. 13, respectively. As we can see, our batch adaptive streaming approach can achieve the highest accuracy for all three types of videos, whose performance is better than the non-adaptive approaches under all batch size settings.

On the other hand, our proposed batch adaptive approach costs moderate bandwidth, which is reasonable as our goal was to maximize accuracy under bandwidth constraint. There is a clear trend that AWStream and Chameleon share: larger batch sizes demand less bandwidth. This is consistent with our analysis in Section II. The results suggest that our approach can achieve our optimization objective, while sometimes it may consume more bandwidth.

It is worth pointing out that there is no clear relationship between accuracy and batch size. Looking into the performance of AWStream and Chameleon for all three types of videos, it is not true that more bandwidth leads to higher accuracy. AWStream and Chameleon exhibit the worst inference performance when the batch size is 1, which implies that a batch size too small may not utilize the bandwidth efficiently (as the

TABLE I
YouTube IDs of The Test Videos

	Surveillance	DashCam	Mall
Video 1	HpdO5Kq3o7Y	ULcuZ3Q02SI	vrvcCtOrNA0
Video 2	RQA5RcIZlAM	HZaLvgP-R8E	NGA54YdyiUw
Video 3	WsYtosQta5Y	diGHJLCg6i4	dlNRXjF8Y7Q
Video 4	1EiC9bvVGnk	BQjavqQqi-0	Xd5ssXY_BVA

[1]The video can be accessed from the URL: https://www.youtube.com/watch?v=*YouTube ID*

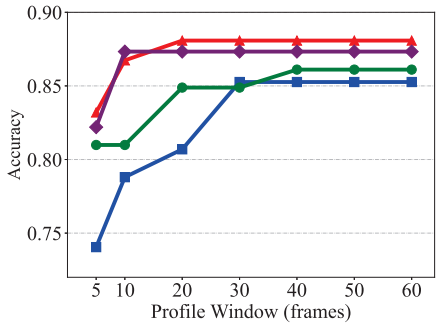


Fig. 8. Configuration's Goodness *vs.* Profile Window

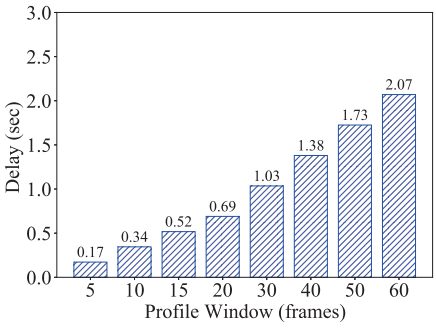


Fig. 9. Delay *vs.* Profile Window

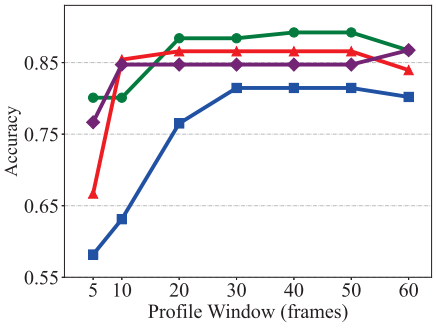


Fig. 10. Actual Accuracy *vs.* Profile Window

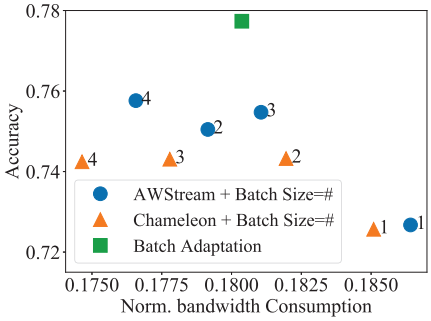


Fig. 11. Accuracy *vs.* Bandwidth for Surveillance Videos

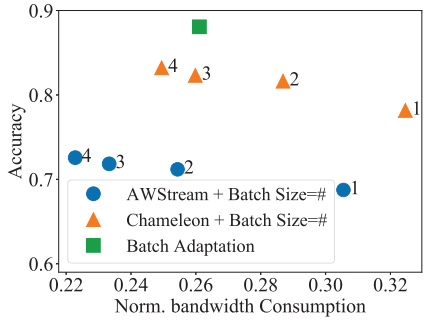


Fig. 12. Accuracy *vs.* Bandwidth for DashCam Videos

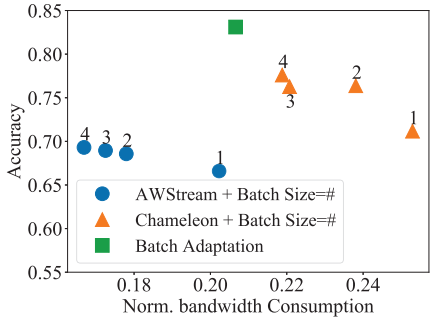


Fig. 13. Accuracy *vs.* Bandwidth for Mall Videos

compression ratio drops). It proves the hardness of our batch size adaptation problem as the effect of adjusting batch size is complicated.

D. Response Delay of Server-Driven Streaming

We further compare the average response delay of our proposed approach with that of the other baselines for all three types of videos. The results presented in Fig. 14, Fig. 15, and Fig. 16 are similar to the results for normalized bandwidth consumption. Our batch adaptive approach does not perform the best nor the worst in this comparison. AWStream has a lower response delay than Chameleon since AWStream uses a streaming approach which relies on server side offline profiling. We can see that server-driven adaptation approaches have long response delays as the streaming adaptation is made based on feedback. Another general observation is that AWStream and Chameleon both show the trend of longer response delays caused by larger batch sizes.

E. Effectiveness of Batch Size Adaptation

Finally, we check if our proposed approach can well adapt to network dynamics and video contents. Fig. 17 is a snapshot of the streaming process for object detection in Video 1 of Type 2, which shows the traces of the adaptation decisions (one segment for a batch and one color for a configuration) and the available bandwidth. We can see that our approach in general can adapt to network fluctuations. In some scenarios, when the bandwidth drops, the accuracy still remains by adjusting the batch size and the configuration, which validates the effectiveness of streaming adaptation.

VII. Related Work

A. Video Analytics Systems

The pervasive camera deployment nowadays generates significant demands for analyzing visual data. With the rapid development of DNN-based learning in recent years, video analytics has been enabled and enhanced to efficiently process large volumes of video data. Glimpse [22] is one of the pioneering video analytics systems, which provides real-time object recognition by capturing and tracking objects of interest from a subset of frames held in an active cache with the assistance of detecting and labeling by server. VideoStorm [23] aims at processing video analytics queries on live video streams at scale. It allocates resources more efficiently by considering the resource-quality profiles of queries, which are handled differently given the distinct quality and lag requirements. Since the new edge computing paradigm emerges, researchers have investigated the improvement of video analytics systems utilizing the capability of edge resources. DeepDecision [24] enables real-time object detection for AR applications by offloading the computation to the edge/cloud. It examines the complex interaction between model accuracy, video quality, battery constraints, network data usage, and network conditions to choose where and which deep learning model to run based on application requirements. To overcome

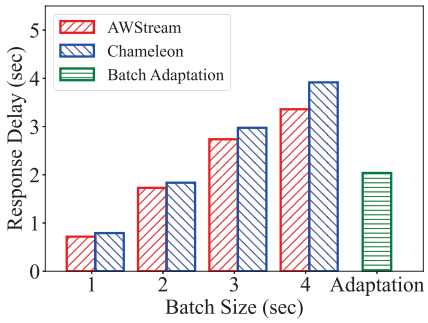


Fig. 14. Response Delay for Surveillance Videos

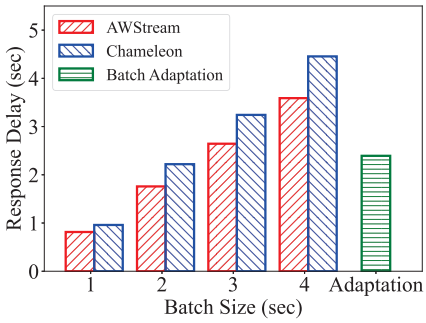


Fig. 15. Response Delay for DashCam Videos

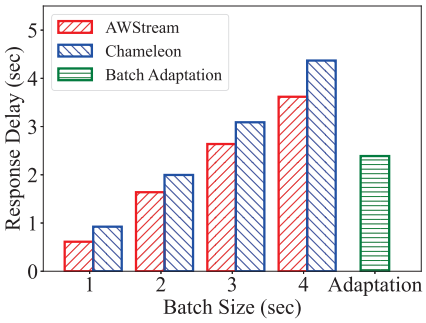


Fig. 16. Response Delay for Mall Videos

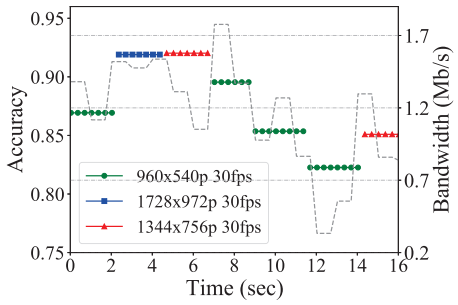


Fig. 17. A snapshot trace of streaming adaptation

the long latency of existing offloading techniques, Liu et al. [25] designed a video analytics system that enables high accuracy object detection at 60fps. Low latency offloading techniques were designed and employed, including decoupling the rendering pipeline from the offloading pipeline and fast object tracking to maintain detection accuracy. CloudSeg [26] is an edge-to-cloud framework for vision analytics designed to achieve high inference accuracy with low streaming cost. It sends the video stream in low resolution, and recovers them into the high-resolution frames via a cloud-assisted super-resolution procedure. It trades computation resources in cloud for bandwidth saving with negligible accuracy degradation.

B. Adaptive Streaming for Video Analytics

Besides processing video data, video streaming is also a crucial part of video analytics, whose optimization has been a focus in prior works. Adaptive streaming in video analytics systems generally attempt to achieve better accuracy-bandwidth trade-off by finding the optimal configuration through certain procedures such as profiling or filtering. Chameleon [19] applies new configuration for the video analytics pipeline based on periodical profiling. As the best configurations likely remain stable over short-period of time and are shared in a group of similar cameras, Chameleon reduces computation and transmission while satisfying the target accuracy by exploiting the spatial and temporal correlations between optimal configurations [27]. AWStream [20] offline profiles the Pareto-optimal set of configurations and online tunes the data rate by using the mapping of bandwidth consumption and application accuracy. Reducto [28] applies the adaptation logic mainly on the source camera. It filters video frames according to features and thresholds selected by server, and offloads filter computation to the camera side to reduce the bandwidth consumption. This source-driven approach is arguably sub-optimal since the camera's computation capability may be insufficient to extract content features as well as the server, which motivates the design of server-driven streaming protocols letting the server decide what/when to stream from the camera [29]. DDS [5] is a video analytics system built with a server-driven adaptive streaming approach. The camera sends two rounds of video data: the first stream is delivered in low resolution, which is analyzed by the server to generate the re-transmission suggestions, while the second stream is sent in high quality to improve inference accuracy, which only encodes the requested subregions of the frames. DDS maintains higher accuracy while reducing bandwidth usage at the cost of increased response delay.

VIII. Conclusion

In this paper, we proposed the batch adaptive streaming for video analytics applications. We first identified the necessity of batching in transmission and computation. We also investigated the impacts of changing batch size on various factors to show that batch size adaptation can help improve video analytics. We formulated the optimization problem for batch adaptive streaming and further solved it using instant profiling and a DRL-based adaptation. The implementation issues were addressed to develop a practical solution, whose performance was evaluated and proved to be superior through extensive trace-driven simulations.

Acknowledgments

This work has been partially supported by National Key RD Program of China (Nos. 2018YFB1800302 and 2018YFB1800805), National Natural Science Foundation of China (Nos. 61902257, 61772345, 61872215, and 62102342), Natural Science Foundation of Guangdong Province (No. 2021A1515012633), Shenzhen Science and Technology Program (Nos. RCYX20200714114645048, JCYJ20190808142207420, GJHZ20190822095416463, and RCYX20200714114523079), and Pearl River Young Scholars funding of Shenzhen University. J. Liu's research is supported by an NSERC Discovery Grant.

REFERENCES

[1] S. Wang, S. Yang, and C. Zhao, "Surveiledge: Real-time video query based on collaborative cloud-edge deep learning," *Proceedings of IEEE INFOCOM 2020*, pp. 2519–2528.

[2] T. Abdullah, A. Anjum, M. F. Tariq, Y. Baltaci, and N. Antonopoulos, "Traffic monitoring using video analytics in clouds," in *Proceedings of 2014 IEEE/ACM International Conference on Utility and Cloud Computing*, pp. 39–48.

[3] S. Coscetti, D. Moroni, G. Pieri, and M. Tampucci, "Factory maintenance application using augmented reality," in *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, ser. APPIS 2020, pp. 1–6.

[4] Y. Chen, J. Sun, X. Jin, T. Li, R. Zhang, and Y. Zhang, "Your face your heart: Secure mobile face authentication with photoplethysmograms," in *Proceedings of IEEE INFOCOM 2017*, pp. 1–9.

[5] 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 ACM SIGCOMM 2020*, pp. 557–570.

[6] B. Zatt, M. Porto, J. Scharcanski, and S. Bampi, "Gop structure adaptive to the video content for efficient h. 264/avc encoding," in *2010 IEEE International Conference on Image Processing*, pp. 3053–3056.

[7] T. Zhang, F. Ren, W. Cheng, X. Luo, R. Shu, and X. Liu, "Modeling and analyzing the influence of chunk size variation on bitrate adaptation in dash," in *Proceedings of IEEE INFOCOM 2017*, pp. 1–9.

[8] S. De, A. Yadav, D. Jacobs, and T. Goldstein, "Automated inference with adaptive batches," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1504–1513.

[9] "Nvidia tensorrt documentation," https://docs.nvidia.com/deeplearning/tensorrt/best-practices/index.html#batching, Accessed: 2021.

[10] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, vol. abs/1804.02767, 2018.

[11] M. G. S. Bhatnagar, R. S. Sutton and M. Lee, "Natural actor–critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *the Computing Research Repository*, vol. abs/1707.06347, 2017.

[13] H. Choi, J. Yoo, J. Nam, D. Sim, and I. V. Bajić, "Pixel-wise unified rate-quantization model for multi-level rate control," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 1112–1123, 2013.

[14] Y. Sun, L. Li, Z. Li, and S. Liu, "Referenceless rate-distortion modeling with learning from bitstream and pixel features," in *Proceedings of ACM MM 2020*, pp. 2481–2489.

[15] G. Correa, P. A. Assuncao, L. V. Agostini, and L. A. da Silva Cruz, "Pareto-based method for high efficiency video coding with limited encoding time," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 9, pp. 1734–1745, 2015.

[16] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *Proceedings of IEEE INFOCOM 2018*, pp. 1871–1879.

[17] G. V. de la Cruz Jr, Y. Du, and M. E. Taylor, "Jointly pre-training with supervised, autoencoder, and value losses for deep reinforcement learning," in *Proceedings of ALA Workshop at AAMAS 2019*, pp. 1–8.

[18] A. Galanopoulos, J. A. Ayala-Romero, G. Iosifidis, and D. J. Leith, "Automl for video analytics with edge computing," in *Proceedings of IEEE INFOCOM 2021*, pp. 1–10.

[19] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of ACM SIGCOMM 2018*, pp. 253–266.

[20] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proceedings of ACM SIGCOMM 2018*, pp. 236–252.

[21] 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.

[22] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of ACM SenSys 2015*, pp. 155–168.

[23] 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 *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017, pp. 377–392.

[24] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proceedings of IEEE INFOCOM 2018*, pp. 1421–1429.

[25] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

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

[27] S. Jain, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. Gonzalez, "Scaling video analytics systems to large camera deployments," in *Proceedings of HotMobile 2019*, pp. 9–14.

[28] 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 ACM SIGCOMM 2020*, pp. 359–376.

[29] C. Pakha, A. Chowdhery, and J. Jiang, "Reinventing video streaming for distributed vision analytics," in *10th USENIX Workshop on Hot Topics in Cloud Computing*, 2018.