Profiling-free Configuration Adaptation and Latency-Aware Resource Scheduling for Video Analytics

Tian Zhou*, Fubao Wu*, Lixin Gao*
* Department of Electrical and Computer Engineering, University of Massachusetts Amherst, USA
tzhou@umass.edu, fubaowu@umass.edu, lgao@ecs.umass.edu

2022 IEEE International Conference on Big Data (Big Data) | 978-1-6654-8045-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/BigData55660.2022.10020573

***Abstract*—With increasingly deployed cameras and the rapid advances of Computer Vision, large-scale live video analytics becomes feasible. However, analyzing videos is compute-intensive. In addition, live video analytics needs to be performed in real time. In this paper, we design an edge server system for live video analytics. We propose to perform configuration adaptation without profiling video online. We select configurations with a prediction model based on object movement features. In addition, we reduce the latency through resource orchestration on video analytics servers. The key idea of resource orchestration is to batch inference tasks that use the same CNN model, and schedule tasks based on a priority value that estimates their impact on the total latency. We evaluate our system with two video analytic applications, road traffic monitoring and pose detection. The experimental results show that our profiling-free adaptation reduces the workload by 80% of the state-of-the-art adaptation without lowering the accuracy. The average serving latency is reduced by up to 95% comparing with the profiling-based adaptation.**

I. Introduction

Millions of cameras have been deployed in various areas such as road intersections, airports, factories, homes, and classrooms. The proliferation of cameras makes video analytics possible for diverse applications including traffic control, business intelligence, crime prevention, smart factories, and distance education [1]. For example, Microsoft collaborated with the City of Bellevue in Washington State to use the widely deployed traffic camera feeds to produce actionable insights for the goal of zero traffic deaths and serious injuries by 2030 [2]. They deployed a video analytics platform to produce directional counts of traffic users (vehicles, bicycles, etc.), and live alerts on abnormal traffic volumes. Video analytics provides organizations with hindsight, insight, and foresight into their operations through automatically analyzing, detecting, and trigger alerts seen by cameras, and are poised to revolutionize the efficiency and effectiveness of video surveillance technologies.

Video analytics relies on Computer Vision (CV) techniques such as object detection and classification. Recent advances in computer vision and deep learning provide increasingly accurate deep neural networks (DNNs) for these tasks. However, DNN models usually demand more compute power and is usually handled with Graphics Processing Units (GPUs). Currently, it is not feasible for an edge camera (even the most powerful camera) to perform real-time object detection with these DNN models. For example, the latest Raspberry Pi 4B model augmented with the Intel Movius Neural Compute Stick, which is a plug-and-play hardware to accelerate neural network computation, can process only 2-4 frames per second with YOLOv3-tiny model [3], [4]. Therefore, smart cameras typically offload analytics tasks to an edge server, which empowered with powerful computing resource and is located close to the camera on the Internet.

Building a video analytic system on an edge server that supports tens or hundreds of cameras can be challenging. First, analyzing video streams is compute-intensive. For example, GeForce GTX 1080 Ti GPU can only process 33 frames per second with YOLOv4 [5]. Even powered with a high-end GeForce RTX 2080 Ti GPU, an edge server can process only 62 frames per second [6]. This is barely enough for serving two 30 frame-per-second live video streams. Second, the DNN models might need to be adapted over time due to changes in video content. For example, when a traffic congestion happens, the density of vehicles greatly increases and a powerful DNN model may be needed. Meanwhile, when there are very few vehicles on roads, such as at midnight, a simple DNN model might be accurate enough to identify all vehicles. Third, the resource orchestration for performing live video analytics on multiple streams is challenging. The resource demand of different stream may vary significantly. It is challenging to schedule computing resources to maintain good overall throughput while preventing high latency.

We propose a live video processing system for multiple video streams on edge servers. First, we propose a configuration adaptation scheme that does not need to perform profiling online. The configuration adaptation is to select a configuration, such as the video resolution and frame rate, for video analysis that provides an acceptable accuracy. The state-of-the-art adaptation methods [7]–[9] profile video periodically to select a configuration. These schemes periodically profile the workload and accuracy of each candidate configuration by analyzing sampled video clips. These profiling-based methods incur heavy workloads at the profiling points. Different from these profiling-based adaptation methods [7], [8], [10]–[12], we propose a profiling-free adaptation that selects a config-

uration using a prediction model based on object movement features. We find that the best configuration highly correlates with the movement speed of objects in the video. For example, when the key object in a video moves slowly, we can process the video by sampling some frames periodically without losing track of the object. We train a prediction model to capture the relationship. Compared to the profiling-based adaptations, our method requires much less computing resources. And it does not require a burst of computation resources during profiling. Further, we can perform configuration adaptation more frequently than periodic profiling adaptations. As a result, our method reacts more promptly to pattern changes of object movement.

Second, we design a task processing engine for resource orchestration. Our task processing engine schedules GPU resources for video analytics on adapted video streams, where consecutive frames may be adapted to various resolutions. We design a batching mechanism. It picks and batches frames according to their resolutions and target DNNs, instead of their source streams. This enables us to perform batch-processing on adapted video streams. In addition, we propose a prioritized scheduling policy. Different from typical scheduling policies that optimize the latency of serving each frame, our policy takes the latency of a stream into consideration. As a result, we provide lower overall latency.

We evaluate our system with two live video analytic applications: traffic monitoring and pose detection. Our evaluation results show that our profiling-free adaptation approach is able to reduce the workload by about 80% of the periodic profiling adaptation, without reducing the accuracy of video analytics. With our resource orchestration, the average serving latency is reduced by 95% comparing with the state-of-the-art configuration adaptation method. We are able to analyzes ten 720p traffic monitoring video streams with average latency of 200ms with one GPU.

The rest sections of this paper are organized as follows. In Section II, we briefly introduce video analytics on edge servers about common techniques and challenges. Then, we introduce our profiling-free configuration adaptation idea for live video analytics in Section III, and the resource scheduling for adapted video frames in Section IV. After that, we implement a profiling-free video analytic system PFad in Section V and evaluate it in Section VI. Finally, we introduce related works and conclude our work in Section VII and VIII respectively.

II. Motivation

Cameras have been widely deployed and empowered many video analytic applications. But smart cameras are usually not powerful enough to analyze live video streams.

It is common to use edge servers to provide computing power for video analytics. These edge servers locate close to cameras. So that cameras can stream their video data to the edge server in real time. We illustrate the edge server systems that serve video analytics in Fig. 1. Usually, an edge server is responsible for analyzing videos from several cameras.

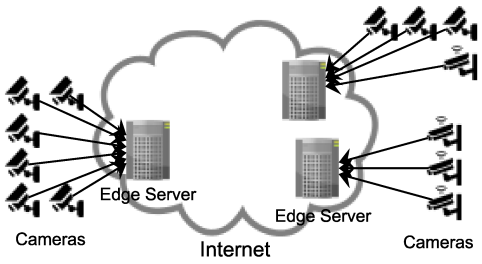


Fig. 1. Video Analytics on Edge Servers

Video analytics utilize computer vision techniques such as object detection, image classification, and optical character recognition. With recent advances in computer vision, many powerful DNN models with high detection accuracy have been proposed. However, inference with DNN models is compute-intensive. An edge server usually needs to equip with specialized hardware, such as Graphics Processing Units (GPUs), to analyze videos with DNN.

Simply equipping with some GPUs is usually not enough to fulfill DNN inference for multiple video streams. For example, YOLO and faster-R-CNN are the state-of-the-art DNN models for object detection. A high-end GeForce RTX 2080 Ti GPU can process 62 frames per second with YOLOv4 [6]. That supports just two streams of 30-FPS videos. A less powerful GPU model, such as GeForce GTX 1080 Ti, just runs 33 FPS [5]. The faster-R-CNN model is a more accurate but slower object detection model. A faster-R-CNN model with ResNet-101 just processes 20 frames per second on an NVIDIA Tesla V100 GPU (about 20% faster than its commodity vision NVIDIA GeForce 1080 Ti) [13].

In many cases, it is possible to downgrade the frame rate and resolution without affecting the video analytic results. For example, in a speeder detection application, we want to detect vehicles that move faster than a predefined speed limit and capture their plates. The video stream from the surveillance camera is in resolution 1920x1080 and 30 frames per second (FPS). Actually, for the purpose of identifying cars, it is accurate enough to use a lower resolution such as 854x480. The corresponding object detection model usually costs less computing resource. And it is usually good enough to track vehicles using 10 frames per second instead 30.

The best frame rate and resolution required at different video segments can also be different. In the above example, when a traffic jam happens or the traffic light is red, all vehicles moves slowly. We can identify and track all vehicles at the frame rate of 1 or 2 FPS. And only when a speeder is found, we need to use a high resolution to read its plate. Otherwise, we can just use the low resolution to identify cars.

Configuration adaptation has been proposed for video analytics. State-of-the-art adaptation methods choose frame rate and resolution dynamically through periodical profiling. They compute the accuracy the workload on the first few seconds of each video segment for all candidate configurations [1],

[7], [12], [14], and then select the best one based on certain accuracy or workload requirements. Despite many ideas are proposed for reducing the number of candidate configurations to profile [1], [7], [11], there are still tens of configurations to examine. Since the profiling procedure performs the analytics for all candidate configurations, it is compute-intensive.

An edge server needs to analyze several video streams. These streams compete for the computing resources of GPUs on the edge server. To make better use of GPU computing resources, frames should be packed as batches before submitting to a GPU. For example, compared to submitting frames one-by-one, the throughput of a ResNet-152 model triples when frames are submitted as batches of 8-frames on an NVIDIA Tesla K80 GPU [15]. But only frames of the same size can be batched. Frames from different streams may be in different sizes. After adaptation, even consecutive frames of the same stream may be of different resolutions. Different from the image analytics, frames in a video are related. Their sequential order matters for video analytics. So the scheduling of frame analytic tasks is crucial for edge servers.

III. Profiling-Free Configuration Adaptation

In this section, we propose a profiling-free configuration adaptation for video analytics. We first introduce the idea of adapting with a object movement-based configuration prediction model. Then, we describe movement features and the prediction model.

A. Profiling-free Adaptation

Consecutive frames in a video are correlated. Usually, it is unnecessary to analyze every frame to locate and track objects in the video. Even when a video frame is analyzed, it does not have to be analyzed with the highest resolution. In this paper, we refer to the combination of frame rate and resolution as a *configuration*. Analyzing a video with a lower resolution or lower frame rate costs less computing resources. So, state-of-the-art video analytic systems adapt the configuration to speed up the video analytics [1], [7], [11], [12], [14].

Current configuration adaptation methods are profiling-based. They select configurations by profiling the first few seconds of a video segment. By analyzing the same piece of video with a set of candidate configurations, they are able to choose the configuration that satisfies the analytic accuracy requirement and costs the least amount of computing resources. But the profiling itself is very costly in terms of computing resources.

We notice a strong correlation between the object movement and the configuration. For example, Fig. 2 shows the maximum speed of cars and mean distances among cars in a traffic surveillance video. It also shows the best configuration (the fastest configuration that satisfies an accuracy requirement) for a car counting application. As shown in the figure, in most parts of the video, cars move slowly. The low configuration of 426x240 resolution and 2 FPS is enough to count all cars. At second 100 the traffic light turn green, cars start to speed up. Meanwhile, new cars keep getting in the screen and old

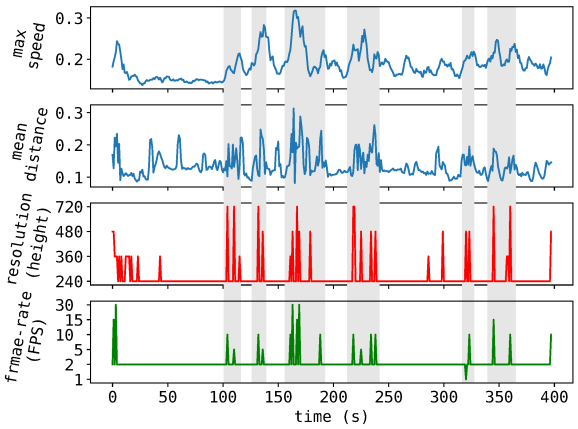


Fig. 2. Correlation between object movement features and minimum configuration

cars keep moving out of the screen. As highlighted with gray background, the maximum speed drastically changes in the period 100-250 second. Correspondingly, the best frame rate changes when the maximum speed changes. Similarly, when the mean distance among cars suddenly drops or increases, we need a higher resolution.

We propose a profiling-free adaptation mechanism that predicts configuration based on movement features of video objects. By learning the relationship between the movement of objects and the best configuration, we predict the configuration instead of choosing one by profiling all candidate configurations. Then, we generate analytic tasks of frame based on the predicted configuration.

The movement of objects can be got from the intermediate video analytic results without additional cost. Object identification and tracking are usually the core operations of video analytics. The movement trajectory from the tracking operation cannot only be used to compute the analytic results, but also be used to extract movement features such as speed. We capture the relationship between the movement features and the best configuration with a machine learning model. Then, we can use the model to predict the configuration for the next video segment based on the current movement features.

B. Movement-Based Video Features

We extract the movement of all video objects within a time slot into movement features. We model two types of features, object-level and video-level. The object-level features describe the information of each individual object. The video-level features describe the information across objects. For example, it captures the distance among objects.

We extract the feature based on the movement trajectory of each object. The trajectory of an object k describe its center position $\vec{p}(k)$, height $h(k)$ and width $w(k)$ of the object in each frame. It is formulated with a sequence of pairs $[\cdots, (\vec{p}_i(k), h_i(k), w_i(k)), \cdots]$ where i is the frame index. To describe information such as movement speed, we also need the time of each frame t_i.

1) Object-Level Features: For each object, we capture four features, including its size and three kinds of movement. Since the number of objects in each time slot is dynamic and the number of analyzed frames in each time slot is also non-predefined, we summarize the information of each object in each frame to get the feature for that time slot.

• *Size*. The size of objects directly relate to the resolution. An object k's size $s_i(k)$ in frame i can be derived with the following equation.

$$s_i(k) = w_i(k) \cdot h_i(k) \tag{1}$$

Denoting the size of each object in each frame with a vector $\mathbf{S}$, we summarize the size feature $\mathbf{f_s}$ with its sufficient statistics $\mathbf{f_s}$ as Equ. (2).

$$\mathbf{f_s} = [\min(\mathbf{S}), \text{mean}(\mathbf{S}), \max(\mathbf{S}), \text{std}(\mathbf{S})] \tag{2}$$

• *Position Change Rate (xy-plane Velocity)*. The physical movement of an object in the real world can be projected to the movement in the frame and movement perpendicular to the frame. We capture the movement in frames using the position change rate. The position change of object k at frame i is computed with its position $\vec{p}_i(k)$ and time t_i as the following equation.

$$\vec{v}_i(k) = \frac{\vec{p}_i(k) - \vec{p}_{i-1}(k)}{t_i - t_{i-1}} \tag{3}$$

Its L2-norm $||\vec{v}_i(k)||$ is its position change rate. Similar to the size feature, we summarize the position change feature $\mathbf{f_v}$ with its sufficient statistics using Equ. (2).

• *Size Change Rate (z-axis Velocity)*. We capture an object's movement perpendicular to the frame with its size change rate. Using the size computed with Equ. (1), we get the size change rate $z_i(k)$ of object k at fame i with following equation.

$$z_i(k) = \frac{s_i(k) - s_{i-1}(k)}{t_i - t_{i-1}} \tag{4}$$

We summarize size change rate feature as $\mathbf{f_z}$.

• *Aspect Ratio Change Rate (Rotation and Morphing)*. Movement consists of translation and rotation. The shape change is a quick identifier for object rotation. It can be captured with the change of aspect ratio. In addition, the change of aspect ratio also indicates the morphing of an object, which usually relates to the movement of the object parts such as the limbs of a human object. We compute the change rate of the aspect ratio $a_i(k)$ for object k at frame i as the following equation.

$$a_i(k) = \frac{w_i(k)/h_i(k) - w_{i-1}(k)/h_{i-1}(k)}{t_i - t_{i-1}} \tag{5}$$

We summarize the aspect ratio change rate feature as $\mathbf{f_a}$.

2) Video-Level Feature: The video-level features capture information from a global view, such as the spreed of objects in a frame. In addition, we also consider global information such the previous configuration. Different from object-level features, the video-level features usually capture the relationship among objects. So for each frame, there is a $n \times n$ matrix to describe the movement information, suppose there are n objects in the frame. We flatten the matrix sequence to summarize them.

• *Distance Distribution*. The location of objects affects the resolution. When two objects stick close to each other, we may need a higher resolution to distinguish them. Similarly, we may also need a higher resolution when objects spreed away from each other. Suppose there are k objects in frame i, $d^{(i)}_{a,b}$ denotes the distance between object a and b in frame i, and $\mathbf{D_i}$ denotes the pairwise distance matrix of all objects.

$$d^{(i)}_{a,b} = ||\vec{p}_i(a) - \vec{p}_i(b)|| \tag{6}$$

$$\mathbf{D_i} = \begin{bmatrix} 0 & d^{(i)}_{1,2} & d^{(i)}_{1,3} & \cdots & d^{(i)}_{1,k} \\ 0 & 0 & d^{(i)}_{2,3} & \cdots & d^{(i)}_{2,k} \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{7}$$

For a time slot starting from frame f and ending at frame l, we extract all the non-trivial distances into a vector $\mathbf{D}$ as follows.

$$\mathbf{D} = [\text{triur}(\mathbf{D_f}), \text{triur}(\mathbf{D_{f+1}}), \cdots, \text{triur}(\mathbf{D_l})] \tag{8}$$

Then, we summarize the distance distribution feature over $\mathbf{D}$ as $\mathbf{f_d}$ using Equ. (2).

• *Relative Velocity Distribution*. The relative movement between objects provides additional information to the individual movement. For example, when two objects are moving towards each other and about to overlap, we need to use higher frame rate and higher resolution in the future time slots. We denote the relative velocity between objects a and b in frame i as $r^{(i)}_{a,b}$ as follows.

$$r^{(i)}_{a,b} = ||\vec{v}_i(a) - \vec{v}_i(b)|| \tag{9}$$

We can build up a matrix $\mathbf{R_i}$ about the relative movement velocity among all object pairs for each frame i. Using the same extraction method as the distance distribution feature, we extract the non-trivial entries into a vector $\mathbf{R}$ and summarize the relative velocity distribution feature as $\mathbf{f_r}$.

• *Number of Objects*. In addition to the movement features, we include the number of detected objects in the current time slot. A larger number usually indicates a higher volume and a higher frame rate. We denote the number of objects in frame i with n_i and the sequence of each selected frame as $\mathbf{N}$.

$$\mathbf{N} = [n_f, \cdots, n_l] \tag{10}$$

We summarize the number feature as $\mathbf{f_n}$.

• *Configuration*. We also consider the current configuration as part of the global-scope feature. Since the detection is performed on the adapted frames, the adaptation configuration affects the detection results and extracted movement features. So it provides confidence information for the extracted movement features. Thus, we add the resolution and frame rate with its corresponding movement features. Since time slot is the basic unit of adaptation, the resolution and frame rate of a time slot is static.

$$\mathbf{f_c} = [r, f] \tag{11}$$

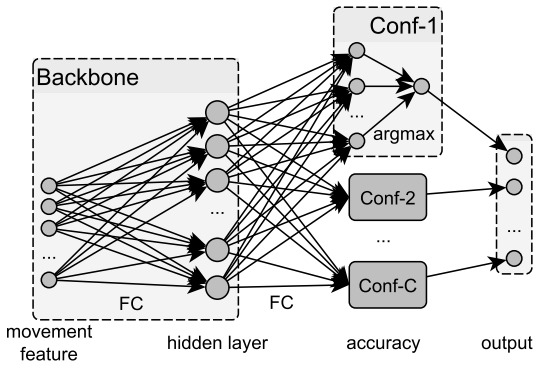


Fig. 3. Prediction Model for Accuracy

3) Summary of Features: To predict the configuration for the next time slot, we consider the feature of several recent time slots. For each time slot i, we get its unit feature $\mathbf{f}^{(i)}$ as the composition of the features mentioned above.

$$\mathbf{f}^{(i)} = [\mathbf{f_s}^{(i)}, \mathbf{f_v}^{(i)}, \mathbf{f_z}^{(i)}, \mathbf{f_a}^{(i)}, \mathbf{f_d}^{(i)}, \mathbf{f_r}^{(i)}, \mathbf{f_n}^{(i)}, \mathbf{f_c}^{(i)}] \quad (12)$$

The unit feature of one time slot just describes the video dynamics in a short period. It can hardly describe the patterns lasting for several time slots. Therefore, we derive the feature of a time slot as the composition of unit features of the most recent k time slots.

$$\mathbf{F}^{(i)} = [\mathbf{f}^{(i-k+1)}, \cdots, \mathbf{f}^{(i-1)}, \mathbf{f}^{(i)}] \quad (13)$$

C. Configuration Prediction Model

We predict the accuracy of each configuration and select one for the next time slot based on the predictions. We design a neural network to predict the accuracy from features. So given a minimum acceptable accuracy, we select the fastest configuration that satisfies the requirement. To make the model simpler, we discretize the accuracy value and predict its range instead of the accuracy value. We pay more attention to the range where people care the most. For example, we have ranges: 0-70%, 70%-80%, 80%-85%, 85%-90%, 90%-95%, 95%-100%.

Basically, we can predict the accuracy of each configuration with a neural network. In order to ensure fast prediction, we keep the neural network simple.

Inspired by the deep neural network design of the computer vision field [16], [17], we extract the common computation of all configurations as a backbone network. The accuracy prediction part of each configuration shares the output of the backbone network. As shown in Fig. 3, we process the input features with a fully-connected layer and derive an intermediate feature in the hidden layer. Then, each configuration predicts its own accuracy based on the hidden layer.

IV. Scheduling for Adaptation Tasks

The adaptation brings additional challenges for serving analytic tasks with low latency. First, the adaptation makes the resolution of consecutive frames different, and correspondingly they are usually handled by different DNN models. It prevents us from getting analytic results of a video segment in one batch. While processing frames one-by-one or with smaller batches results in higher overhead. So we adopt a multiple-queue mechanism that batches frames according to their target model regardless of their source streams and temporal order.

The cross-stream batching technique improves the overall throughput at the cost of increased serving latency in extreme cases. Frames are independent during DNN inference. But the temporal order is important for following steps such as tracking. The tracking step requires the detected objects to be submitted in the right temporal order. Otherwise, identical objects in different frames may not be able to be associated correctly. But consecutive frames in a stream may be put into different queues based on their resolutions. If the queue with later frames are processed by the GPU first, the processing results cannot be submitted to the tracking step until the earlier frames are processed. That introduces additional latency. And in some extreme cases, that leads to starvation of some video streams.

Existing scheduling policies, such as the first-come-first-serve (FCFS) policy, prevent the task starvation. When the GPU is available, we can choose the queue whose first batch waits the longest time. Since frames of the same stream arrives in the temporal order, we can guarantee that a frame is processed at most $(N-1)B$ frames behind its precedent frame, where N is the number of queues and B is the maximum number of frames in each batch.

But the FCFS policy does not consider the processing time of batches from different queues or the current workload of the system. As a result it may lower the average latency of the whole system. The shortest-job-first (SJF) policy reduces the global latency. The DNN model for an image with smaller resolution is usually simpler and takes less inference time. Therefore, given the same amount of waiting tasks, processing the queue of a smaller resolution can result in lower global latency. However, in many cases, the stream is adapted to a relatively low configuration for most cameras. Such as the traffic volume is low on all roads at midnight. So corresponding queues hold more tasks than others. Scheduling such queues earlier reduces the waiting time of all tasks and correspondingly the overall latency.

We propose a latency-aware scheduling policy that aims to reduce the overall stream-level latency. It selects queues based on how the selection affects the overall latency. In our scheduling, the expected processing time of each selected batch is known, and the processing is non-preemptive. First, we avoid starvation while pursuing high throughput. Since the expected processing time of a batch is known, we can compute how much total waiting time is removed by spending one unit of processing time, which is the return-on-investment of selecting a queue. For the efficiency of computation, we approximate the total waiting time of the first batch in a queue with its upper bound, which is the waiting time of the first task in the queue. Therefore, we have the return-on-investment of a queue i as t_i^q/t_i^p where t_i^q is the waiting time of the first

task in queue i, and t_i^p is the expected processing time of the batch.

For every stream, frame analytic results are used in the temporal order. Each stream waits for its next frame analytic results for tracking. Suppose there are two batches with same return-on-investment. One batch contains n frames from the same stream, while the other one contains the first frame of n different stream. By processing the second batch, the tracking and successive stream-level analytic steps of n streams are able to move forward. While, for the first batch, only when the n frames are consecutive and are the first n unprocessed frames, the first batch put forward the stream-level processing by n frames. Otherwise, less than n frames are pushed forward. Therefore, the total stream-level latency of selecting the second batch is no worse than selecting the first one. When configuration switch happens, it is better.

We put the stream-level latency into consideration. The oldest unprocessed task of a stream is the critical task of the stream. Processing a batch with more critical tasks results in better overall performance. Therefore, we propose the following priority function of each queue i as Equ. (14).

$$p_i = l_i \cdot t_i^q / t_i^p \tag{14}$$

where l_i is the number of critical tasks in the first batch of queue i. It can be derived by maintaining a task pointer for each stream.

The policy balances the overall latency of all streams and the stream-level latency of each stream. When the workload is light, the $1/t_i^p$ term dominates. Our policy tends to select the queue with a smaller processing time, which is close to the SJF policy. It makes better use of batch processing. But different from SJF, due to the existence of the $l_i \cdot t_i^q$ term, our policy does not starve any queue. It is easy to prove that as long as the system's processing capacity is not overwhelmed, the maximum delay between consecutive frames in a stream is $\frac{N \cdot B \cdot \max_i\{t_i^p\}}{\min_i\{t_i^p\}}$ where N is the number of queues and B is the maximum batch size. When the workload is heavy, the $l_i \cdot t_i^q$ term dominates. Our policy selects the queues with a longer total waiting time.

V. PFad: Profiling-Free Video Analytics System

In this section, we introduce our design for video analytics on edge servers with profiling-free adaptation, PFad. We first give an overview of our design. Then, we introduce the processing pipeline of a video stream using our profiling-free adaptation mechanism. After that, we describe the task processing engine module in our system, which manages computing resources and serve DNN tasks of all video streams.

An edge server handles the real-time analytics of multiple video streams. Due to the data loading and CUDA kernel initialization overhead, submitting images as batches to a DNN model usually brings higher throughput on a GPU. However, allowing the analytic pipeline of each video stream to access the GPU is inefficient. First, batching frames also introduces additional waiting time. If the frame rate of a video is not high enough, non-batching submission may result in lower latency. Second, tasks from different streams compete for the computing resource. Lowering the latency of a stream by disabling batching costs more computing resource. That further jeopardizes the performance of other streams.

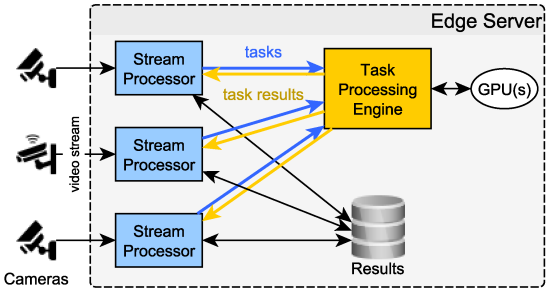


Fig. 4. Overview of PFad

To serve multiple streams efficiently, we decouple the stream processing and DNN processing. Fig. 4 shows the basic structure of our system. Each video stream is handled by a dedicated stream processing module, referred to as a *stream processor*. It parses the video stream and generates frame-level analytic *tasks*. Then, tasks are submitted to a *task processing engine*, which manages the GPU(s) and serves tasks of all streams. After the tasks are served, their results are sent back to their source stream processors for further processing.

The stream processor serves the analytics of a video stream. Fig. 5 illustrates the structure of a stream processor. It analyzes a video stream with a pipeline shown with gray boxes. First, it decodes the incoming video stream and get individual frames. The raw frames are adapted by sampling and resizing according to the configuration predicted by profiling-free configuration adaptation. Then, the adapted frames are encapsulated into analytic tasks and submitted to the task processing engine. Then, objects in frame analytic results are processed through tracking. It associates identical objects in different frames and outputs the trajectory of each unique object. After that, objects and their trajectories are converted to the application-specific analytic results. For example, we can use an object detection model for cars on a traffic surveillance video stream. With the car detection results in each frame, we can get their trajectories through tracking. Then, with the trajectories, we can get the traffic volume by counting the number of unique trajectories, or detect speeders by examining the speed using car trajectories.

The stream processor dynamically adapts the configuration for the DNN processing phase. It gets the object movement features through the object trajectories, which is the intermediate results of object tracking. Then, we predict the configuration using the profiling-free configuration adaptation described in Section III. The feature is extracted from the result of frame DNN analytic and it takes time. It may involve some additional latency for waiting for features of configuration prediction. To avoid the latency, we decouple the configuration prediction and video adaptation with two parallel threads. The configuration prediction happens when necessary object

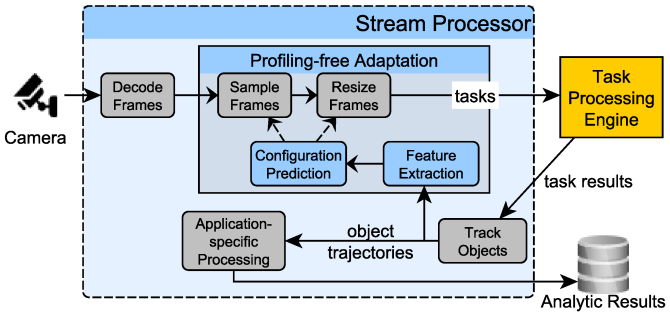


Fig. 5. Stream Processor

trajectories are ready. And we use a buffer to remember the latest predicted configuration. When a configuration is needed in the adaptation thread, we use whatever we have currently.

We delegate the GPU access of all tasks from video streams with a unified task processing engine. It receives frame analytic tasks from stream processors and organizes them into task queues. Then, it schedules the tasks as batches and serves them on GPU. After that, the engine sends task results back to their source stream processors.

The task processing engine focuses on reducing the latency of each stream by orchestrating GPU. First, it reduces the waiting time and total computing time by batching tasks regardless of their source streams and temporal order. However, the out-of-order batching may introduce some reordering latency. As a result, the stream-level increase. So secondly, to reduce the stream latency, we use the latency-aware scheduling policy described in Section IV.

VI. Evaluation

In this section, we evaluate our system. We first analyze the performance of the configuration prediction model. Then, we compare the total workload and video analytic accuracy of our profiling-free adaptation with those of the profiling-based adaptation. After, that we compare the latency of different scheduling policies.

A. Settings

We evaluate our video analysis system with two live video analytic applications on a server with Quadro RTX 5000 GPU. We perform a traffic monitoring application and a pose detection application. We collect traffic surveillance videos and dancing videos from YouTube. TABLE I shows more details of our evaluation dataset. For the traffic monitoring application, we count the number of unique vehicles that pass through the camera. We count the ground truth number by human-labeling. For the pose detection application, we estimate the position of the 17 key points defined by COCO [18] for all frames. We predict poses for skipped frames using a speed-based position estimation [19] and compare them with ground truth poses via the object keypoint similarity (OKS) metric [20]. We collect the ground truth poses of each frame using the raw resolution, which is usually 1080p or 4K. To thoroughly test the performance of our system, we exclude the raw resolution from the adaptation knobs.

TABLE I
Video Dataset Summary

Application	length (min.)	# clips	# objects (per frame)
Traffic Monitoring	600	90	0 - 20
Pose Detection	750	75	0 - 1

The setting of the stream processor is listed as follows. We adapt among 4 resolutions [240p, 360p, 480p and 720p] and 6 frame rates [1, 2, 5, 10, 15 and 30] FPS. We perform the configuration adaptation every second. We use the cascading accuracy thresholds for adaptation as $[0.95, 0.9, 0.8, 0.7, 0.5]$. It means that if no configuration is predicted to be more accurate than 0.95, we will try to select the fastest one with accuracy no less than 0.9. If not one satisfies 0.9, we try the next threshold 0.8. We train the prediction model for each application separately.

We compare our system with the periodic profiling configuration adaptation method. Since the traffic light and dancing pattern usually change at the level of several tens of seconds, We select a configuration every 30 second. We implement the periodic profiling adaptation with state-of-the-art optimizations. We first reduce the configuration space using the Pareto principle through an offline profiling [1], [7], [11]. Then, we compute a frame alignment table for each resolution based on the candidate configurations. It points out the common frame used by different frame rates. So during online profiling, we just need to process those frame once. For example, given a 30FPS video, to profile configuraitons 480p-15FPS, 480p-10FPS, 480p-5FPS, 480p-3FPS,480p-1FPS, we just need to process 15+1=16 frames per second instead of 15+10+5+3+1=34 frames.

We implement our system in Python and utilize existing object detecting models. We employ the YOLOv5-median model [21] for vehicle detection and the OpenPose model [22] for pose detection. For the tracking phase after the object detection, we adopt the SORT algorithm [23] for tracking bounding boxes (vehicle monitoring) and the nearest-neighbor algorithm for key points (pose detection). The code is open-source and public available [1].

B. Impact of Features on Configuration Prediction

We examine the impact of different features on the configuration prediction accuracy. We compare the prediction accuracy of prediction models trained with only parts of features in TABLE II. Some different configurations may result in roughly the same video analytic accuracy and workload. So we consider a prediction is correct as long as the predicted configuration's video analytic accuracy and workload is 5% and 10% around the ground-truth configuration respectively.

When all features are used, we archive the prediction accuracy of 0.622 and 0.679 on the traffic monitoring and pose

[1]Source Code: https://github.com/yxtj/VideoServing

TABLE II
CONFIGURATION PREDICTION ACCURACY USING DIFFERENT FEATURES

Feature	Traffic Monitor	Pose Detection
Object-level only	0.582	0.611
- no PCR (xy-move)	0.275	0.193
- no SCR (z-move)	0.511	0.495
- no ACR (rotation)	0.518	0.512
Video-level only	0.356	0.534
All features	0.622	0.679

detection application respectively. The accuracy drops about 0.05 - 0.25 if we use the object-level or video-level features only. In both applications, the prediction model trained with object-level features outperforms the model using video-level features. Compared with pose detection, the traffic monitoring application benefits less from the video-level features. In the traffic monitoring application, vehicles on the same road roughly follow the same movement pattern and there are on strong interactions among cars. The object-level features are good enough to capture the movement pattern. While for dancing, the joints of a human body do not share the same moving pattern and the movement of human body is more complex. So the relative movement captured by the video-level features also plays an important role in deciding the configuration.

We further analyze the impact of different types of movement captured in the object-level features. We remove the position change rate (PCR), size change rate (SCR), and aspect ratio change rate (ACR) from the object-level features respectively, and compare the corresponding prediction accuracy. They indicate the movement in the frame (x- and y-axis), movement perpendicular to the frame plane (z-axis), and the rotation of objects respectively. As shown in TABLE II, after removing the position change rate, the prediction accuracy significantly drops to about 0.2 from 0.6 above. While removing the size change rate or aspect ratio change rate decreases the accuracy by about 0.1.

C. Performance of Profiling-Free Adaptation

We compare our profiling-free adaptation with the periodic profiling adaptation. Shown in Fig. 6, we compare the compute resource requirement and accuracy for each second of an example video clip on both the traffic monitoring and pose detection. As shown in both Fig. 6(a) and 6(c), the profiling-free adaptation significantly reduces the resource demand. For the traffic monitoring application, The average resource requirement of the periodic profiling adaptation is 428.80 GFLOPs (giga floating point operations) while that of our profiling-free adaptation is 55.57 GFLOPs, which is about just 13% of the profiling-based method. For the pose detection, our profiling-free adaptation requires about 25% of workload compared to the periodic profiling adaptation on average. In addition, the profiling-free adaptation also significantly reduces the peak resource requirement. On the two example videos, the peak resource requirement of the profiling-free adaptation is just about 2.5% and 6% of the periodic profiling method. It is because that the periodic profiling adaptation needs to profile all candidate configurations at the beginning of each segment. While the profiling-free adaptation just processes the video with one configuration at any time. The lower peak resource requirement helps in lowering the overall serving latency. We show it in later sections.

The video analytic accuracy using the profiling-free adaptation is higher than that of the periodic profiling adaptation. Fig. 6(b) and 6(d) show the video analytic accuracy of each video second. In most time, the profiling-free adaptation performs the same as the periodic profiling adaptation, since the two adaptation methods choose the same configuration at the time. For about 20% time, the profiling-free adaptation choose a different configuration. Because the profiling-free adaptation adapts the best configuration more frequently, it reacts the content changes more quickly. So for those period, the profiling-free adaptation reaches higher video analytic accuracy on average. For traffic monitoring, the profiling-free adaptation increases the average accuracy from 0.846 to 0.869. And for the pose detection application, it increases the accuracy from 0.828 to 0.845.

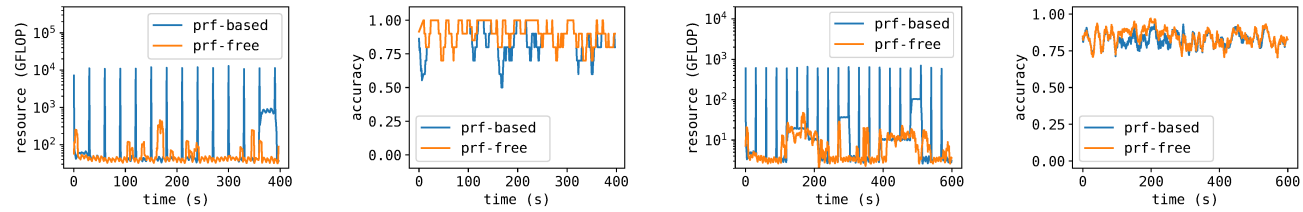
D. Impact of Latency-aware Scheduling

We compare the serving latency of our latency-aware scheduling policy with some popular scheduling policies. We launch 20 video streams for traffic monitoring and schedule them with the first-come-first-server (FCFS), short-job-first (SJF) and our latency-aware scheduling (LAS) policy. Therefore, we limit the maximum GPU usage to 110% of the average workload, so that the capacity matches the workload and the latency caused by busy parts in videos can be caught up later. and measure the average latency of all streams in every second in Fig. 7(a). We annotate the period during which the workload overwhelms the computing capacity with gray background. The FCFS policy performs the worst with average latency 0.338 second, while SJF and LAS reaches 0.297 and 0.300 second respectively. We zoom in to the busy periods (including 5 seconds after) and show the latency distribution in Fig. 7(b). Scheduling with the LAS policy, 99% tasks finishes in 1.299 second, while that number is 1.346 and 1.576 for the FCFS and SJF policy respectively.

The scheduling also affects the configuration prediction accuracy. It is because that movement features for configuration prediction are derived from frame analyzing results. And due to the serving latency, the movement feature is usually not computed from the latest frames. We compare the prediction accuracy under different scheduling policies for the traffic monitoring and pose detection application. Since our LAS policy reaches lower latency, its configuration prediction accuracy outperforms FCFS and SJF by about 5%. The experiment setting is same as the latency experiment above.

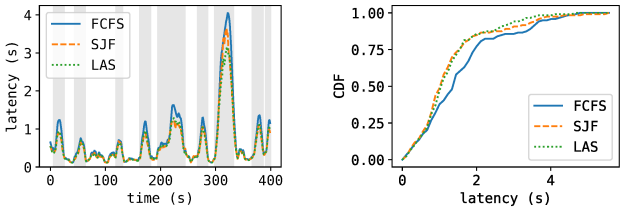
E. Serving Multiple Streams

We evaluate the latency of our system under 20 video streams. Shown in Fig. 8, the average latency of our system is 0.208 second and 0.387 second for traffic monitoring and pose



(a) Traffic Monitoring - Workload (b) Traffic Monitoring - Accuracy (c) Pose Detection - Workload (d) Pose Detection - Accuracy

Fig. 6. Performance of Profiling-Free Adaptation



(a) Average Latency Overtime (b) Latency in Busy Periods

Fig. 7. Latency of Different Scheduling Policies

TABLE III
CONFIGURATION PREDICTION ACCURACY UNDER SCHEDULING POLICIES

Policy	Traffic Monitor	Pose Detection
FCFS	0.561	0.632
SJF	0.599	0.628
LAS	0.614	0.661

detection application respectively. While the average latency of using the periodic profiling adaptation is 5.05 and 7.23 respectively. Our system is about 20x faster.

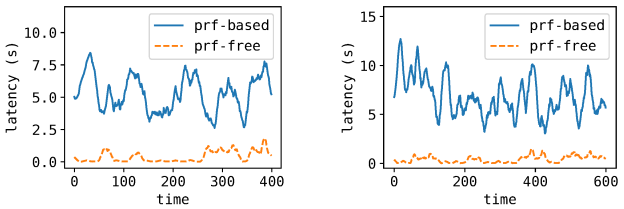
VII. RELATED WORKS

Configuration adaptation is an important part of video serving. It is used for both reducing the bandwidth consumption and compute resource requirement. VideoStorm [11] profiles a video stream offline and use the selected configuration for the whole lifespan of the video. Systems such as AWStream [7], Chamelon [8], JetStream [24], VideoEdge [1], JCAB [25] profile video streams online. They dynamically select the best configuration based on the profiling result of current video segment and current bandwidth. For better performance of online profiling, some works reduce the number of candidate configurations by combining the offline and online profiling [1], [7], [11].

The relationship between video content and configuration is studied in recently years. VCMaker [26] and Cuttlefish [27] utilize the speed of video objects together with bandwidth information to decide the configuration. They adopt reinforcement learning models to select configurations. Kim *et al.*propose to reduce the workload of profiling by utilizing the underlying characteristics including movement of video objects [28]. The work uses the movement information to estimate the performance of some configurations without actually profiling them. MIRIS [29] adjusts the frame rate for tracking queries. It uses a low frame rate to compute possible trajectories of cars and figures how to which part of the video they should look carefully to resolve the uncertainty.

Some researchers utilize the computing power on cameras to offload some computation workloads [1], [11], [30], [31]. Elf [32] executes counting queries completely on cameras. CloudSeg [33] reduces network traffic by uploading low-resolution frames and recovering them via super resolution. Reducto [14] dynamically filters frames on cameras by analyzing the time-varying relationship between video content, filtering threshold, and query accuracy. Zero-streaming [34] pre-processes videos and execute video queries using the computing power of both cluster and cameras.

Scheduling of DNN tasks is studied from many perspectives recently. InferLine [15] adjusts batch sizes and chooses different hardware types, including GPU, TPU, and FPGA. It employs both a low-frequency planer and a high-frequency planer to schedule DNN tasks. Nexus [35] schedules DNN tasks among a GPU cluster. It considers the data transfer across nodes while batching tasks. Yao's work [36] casts ML tasks as imprecise computations, each with a mandatory part and several optional parts. By scheduling among these parts, it achieves both good latency and accuracy.



(a) Traffic Monitoring (b) Pose Detection

Fig. 8. End-to-End Comparison of Serving Latency with 20 Streams

VIII. CONCLUSION

In this paper, we propose a video analytic system for massive video streams. First, we propose a profiling-free adaptation to reduce analytic workload without comprising accuracy. It selects configurations with a prediction model instead of profiling. Second, we design an out-of-order batching and prioritized scheduling policy for resource orchestration.

They improve the serving throughput and reduces the stream-level latency. We evaluate our system with real-world videos. Experiment results show that compared to the state-of-the-art profiling-based adaptation, our system improves the overall accuracy by about 3% while consumes about 20% workload. And our average latency for live analytics is about 20x smaller than state-of-the-art periodic profiling adaptation.

IX. Acknowledgment

This work was supported in part by National Science Foundation Grants CNS-1815412, CNS-1908536.

References

[1] C.-C. Hung, G. Ananthanarayanan, P. Bodík, L. Golubchik, M. Yu, V. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, October 2018, pp. 115–131. [Online]. Available: https://www.microsoft.com/en-us/research/publication/videoedge-processing-camera-streams-using-hierarchical-clusters/

[2] G. Ananthanarayanan, V. Bahl, Y. Shu, F. Loewenherz, D. Lai, D. Akers, P. Cao, F. Xia, J. Zhang, and A. Song, "Traffic video analytics - case study report," Tech. Rep. MSR-TR-1970-3, December 2019. [Online]. Available: https://www.microsoft.com/en-us/research/publication/traffic-video-analytics-case-study-report/

[3] "Intel neural compute stick 2 (intel ncs2)," https://www.intel.com/content/www/us/en/developer/tools/neural-compute-stick/overview.html, 2021.

[4] A. Rosebrock, "Yolo and tiny-yolo object detection on the raspberry pi and movidius ncs," https://www.pyimagesearch.com/2020/01/27/yolo-and-tiny-yolo-object-detection-on-the-raspberry-pi-and-movidius-ncs/, Jan 2020.

[5] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[6] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-YOLOv4: Scaling cross stage partial network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 13 029–13 038.

[7] 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*. ACM, 2018, pp. 236–252.

[8] 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*. ACM, 2018, pp. 253–266.

[9] F. Romero, M. Zhao, N. J. Yadwadkar, and C. Kozyrakis, "Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines," *arXiv preprint arXiv:2102.01887*, 2021.

[10] 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*, 2016, pp. 123–136.

[11] 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 ({NSDI} 17)*, 2017, pp. 377–392.

[12] H. Shen, S. Han, M. Philipose, and A. Krishnamurthy, "Fast video classification via adaptive cascading of deep models," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3646–3654.

[13] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*. Springer, 2020, pp. 213–229.

[14] 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 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 359–376.

[15] D. Crankshaw, G.-E. Sela, X. Mo, C. Zumar, I. Stoica, J. Gonzalez, and A. Tumanov, "Inferline: latency-aware provisioning and scaling for prediction serving pipelines," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 477–491.

[16] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[17] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[18] "Coco - common object in context," https://cocodataset.org/.

[19] F. Wu, L. Gao, T. Zhou, and X. Wang, "Motrack: Real-time configuration adaptation for video analytics through movement tracking," pending on GLOBECOM 2021.

[20] "Coco - keypoint evaluation," https://cocodataset.org/#keypoints-eval.

[21] G. Jocher, A. Stoken, J. Borovec, and et. al., "Yolov5," https://github.com/ultralytics/yolov5, https://doi.org/10.5281/zenodo.4418161.

[22] G. H. Martınez, "Openpose: Whole-body pose estimation," 2019.

[23] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*. IEEE, 2016, pp. 3464–3468.

[24] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and degradation in jetstream: Streaming analytics in the wide area," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, 2014, pp. 275–288.

[25] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 257–266.

[26] N. Chen, S. Zhang, S. Quan, Z. Ma, Z. Qian, and S. Lu, "Vcmaker: Content-aware configuration adaptation for video streaming and analysis in live augmented reality," *Computer Networks*, vol. 200, p. 108513, 2021.

[27] N. Chen, S. Quan, S. Zhang, Z. Qian, Y. Jin, J. Wu, W. Li, and S. Lu, "Cuttlefish: Neural configuration adaptation for video analysis in live augmented reality," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 830–841, 2020.

[28] W.-J. Kim and C.-H. Youn, "Lightweight online profiling-based configuration adaptation for video analytics system in edge computing," *IEEE Access*, vol. 8, pp. 116 881–116 899, 2020.

[29] F. Bastani, S. He, A. Balasingam, K. Gopalakrishnan, M. Alizadeh, H. Balakrishnan, M. Cafarella, T. Kraska, and S. Madden, "Miris: Fast object track queries in video," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1907–1921.

[30] G. Ananthanarayanan, Y. Shu, M. Kasap, A. Kewalramani, M. Gada, and V. Bahl, "Live video analytics with microsoft rocket for reducing edge compute costs," July 2020. [Online]. Available: https://www.microsoft.com/en-us/research/publication/live-video-analytics-with-microsoft-rocket-for-reducing-edge-compute-costs/

[31] G. Ananthanarayanan, Y. Shu, L. Cox, and V. Bahl, "Project rocket platform—designed for easy, customizable live video analytics—is open source," Microsoft Research Blog, January 2020. [Online]. Available: https://www.microsoft.com/en-us/research/publication/project-rocket-platform-designed-for-easy-customizable-live-video-analytics-is-open-s

[32] M. Xu, X. Zhang, Y. Liu, G. Huang, X. Liu, and F. X. Lin, "Approximate query service on autonomous iot cameras," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 191–205.

[33] 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 (HotCloud 19)*, 2019.

[34] M. Xu, T. Xu, Y. Liu, X. Liu, G. Huang, and F. X. Lin, "A query engine for zero-streaming cameras," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–3.

[35] 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*, 2019, pp. 322–337.

[36] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher, "Scheduling real-time deep learning services as imprecise computations," in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2020, pp. 1–10.