Edge-assisted Adaptive Configuration for Serverless-based Video Analytics

Ziyi Wang*, Songyu Zhang*, Jing Cheng*, Zhixiong Wu*, Zhen Cao†, Yong Cui*
*Department of Computer Science and Technology, Tsinghua University, China
†Computer Network and Protocol Lab, Huawei Technologies, China

2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-3986-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDCS57875.2023.00058

***Abstract*—The growth of video volumes and increased DNN capabilities have led to a growing desire for video analytics, which demands intensive computation resources. Traditional resource provisioning strategies, such as configuring a cluster per peak utilization, lead to low resource efficiency. Serverless computing is a promising way to avoid wasteful resource provisioning since video analytics regularly encounters bursty input workloads and fine-grained video content dynamics. For serverless-based video analytics, the application configuration (frame rate, detection model, and computation resources) will impact several metrics, such as computation cost and analytics accuracy. In this paper, we investigate the joint configuration adjustment problem for video knobs and computation resources provided by the serverless platform. We propose an algorithm that can efficiently adapt configurations for video streams to address two key challenges in serverless-based video analytics systems, including the complex relationships between the configurations and the key performance metrics, and the dynamically best configuration. Our algorithm is developed based on Markov approximation to minimize the computation cost within an accuracy constraint. We have developed a prototype over AWS Lambda and conducted extensive experiments with real-world video streams. The results show that our algorithm can greatly reduce the computation cost under the constraint of target accuracy.**

***Index Terms*—Video analytics, Edge computing, Serverless computing, Deep neural network**

I. Introduction

The deployment of cameras has skyrocketed in recent years. For instance, it is predicted that the global market for surveillance cameras would reach US$ 45.93 billion by 2027 [1], [2]. To fully unleash the potential of these installed cameras, video analytics applications have been developed to assist various public and private institutions (such as law enforcement agencies and retail businesses) to increase efficiency, decrease expenses, and improve security [3], [4], [5]. High-performance video analytics systems are required due to the ever-increasing deployment scale of cameras and customers' growing demand for video analytics applications.

Deep neural networks (DNNs) have transformed the accuracy of video analytics with increased resource requirements [6], [7]. The existing solutions either rely on the resource-rich cloud to build virtual machine clusters or invest money on powerful hardware to build private clusters in order to handle the resource challenge of high-accuracy video analytics at scale [4], [8]. Consequently, the execution of expensive DNNs can result in very high computation costs. For instance, computing analytics results for a year of video footage (60 FPS) using a 3FPS DNN would cost approximately US$ 54K on a 2-core Google Cloud Platform (GCP) instance with an NVIDIA T4 [9]. Despite the fact that cameras can continuously provide video streams, providing a dedicated cluster to analyze them is unnecessary because the ideal configuration and resource needs for a video analytics pipeline change with time. As a result, typical resource provisioning techniques like configuration based on peak usage or one-time offline profiling result in low resource efficiency.

Given that video analytics frequently experiences bursty input workloads and fine-grained video content dynamics, serverless computing is a promising solution to prevent needless resource provisioning [10], [11], [12]. Serverless computing has evolved as a general-purpose computing abstraction to reduce time-consuming administration tasks and offer fine-grained autoscaling computing infrastructures. Function as a Service (FaaS) solutions, which reflect the idea of serverless computing, have achieved commercial success in recent years (e.g., AWS Lambda [13] and Google Cloud Functions [14]). In FaaS platforms, monolithic application codes are separated into a number of functions. A microservice is implemented by each function, which can be individually configured and invoked. The lightweight virtualization approach allows function instances to scale up or down instantly in milliseconds depending on their input workloads, enabling quick and flexible responses. This feature makes it possible for serverless functions to achieve fine-grained scalability and flexibility by allowing them to handle fine-grained input workload variations without the necessity for mandatory resource scaling. Additionally, the pay-as-you-go pricing model of FaaS ensures that no money is squandered on underutilized resources, leading to excellent cost-efficiency.

In serverless-based video analytics, frames are extracted from the video at different sampling rates, compressed into various resolutions, and then analyzed by different DNN models on the serverless computing platform. Users also need to determine the computation resources allocated to the corresponding serverless function. We refer to a particular combination of frame rate, DNN model, and computation resource as a configuration. Evidently, different configurations result in various degrees of accuracy and computation cost. Our measurement study demonstrates that finding the optimal configuration is critical to achieving high-accuracy and cost-

This work was supported by NSFC Project under Grant 62132009 and Grant 62221003. *(Corresponding author: Yong Cui.)*

2575-8411/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDCS57875.2023.00058

effective video analytics. By dynamically selecting appropriate configurations based on changes in video content, significant benefits can be obtained.

The decisions made regarding the configurations of the video knobs and computation resources have an impact on each other and are crucial to the overall effectiveness of video analytics. While there is a broad range of existing efforts from both industry and academia on optimizing configurations for video knobs [4], [8], [15], [16], [17], the collaborative configuration tuning for video knobs and computation resources offered by the serverless platform remains an unexplored area.

In this paper, we propose a framework where the edge server selects key frames and sends them to the serverless platform for object detection while using the remaining frames for local object tracking. Different DNN models are deployed on the serverless platform to match various resolutions. Compared to large DNN models, smaller models with fewer convolutional layers are less accurate but cheaper and faster. The objective of the problem is to decide the frame rate, DNN model, and serverless platform's computation resources for the video stream to minimize the overall computation cost, subject to a service accuracy constraint. This problem is challenging for the following reasons:

(1) The relationships between the configurations and the key performance metrics are intricate. Key metrics are jointly affected by various configurations. And one configuration is responsible for several metrics. For instance, a higher frame rate increases accuracy but also increases computation cost. Accuracy is affected by frame rate, memory size, etc. Moreover, their relationships are non-linear. To figure out the complicated relationships, we need extensive testing and appropriate function fitting. Only by doing this, it is possible to select the optimal set of configurations that minimizes the computation cost while satisfying the accuracy criteria.

(2) The optimal configuration varies over time. To maintain high accuracy, we may employ the most expensive configuration all the time, but this approach will eat up more computation resources. Accuracy and computation cost seems to be a conflict duel. However, the video content information can get us out of the dilemma. In many cases, some policies that lower the resolution and frame rate can cut computation expenses without sacrificing accuracy. For instance, when the target moves slowly, we can use a lower frame rate. And when the target object is large in the frame, we can reduce frame resolution, which would not harm accuracy. Therefore, the ideal configuration is content-related and changes over time. However, how to utilize content information to optimize the computation resources and accuracy is challenging.

These challenges motivate us to propose an adaptive configuration adjustment algorithm for video analytics, which is capable of optimizing the trade-off between accuracy and computation costs. Our solution aims to find the most suitable video knob configuration and computation resource allocation strategy for an edge-assisted video analytics system.

To the best of our knowledge, this is the first work to jointly tune configurations for video knobs and computation resources provided by the serverless platform in the edge environment, explicitly taking into account the trade-off between the analytics accuracy and computation costs. The main contributions of this paper are summarized as follows.

(1) We propose an edge-assisted serverless framework for video analytics. The fine-grained and automatic resource management provided by serverless computing helps our framework dramatically improve resource efficiency.

(2) We formalize the joint video knobs and computation resources configuration problem, for optimizing the trade-off between accuracy and computation costs. We then develop a novel algorithm that efficiently adapts configurations for video streams. It utilizes the Markov approximation to minimize the computation cost under an accuracy constraint.

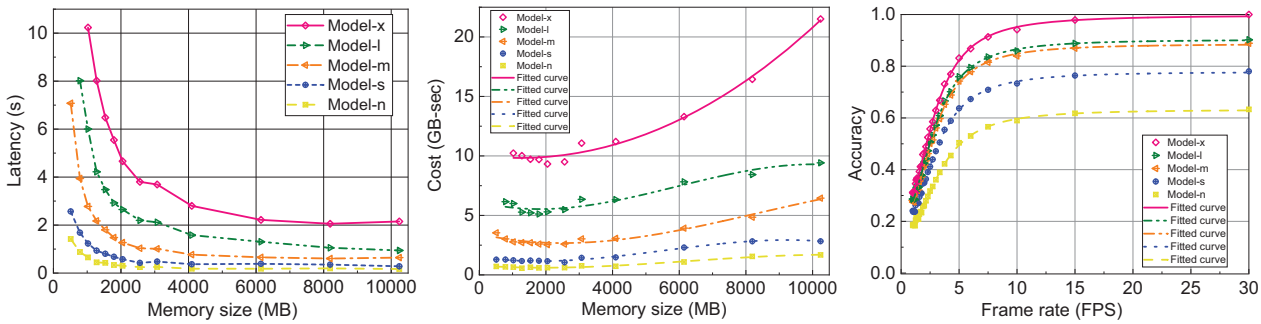
(3) We implement a prototype of a joint configuration optimization framework for the AWS Lambda platform and evaluate the design through extensive and practical experiments with accuracy and cost profiles obtained from our experiments. Results confirm the superiority of our approach compared to several baselines.

The rest of the paper is structured as follows. Section II explains the background and motivation. The framework is shown in Section III. Section IV presents the formulation and model. Section V elaborates the details of our algorithm. Section VI describes the implementation details. Experimental results are presented in Section VII. Section VIII discusses the related work. Finally, we conclude the paper in Section IX.

II. Background and motivation

The commercial success of cloud computing has been shown during the past twenty years. Recently, cloud service providers have proposed serverless computing and used this paradigm in their FaaS offerings to simplify cloud programming and make cloud resources easier to utilize [13], [14]. Developers just need to register functions with the serverless platform and specify events that will cause the functions to run at deployment time. The serverless platform is in charge of processing requests, scaling resources, and guaranteeing fault tolerance. Building high-accuracy and cost-efficient video analytics applications is an ideal fit for the fine-grained and highly parallel computing modes offered by serverless computing.

In this paper, we focus on the configuration adjustment problem for serverless-powered video analytics applications and take the typical object detection task as a case study. Generally, given an input video, three types of configurations can be adjusted: memory size, detection model, and video frame rate. We then discuss how these critical configurations impact performance and cost. Since the camera and the edge server are purchased upfront, their costs amortized per frame will approach zero in the long run [18], but the computation cost on the serverless platform is paid by time. In addition, there is no charge for transferring data from the Internet to the serverless platform [19], [20]. Therefore, we only discuss the computation cost on the serverless platform in this paper without considering the transmission cost.



(a) Latency vs. memory size. (b) Cost vs. memory size. (c) Accuracy vs. frame rate.

Fig. 1. Impact of configurations on the detection latency, cost and accuracy.

(1) Memory size: In current serverless platforms, memory size is the only computation resource knob controlled by users [21], [22]. When memory size is configured, other resources (such as CPU power) are determined proportionally. To explore the impact of memory size on latency and cost, we deploy 5 object detection functions on the serverless platform. Each function contains a YOLOv5 model [23] with a particular size. The results are shown in Fig. 1(a) and 1(b). We can find that the latency decreases as memory size increases because of the increase of computation resources (memory size and thus CPU power). However, the cost first decreases and then increases. Since the serverless platform charges for the total amount of gigabyte-seconds (memory $\times$ latency) consumed by a function, the relationship between cost and memory can be complex, depending on the relationship between latency and memory size. In our case, the latency decreases fast when the memory size is relatively small, corresponding to the decrease in cost, and then decreases slowly as the memory size grows, corresponding to the increase in cost. To sum up, the memory size has a significant impact on cost. We can find an optimal memory size that reduces the cost greatly. In our case, by selecting optimal memory size, we can save up to 62.89% cost compared to the maximum cost for a certain model.

(2) Detection model: As mentioned above, we deploy 5 different YOLOv5 detection models (x is the largest and l, m, s stand for large, medium and small while n is the smallest model) [23]. The accuracy, latency and cost of different models vary a lot. As shown in Fig. 1, a larger model has better performance (accuracy) but higher latency and thus higher cost. For a given target accuracy, we don't necessarily choose the largest model. Instead, in order to save cost, we can choose the smallest model which satisfies the target accuracy. Under the constraint of accuracy, the impact of model configurations on cost is very significant. For example, if the target accuracy is 0.8, we have 3 options of models. By choosing the smallest model, we can save up to 72.75% cost compared to that of the largest model.

(3) Video frame rate: The relationship between accuracy and frame rate is shown in Fig. 1(c). The accuracy of each frame is computed by comparing the detected or tracked objects with the objects detected by the largest model. Therefore, a larger frame rate means a larger proportion of detections and thus higher accuracy. On the other hand, the cost can vary a lot as the frame rate changes. For a given target accuracy, we expect to choose the smallest frame rate which satisfies the target since the cost is proportional to the number of frames processed by the serverless platform. For example, when the target accuracy is given as 0.8 and the model is chosen as the largest model, by selecting the optimal frame rate, we can save 16.67% cost compared to that of the highest frame rate.

The difference in cost between good and bad configurations can be even bigger when taking all three factors into account. In our case, when the target accuracy is set to 0.8, we can save costs up to 89.37% compared with the worst configuration. Therefore, a huge reduction in cost can be obtained by choosing a proper configuration of memory size, model and frame rate.

III. Edge-assisted Serverless Framework

In this section, an overview of our framework is given first. Then we will elaborate on two key components of the framework, including edge server and serverless platform.

A. Framework Overview

As shown in Fig. 2, the camera connects to the edge server and continually streams its captured videos to it. Video analytics tasks are carried out by means of deep neural network models, which require a relatively large amount of computation and memory usage. Due to the limited computation power of edge servers, it is difficult to support such large-scale inference computation of deep neural network models entirely on edge servers. However, transmitting all the videos to the cloud platform for analytics will increase unnecessary computation and transmission costs, because there are a large number of redundant frames in the video that do not require repeated transmission and computation (object detection). Therefore, this paper proposes a cloud-edge collaborative video analytics framework. The edge server is responsible for performing video buffering and lightweight object tracking, while the cloud platform runs a DNN-based object detection model only on some key frames to reduce computation cost. In this paper, the partial frames sent to the serverless cloud

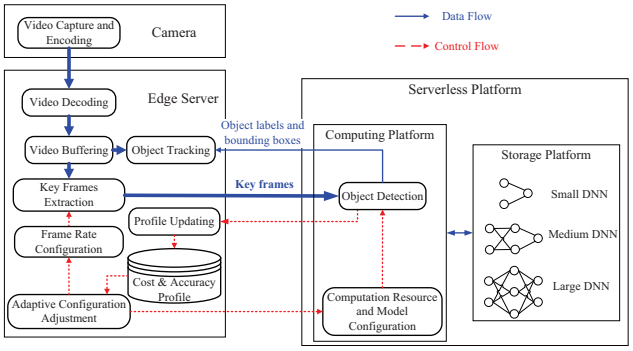


Fig. 2. Edge-assisted serverless framework for video analytics.

platform to perform object detection are called key frames. The fine-grained and automatic resource management provided by serverless computing helps our framework dramatically improve resource efficiency.

B. Edge Server

After the edge server obtains the video stream transmitted from the camera, the video decoding module decodes the video stream into continuous video frames. According to the object detection result message corresponding to the key frame from the serverless cloud platform, the video buffering module buffers the key frame and its analytics result. The lightweight object tracking module will use the detection result of the key frame as the target to be tracked, and execute the continuous object tracking algorithm to obtain the analytics result of the non-key frame. Specifically, we use the Lucas-Kanade object tracking algorithm [24], which maps the location of the object from one frame to the next in two steps: (1) extract feature points representing the moving object, and (2) estimate where those feature points could be in the second frame to locate the object. Using object tracking can reduce the number of video frames sent to the serverless platform to perform object detection tasks, thereby reducing the amount of computation and memory usage while ensuring that the accuracy of video analytics does not drop excessively.

A key module on the edge server is the adaptive configuration adjustment algorithm. The relationships between the solution spaces and the optimization objectives are obtained from the cost profile and accuracy profile, which are used as the inputs of the algorithm. Its outputs are the configurations of video frame rate, detection model, and computation resources. The frame rate configuration is directly used locally on the edge server to guide the key frames extraction module. The configurations of the detection model and computation resources are sent to the serverless platform through the control flow to guide the object detection module.

C. Serverless Platform

The serverless cloud platform is used to perform deep neural network inference computation tasks. In addition to receiving key frames from the edge server, it also receives the computation resources and model configurations generated by the adaptive configuration adjustment module. Then it performs object detection inference on key frames according to these configurations. The model selected in this paper is YOLOv5 [23], the latest work of the single-stage object detection model. In order to adapt to different needs, YOLOv5 also provides a variety of models from small to large. Their structures are similar, but the network depth and width are different, and the corresponding parameters and computations are also different. A larger model can achieve better detection results, but at the same time, the corresponding computation cost and memory usage will be larger. We deploy these models on the serverless platform, and the algorithm can select the corresponding model according to the needs of the scenario. After getting the results of object detection, including the object labels and bounding boxes in the key frame, the serverless platform feeds these results back to the edge server for subsequent object tracking and profile updating.

In fact, given that our framework has a clear distinction between control and data flow, it is trivial to use this framework in many video analytics application scenarios. In addition to obtaining the identity of the target object and the corresponding confidence score, the framework can also obtain other information by replacing different DNN models on the serverless platform according to the needs of the application, such as semantic segmentation and object reidentification.

IV. Formulation and Model

As illustrated in Fig. 2, there are N parallel DNN models $d_1, d_2, ..., d_N$ deployed on the serverless platform, with different input sizes of images. It has been well studied that the DNN model can be compressed to a smaller size at the expense of accuracy [25]. Such techniques include removing some expensive convolutional layers and reducing input image resolution. Thus in our design, the DNN model with lower input image resolution has a faster processing speed and needs fewer resources. Different pre-trained DNN models have different input image resolutions which have already been defined in their model architectures. That is, when we determine the model selection, its input image resolution is fixed. This is consistent with previous work [15].

We divide time into discrete time slots, each of which has a duration that matches the timescale at which configurations can be updated. We introduce a binary variable x_t^i to indicate whether model d_i is selected in time slot t. Then we use x_t to denote $\{x_t^i | \forall d_i \in \{d_1, d_2, ..., d_N\}\}$, which is the collection of model selection variables. Similarly, we use f_t to represent frame rate selection for the video stream in time slot t, and m_t is the memory allocation value. To facilitate our presentation, the major notations are summarized in Table I. In the remainder of this section, we first provide analytical models on the computation cost and accuracy. Then, we present the problem formulation.

TABLE I
MAJOR NOTATIONS USED IN THIS PAPER.

Notation	Description
d_i	The $i-th$ DNN model deployed on the serverless platform
m_t	Memory configuration in time slot t
f_t	Frame rate configuration in time slot t
x_t^i	Whether model d_i is selected in time slot t (binary variable)
x_t	Model selection configuration vector in time slot t
$F_C^t(m_t)$	Computation cost of the memory configuration m_t
$F_A^t(f_t)$	Analytics accuracy of the frame rate configuration f_t
A	Target analytics accuracy
η	Transition probability in Markov approximation
τ	Smooth parameter used to control exploration vs. exploitation
T_{max}	The maximum number of iterations of the algorithm

A. Computation Cost

As mentioned before, cameras and edge servers are generally purchased and deployed in advance, so their costs tend to be zero in the long run [18]. However, the computation cost of the serverless platform is paid according to the usage. In addition, in our framework, the end-to-end computation bottleneck (DNN model) is on the serverless platform, and the computation load on the edge side is relatively small. Therefore, we mainly consider the computation cost of the serverless platform in the modeling process.

In current serverless platforms, memory size is the only computation resource configuration controlled by users [21], [22]. The allocation of CPU capacity for the function is done proportionally to the amount of memory allocated. Thus, increased memory allocation also means increased CPU allocation, which can improve the performance of a processing-intensive serverless function. Therefore, we only consider memory size as the resource configuration in this work.

Each serverless function can be "sized" by setting the maximum memory size (GB) parameter in the Console or using the API. This value also affects the CPU shares allocated to the function when it runs, but in a manner that is not currently disclosed by the serverless platform. The platform also allows limiting the maximum function execution time (seconds) for a function, to prevent runaway or hanging functions from driving up cost. Since serverless functions run only when a request must be serviced, they only incur charges when used. The general pricing model adopted by serverless FaaS providers is based on two numbers per function invocation [19]:

(1) Maximum memory size (GB): this is not the actual memory used by the function, but rather the maximum memory size parameter in the serverless function's configuration.

(2) Function execution time (seconds): the actual amount of clock time that the function invocation takes to run. Execution time is measured in 100 ms blocks and is always rounded up to the next full block.

For each function invocation, these two values are multiplied together to produce a number with the unit GB-sec. After allowing for a monthly allowance of free GB-sec from the free tier, the billable computation cost is the total GB-sec across all function invocations, multiplied by a fixed GB-sec rate.

Optimizing costs for a serverless function involves balancing memory allocation with execution time. A key ingredient for this optimization is getting insights into memory and execution time for the serverless function. Allocating additional memory to a serverless function may improve performance in some cases, but in other cases, the performance of the serverless function may depend on external factors — in which case, allocating additional memory increases the cost of an invocation without improving its performance. Knowing what factors affect the performance of the serverless function is crucial in the optimization process.

To do so, we implement an object detection serverless function with different DNN models on the AWS platform [13] to perform object detection on a clip from a traffic video. We vary the value of memory allocated to the function and measure its execution time and computation cost. We also replace different object detection models for the measurement. It can be seen from the Fig. 1 that the computation cost will first decrease and then increase with the increase of the allocated memory value. The reason is that with the increase of the allocated memory value and the corresponding increase in computation power (CPU resources), the execution time of the function and the computation costs are reduced. However, as the memory value further increases, the advantage of decreasing execution time has become smaller. At this time, the cost of increasing the memory value becomes the main part, and the computation cost increases.

We can find that the cost-memory functions fitted by different DNN models are similar, but the parameters are different. The relationship between computation cost ($F_C^t(m_t)$) and the memory value (m_t) can be fitted as cubic function:

$$F_C^t(m_t) = \sum_{i=1}^{N} x_t^i \cdot (c_{i1} m_t^3 + c_{i2} m_t^2 + c_{i3} m_t + c_{i4}), \quad (1)$$

where c_{i1}, c_{i2}, c_{i3} and c_{i4} are constant coefficients.

B. Analytics Accuracy

The accuracy models are derived based on the performance measurements obtained from our real experiments. We use YOLOv5 [23], an object detector DNN to perform object detection on a clip from a traffic video with different frame rates. For those undetected video frames, we use the Lucas-Kanade tracking algorithm to track the objects with high fidelity [24], [26], [27]. Since the video segment consists of many frames, we compute accuracy by comparing the detected and tracked objects using different frame rates with

the objects fully detected using the highest frame rate. As for the metric, we use the F1 score, which is the harmonic mean of precision and recall. A detected object is identified as true positive when its bounding box has the same label and sufficient spatial overlap with the corresponding ground truth. The spatial overlap can be measured by IoU (Intersection over Union). In our experiment, an object is correctly detected when IoU $\geq$ 0.5 [26].

The relationship between accuracy and the sampling frame rate is illustrated in Fig. 1. We can observe that a higher frame rate produces better analytics accuracy. The accuracy-framerate functions fitted by different DNN models are similar, but the parameters are different. The relationship between accuracy ($F_A^t(f_t)$) and the frame rate (f_t) can be fitted as exponential functions:

$$F_A^t(f_t) = \sum_{i=1}^{N} x_t^i \cdot \left(\frac{w_{i1}}{1 + (f_t/w_{i2})^{w_{i3}}} + w_{i4}\right), \tag{2}$$

where w_{i1}, w_{i2}, w_{i3} and w_{i4} are constant coefficients.

C. Problem Formulation

We only consider the computation cost on the serverless platform without accounting for the transmission cost as there is no charge for transferring data from the Internet to the serverless platform [19], [20]. Analytics on video streams needs low computation cost and high accuracy. As a result, when designing the adaptive algorithm, we aim at minimizing computation cost under the analytics accuracy constraint. The problem can be formulated as:

$$\begin{aligned} &\min_{m_t, f_t, x_t} F_C^t(m_t). \\ s.t. &\sum_{i=1}^{N} x_t^i = 1. \\ &x_t^i = \{0, 1\}. \\ &F_A^t(f_t) \geq A. \end{aligned} \tag{3}$$

Constraints 1 and 2 ensure that, in each time slot, one and only one DNN model can be selected. Constraint 3 says that the accuracy should be higher than the target value A.

V. Algorithm Design

The formulated problem is a mixed integer nonlinear programming and it is impossible to find an optimal solution in polynomial time. Consequently, we propose to leverage Markov approximation to obtain a solution [28], [29]. Supposed that model selection x_t is fixed, there are two problems left to be solved:

The first problem is optimizing memory allocation to reduce computation cost:

$$\min_{m_t} F_C^t(m_t). \tag{4}$$

It can be proved that the optimal memory allocation can be derived as follows:

$$m_t^* = \sum_{i=1}^{N} x_t^i \cdot \frac{\sqrt{c_{i2}^2 - 3c_{i1}c_{i3}} - c_{i2}}{3c_{i1}}. \tag{5}$$

Algorithm 1 Adaptive Configuration Adjustment Algorithm

Input: $A : accuracy\ threshold$,
$F_C^t(m_t) : computation\ cost\ function$,
$F_A^t(f_t) : accuracy\ function$,
$x_t : initial\ model\ selection\ vector$
Output: $m_t : memory\ configuration$,
$f_t : frame\ rate\ configuration$,
$x_t : model\ selection\ configuration$
1: $Profile\ computation\ cost\ function\ F_C^t(m_t)$
2: $Profile\ accuracy\ function\ F_A^t(f_t)$
3: **repeat**
4: $Randomly\ change\ the\ model\ selection\ vector\ x_t$ $into\ \hat{x_t}\ by\ selecting\ a\ new\ model$
5: $Obtain\ \hat{m_t^*}\ using\ Eq.\ (5)$
6: $Obtain\ \hat{f_t^*}\ using\ Eq.\ (7)$
7: $\eta \leftarrow \frac{1}{1+e^{\frac{\hat{C}-C}{\tau}}}$
8: $With\ probability\ \eta,\ algorithm\ accepts$ $the\ new\ model,\ m_t \leftarrow \hat{m_t^*}, f_t \leftarrow \hat{f_t^*}$
9: $With\ probability\ (1-\eta),$ $algorithm\ keeps\ the\ model\ unchanged$
10: **until** $there\ is\ no\ significant\ reduction\ in\ the$ $computation\ cost\ for\ more\ than\ T_{max}\ iterations$
11: **return** x_t, m_t, f_t

The second problem is adapting frame rates to satisfy the target accuracy:

$$F_A^t(f_t) = A. \tag{6}$$

We can get the critical frame rate that just meets the accuracy requirement:

$$f_t^* = \sum_{i=1}^{N} x_t^i \cdot w_{i2} \cdot \left(\frac{w_{i1}}{A - w_{i4}} - 1\right)^{\frac{1}{w_{i3}}}. \tag{7}$$

Based on the analysis above, we can come to the conclusion that once optimal model selection x_t is found, the two left problems are both easy to solve. However, since model selection variables are binary, the whole problem is a mix-integer nonlinear problem. It is impossible to find an optimal solution in polynomial time. In this paper, we propose to leverage Markov approximation to obtain a solution for model selection, as shown in Algorithm 1. We divide time into discrete time slots. At the beginning of each time slot, the computation cost and accuracy models are updated by the cost and accuracy profiler according to the current video content (line 1 to 2).

Given the cost and accuracy functions, we can find the optimal video knobs and memory configurations for the current slot by solving the above problems. Firstly, we randomly choose a new DNN model $\hat{d}$, then the new model selection vector $\hat{x_t}$ is obtained, under which the optimal $\hat{m_t}$ and $\hat{f_t}$ can be derived by solving the above problems (line 4 to 6). Afterwards, the new computation cost objective value $\hat{C}$ is calculated, and C is known as the objective function value for the old solution $\{m_t, f_t, x_t\}$. In the current iteration, the

model selected is updated to $\hat{d}$ with probability η and keeps unchanged with probability $1-\eta$ depending on the objective value difference $\hat{C}-C$ (line 7 to 9). Therefore, changing DNN model selection is more likely to occur if the new configuration $\{\hat{m_t}, \hat{f_t}, \hat{x_t}\}$ results in a lower objective value. The above iterative processes will continue until T_{max} iterations have been reached.

The parameter τ in line 7 is used to control exploitation versus exploration. When τ approaches infinity, the algorithm tries to explore all possible solutions from time to time without convergence. When τ is small, the algorithm takes more iterations to identify the globally optimal solution since the algorithm may be stuck in a locally optimal solution for a long time before exploring other alternatives. The selection of τ will be discussed in the evaluation section.

VI. Implementation

We prototype our system with a public cloud provider, AWS, and utilize AWS's serverless platform (AWS Lambda [13]) for our cloud-side implementation. Cloud-side part is designed to detect objects on incoming frames and return bounding box results. In particular, we deploy several serverless functions to carry out that task. And YOLOv5 models [23] are chosen for the object detection task. For DNN models having optimal memory configurations separately, we deploy DNN models in different serverless functions. Since AWS Lambda does not support runtime memory adjusting on invocation of serverless functions, we deploy functions in various memory sizes for every model in advance. Codes and running dependencies are packed. Additionally, the pre-trained DNN model weights are also packed into zip files, eliminating the need to download them from cloud object storage services like Amazon S3. Locally packed files are uploaded to Amazon ECS and then used to deploy serverless functions. We share code bases across serverless functions, and memory can be configured differently. Finally, to access serverless functions outside of the AWS platform, we configure API Gateway, which invokes function whenever a legit HTTP request comes. Video frames carried in HTTP POST requests are processed and bounding box results are sent back in the HTTP response.

The edge server is equipped with an octa-core Intel processor running 2.2 GHz with 32 GB of RAM. We build the edge server part with Python language. Once receiving videos, FFmpeg extracts frames, which are indexed and stored in the buffer. Our algorithm determines the model, memory, and frame rate at the beginning of each time slot utilizing profiles from the previous time slot. These solutions are cost-optimal while meeting the target accuracy requirement. Following the result of our algorithm, our system will send frames at the specified frame rate to the serverless function that matches the model and memory choice. In addition, the tracking module is designed to update tracking status using incoming frames and results from AWS Lambda. Specifically, for each bounding box result, we create a tracker and save it in a dictionary. Existing trackers will be sequentially updated utilizing frames that are not sent to the serverless function. When new detection results are returned from AWS Lambda, a new dictionary containing trackers is created and the old dictionary is replaced. Together, the two parts accomplish our design goal, which fulfills accuracy while minimizing cost.

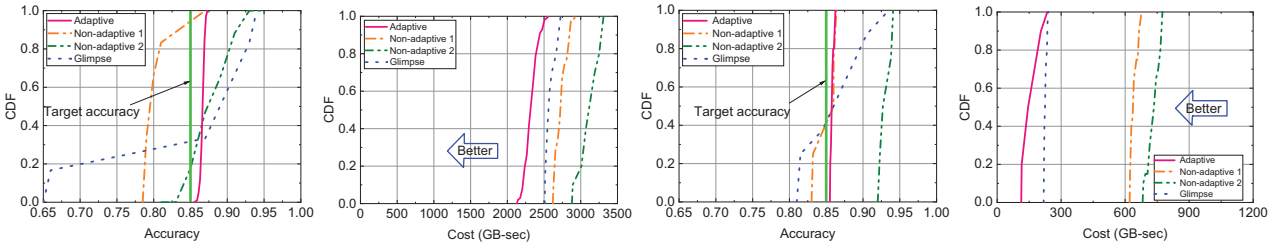
VII. Evaluation

In this section, we evaluate the performance of our algorithm and compare it with several baselines[1]. To set up experiments, we profile inter-relationships between decision domains. First, we profile correlations between cost and memory size. Five YOLOv5 object detection models are deployed on AWS Lambda. For each model, the relationship between the cost and memory during a single detection is profiled. Then, the optimal memory that minimizes the cost is determined based on the profiling result. In subsequent experiments, as long as a model is selected, we will allocate the optimal memory to the model. The cost per detection varies from 0.6 GB-sec to 9.5 GB-sec. Therefore, picking the optimal memory is indispensable for minimizing the cost. Second, we profile the relationship between accuracy and frame rate. For every model, we run the pipeline with a set of fixed framerates and record the resulting accuracies. Thereby, we determine the search space for deciding the models and frame rate under accuracy constraints. From the edge server, we send keyframes to AWS Lambda for detection and implement object tracking based on the returned results. By exploiting the profiling results, our adaptive configuration adjustment algorithm minimizes the cost by deciding the model for detection and the frame rate at which keyframes are sent. To verify the effectiveness, we compare our algorithm with three other baselines:

- *Non-adaptive1*: This baseline uses a fixed configuration. YOLOv5x is chosen as the detection model, and the frame rate is set to 5 FPS. This baseline uses the largest model to guarantee accuracy but a low frame rate to lower the cost.
- *Non-adaptive2*: This baseline also uses a fixed configuration. The YOLOv5l detection model is chosen, and the frame rate is set to 10 FPS. To compensate for the accuracy losses caused by picking a smaller model, a higher frame rate is adopted.
- *Glimpse* [26]: Glimpse client sends selected frames to the Glimpse server for object detection and runs tracking on unselected frames. The selection is based on measuring the pixel difference between neighbor frames and tracking objects using optical flow. For a fair comparison, we change Glimpse's implementation from a static server to AWS Lambda. To unify the performance metrics, the GB-sec cost is selected for computation power consumption.

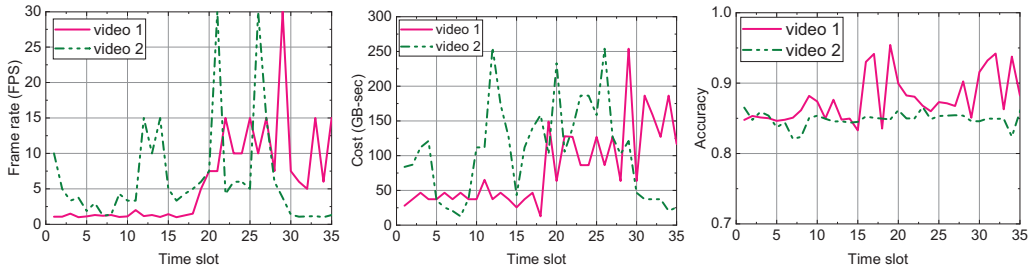
In our evaluation, the F1 score is the metric for accuracy, which computes the harmonic mean of precision and recall for the detected objects' locations and class labels. The higher the F1 score, the better the accuracy performance. The computation cost is measured by multiplying memory and computation

[1]https://github.com/STAR-Tsinghua/ServerlessVideoAnalytics



(a) Inference accuracy performance on traffic dataset. (b) Computation cost on traffic dataset. (c) Inference accuracy performance on drone dataset. (d) Computation cost on drone dataset.

Fig. 3. Inference accuracy and computation cost of our algorithm (Adaptive) vs. several baselines on traffic and drone datasets.



(a) Frame rate over time. (b) Computation cost over time. (c) Inference accuracy over time.

Fig. 4. Runtime behavior of our algorithm over time.

time, which is quantified in GB-sec. For datasets, we use 3 distinctive ones provided by [18]. The traffic dataset contains 7 videos with a total length of 2331s. The dashcam dataset contains 9 videos with a total length of 5361s. The drone one contains 13 videos with a total length of 163s.

A. Algorithm Comparison

In this subsection, we compare our algorithm and the baselines by accuracies and costs on traffic and drone datasets. The target accuracy is set to 0.85 for ours and Glimpse. As shown in Fig. 3(a) and 3(b), Non-adaptive 1 performs badly, which is inferior to ours in terms of cost and accuracy. Non-adaptive 2 outperforms Non-adaptive 1 in accuracy. However, its cost outweighs the competitor's by a large margin. In general, both Non-adaptive strategies' accuracy varies in a wide range, failing to meet target accuracy. For Glimpse, it is more cost-efficient than non-adaptive algorithms, for achieving higher accuracy with less cost. Nevertheless, Glimpse may perform poorly with a non-negligible probability. In comparison, our algorithm achieves target accuracy while costing the least in all measured strategies. In detail, the cost of our algorithm is 14.88% lower than Non-adaptive 1, 25.64% than Non-adaptive 2, and 9.78% than Glimpse.

We apply the same comparison scheme to the drone dataset to verify our algorithm's generalization ability. The accuracy goal for Glimpse and our algorithm is set to 0.85. As shown in Fig. 3(c) and 3(d), Non-adaptive 1's accuracy is relatively near to the target accuracy. Non-adaptive 2's accuracy is much higher than the target accuracy. However, their costs grow non-proportionally higher, which is unbearable compared to the accuracy-cost efficiency achieved by our algorithm. Glimpse's mean accuracy is close to ours. Its cost is lower than Non-adaptive strategies yet falls short of the comparison of ours by a small margin. However, it suffers from an unstable performance. In 40% of tested times, it fails to achieve target accuracy. As a comparison, our strategy assures target accuracy while maintaining the lowest cost among comparing strategies during testing. We can save up to 74.03% cost compared to Non-adaptive 1, 77.34% compared to Non-adaptive 2, and 26.43% compared to Glimpse.

Combining the performances of Non-adaptive strategies across two datasets, we argue that Non-adaptive strategies not only fail to meet target accuracy but also fail to keep relative cost efficiency compared to our algorithm. However, the performances of Non-adaptive strategies are incoherent across datasets. What are the reasons for the inconsistency? By manually examining the two testing datasets, the difference in datasets is revealed. In the traffic dataset, the objects' speed is much faster and the object quantity is much higher, compared to the drone one. In the drone dataset, a smaller model is adequate to achieve target accuracy, and a lower frame rate is also acceptable. By exploiting the two characters, the cost can be cut significantly. By pairing a larger model with a lower frame rate, and a smaller model with a higher frame rate, the initiative of Non-adaptive strategies makes sense. However, fixed configurations fail to adapt to different inputs, resulting in inconsistency across datasets, and failure to meet the accuracy goal. In comparison, our algorithm is the best performer under different testing scenarios, well satisfying the accuracy constraint while significantly reducing cost, by

adaptively adjusting the model, memory, and frame rate based on video content.

B. Detailed Behaviors of Our Algorithm

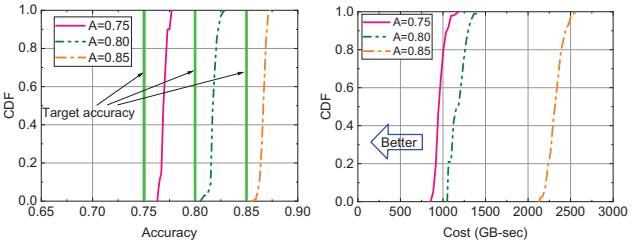
In this subsection, we want to examine the run-time metrics of our algorithm (frame rate, cost, and accuracy) and verify our design goal of adaptively adjusting configuration based on video content. Two videos are picked as testers. By manually checking, the real-time characters of the videos are determined. If there is coherence between the behavior of our algorithm and the characters of the videos, we can verify our design goal. Video 1 is collected from a traffic camera. In the first half, objects move slowly. In the second half, objects move much faster. As shown in Fig. 4, in the first half, the frame rate is low, while the frame rate soars in the second half. We argue that if video content is mostly static, our algorithm will turn down the frame rate to save costs. If video content changes sharply, the algorithm will turn up the frame rate to meet target accuracy. Video 2 is shot by a dash camera. The objects change speed irregularly and constantly. As shown in Fig. 4, the aforementioned pattern persists in the case of video 2. When object speed in video changes, the frame rate changes accordingly. The pattern underpins our claim that the algorithm adaptively adjusts configuration based on video content. As a result, the target accuracy of 0.85 is well met over time slots. The idea behind the adaptation is that "more frames can be handed to do object tracking if the difference between frames is small, so frame rate can be cut and cost saved".

C. The Impact of Different Parameters

In this subsection, we evaluate the impact of different parameters on the run-time behavior of our algorithm.

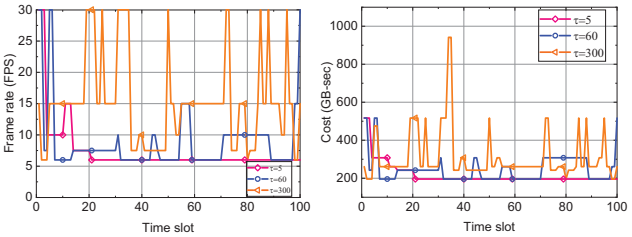
(1) Target accuracy A is a vital constraint in our experiment. In order to explore the performance of our algorithm under different accuracy goals, we set 3 different values: 0.75, 0.8, and 0.85. For each value, our system is tested 100 times. The resulting cost and accuracy statistics are shown in Fig. 5. As displayed in Fig. 5(a), the target constraints are well fulfilled. All trials satisfy the target accuracy. As Fig. 5(b) shows, the more stringent the goal, the higher the cost. However, the ratio of cost and target accuracy grows disproportionately. Because the system will resort to a larger model and higher frame rate together to meet higher requirements, which increases expense dramatically. Moreover, we can observe that the actual accuracy lies between a narrow region above the target goal. Because our goal is to minimize cost, our system is conservative about a higher accuracy for the price of disproportionate cost increase. In short, our system fulfills different accuracy promises while minimizing the cost.

(2) Smooth parameter τ is another important parameter in our experiment, which is used to control exploration versus exploitation in the configuration spaces. In order to explore the influence of τ on convergence, we choose 3 different values: 5, 60, and 300. Fig. 6 shows the fluctuation of cost and frame rate for different τ when the algorithm iterates. Smaller τ contributes to faster convergence and a smaller fluctuation



(a) Inference accuracy under different accuracy constraint A. (b) Computation cost under different accuracy constraint A.

Fig. 5. The impact of accuracy constraint A.

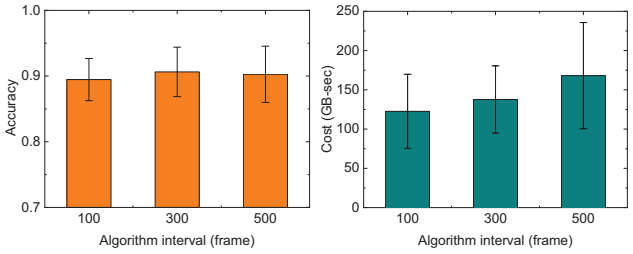


(a) Frame rate under different smooth parameter τ. (b) Computation cost under different smooth parameter τ.

Fig. 6. The impact of smooth parameter τ.

interval. As τ goes up, the interval goes wider and the result is less stable. When τ equals 5, the algorithm converges within 20 time slots. On the contrary, when τ is set to 300, the curve vibrates violently and fails to output a good result. Smaller values of τ have a strong tendency to stick to the optimal cost since they are greedier about lower costs and less sensitive to cost changes. Larger τ tends to explore possible solutions and seek cost changes. However, if τ is too small, the algorithm will stick to the initial selection and fail to change. On the other hand, if τ is too large, the algorithm fails to converge. Therefore, the promising value of τ sits between large and small. In our experiment, we find the most appropriate value for τ is 5, which converges fast and produces a good result. On other datasets, the aforementioned pattern persists and the optimal τ can be determined individually.

(3) The execution interval of the adaptive adjustment algorithm is a critical parameter in our experiments. We choose three sets of values (100, 300, and 500 frames) to examine the performance of our algorithm at various execution intervals. Fig. 7 displays the algorithm's accuracy and computation cost at these intervals. We can find that, on average, the smaller the algorithm execution interval is, the closer the result corresponding to the configuration selected by the algorithm is to our optimization goal, that is, to minimize the computational cost while satisfying the accuracy constraint. Specifically, when the algorithm is executed every 100 frames instead of every 300 or 500, the accuracy achieves the constraint of 0.85 and the computation cost is the lowest. Frequent execution of the algorithm inevitably introduces some computation overhead. As the execution interval of the algorithm



(a) Accuracy at different algorithm intervals.

(b) Computation cost at different algorithm intervals.

Fig. 7. The impact of algorithm interval.

increases, the determined configuration is difficult to adapt to the optimization objective of all video frames in the entire interval, so the computation cost is relatively large.

VIII. Related work

A. Video Analytics

Recent years have witnessed the proliferation of video analytics systems. The key idea is to balance resources and accuracy by adjusting video configuration knobs, such as frame rate, resolution, and DNN models. For example, Jiang *et al.* [4] presented a controller that dynamically picks the optimal video knobs for existing DNN-based video analytics pipelines. Ran *et al.* [30] considered the complex interaction between model accuracy, video quality, battery constraints, network data usage, and network conditions to determine an optimal offloading strategy. Wang *et al.* [15] studied the configuration adaption and bandwidth allocation for multiple video streams, which are connected to the same edge node sharing an upload link. Zhang *et al.* [8] profiled the resource demand and accuracy of different knob combinations for each video query offline, and adjusted configuration knobs for large-scale concurrent queries according to their quality and latency goals online. Zhang *et al.* [16] tuned video frame rate, frame resolution, and quality parameters to save bandwidth and maintain inference accuracy. Xiao *et al.* [31] characterized the complex interaction between video analytics pipelines and video characteristics. They built a performance clarity profile for each pipeline to define its accuracy/cost tradeoff and its relationship with video characteristics. Elgamal *et al.* [32] presented a 3-tier video analytics system to reduce the latency and increase the throughput of NN-based analysis over video streams. They used semantic video encoding in which the video encoder becomes aware of the object detection task. Hung *et al.* [33] proposed a system that identifies the tradeoff between multiple resources and accuracy, and narrows the search space by identifying a "Pareto band" of promising configurations. Jain *et al.* [34] presented a system that leverages a model of cross-camera correlations to reduce the size of the search space, thus reducing the cost of cross-camera analytics. However, these studies only focus on the trade-off between analytics accuracy and computation resources, while ignoring the elastic computation requirements of video analytics and the opportunity to save the overall computation cost by optimizing computation resource allocation provided by the cloud platform.

B. Serverless Computing

Several efforts have been made to unlock the potential of serverless computing in video processing. Fouladi *et al.* [35] provided a framework for interactive video processing applications by invoking thousands of serverless function instances in seconds. Ao *et al.* [36] provided a framework to orchestrate serverless functions in video processing pipelines and exploit the intra-video parallelism to achieve low latency. Zhang *et al.* [1] presented a cloud-edge collaborative serverless video analytics system and solved a cloud-edge partitioning problem for multiple concurrent serverless pipelines. Romero *et al.* [37] presented a heterogeneous and serverless video analytics and processing framework that executes general video pipelines. Jang *et al.* [38] proposed a serverless microservice architecture that operates on top of a fleet of smart cameras for multi-tenant, multi-application real-time video analytics use cases. Kang *et al.* [9] built an end-to-end system for interactive video analytics which manages its own serverless workers across heterogeneous accelerators, and leverages optimizations for jointly optimizing pre-processing and inference. Yu *et al.* [39] presented a serverless-based model serving system that automatically partitions a large DNN model across multiple functions for faster inference and reduced per-function memory footprint. However, these studies are optimized for the framework that combines serverless computing and video processing tasks, without considering the important impact of configuration tuning on application performance.

The most related work is probably [21], in which the authors presented an automated configuration tuning tool for serverless video processing pipelines. However, they only considered serverless-side configuration knobs for each worker: the allocated resource and the assigned workload. We jointly tune configurations for video knobs (e.g., video frame rate and detection model) and computation resources provided by the serverless platform, explicitly taking into account the trade-off between the analytics accuracy and computation costs.

IX. Conclusion

Video analytics has been a driving application of networking research. In this paper, we propose an edge-assisted serverless framework for video analytics. Based on this, we study the joint configuration tuning problem for video knobs and computation resources provided by the serverless platform. We propose an efficient algorithm based on Markov approximation which can select appropriate configurations for video streams, while accounting for the trade-off between accuracy and cost. A prototype with AWS Lambda is further implemented for evaluation. Our extensive experiments with real-world video streams showed the effectiveness and superiority of our algorithm over state-of-art solutions.

REFERENCES

[1] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang, "Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines," in *Proceedings of the 12th ACM Multimedia Systems Conference (MMSys)*, 2021, pp. 80–93.

[2] M. Zhang, F. Wang, and J. Liu, "Casva: Configuration-adaptive streaming for live video analytics," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 2168–2177.

[3] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *IEEE Computer*, vol. 50, no. 10, pp. 58–67, 2017.

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

[5] Q. Zhang, H. Sun, X. Wu, and H. Zhong, "Edge video analytics for public safety: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1675–1696, 2019.

[6] Z. Chen, K. Fan, S. Wang, L.-Y. Duan, W. Lin, and A. Kot, "Lossy intermediate deep learning feature compression and evaluation," in *Proceedings of the 27th ACM International Conference on Multimedia (MM)*, 2019, pp. 2414–2422.

[7] Z. Fang, D. Hong, and R. K. Gupta, "Serving deep neural networks at the cloud edge for vision applications on mobile platforms," in *Proceedings of the 10th ACM Multimedia Systems Conference (MMSys)*, 2019, pp. 36–47.

[8] 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)*, 2017, pp. 377–392.

[9] D. Kang, F. Romero, P. Bailis, C. Kozyrakis, and M. Zaharia, "VIVA: An end-to-end system for interactive video analytics," in *Proceedings of the 12th Conference on Innovative Data Systems Research (CIDR)*, 2022.

[10] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020, pp. 419–434.

[11] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "SAND: Towards high-performance serverless computing," in *2018 USENIX Annual Technical Conference (ATC)*, 2018, pp. 923–935.

[12] Z. Jia and E. Witchel, "Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021, pp. 152–166.

[13] "AWS Lambda," https://aws.amazon.com/lambda/, Accessed: 2023-05-01.

[14] "Google Cloud Functions," https://cloud.google.com/functions, Accessed: 2023-05-01.

[15] 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.

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

[17] M. Hanyao, Y. Jin, Z. Qian, S. Zhang, and S. Lu, "Edge-assisted online on-device object detection for real-time video analytics," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

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

[19] "AWS Lambda Pricing," https://aws.amazon.com/lambda/pricing/, Accessed: 2023-05-01.

[20] I. Stoica and S. Shenker, "From cloud computing to sky computing," in *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, 2021, pp. 26–32.

[21] M. Zhang, Y. Zhu, J. Liu, F. Wang, and F. Wang, "Charmseeker: Automated pipeline configuration for serverless video processing," *IEEE/ACM Transactions on Networking*, 2022.

[22] Z. Wen, Y. Wang, and F. Liu, "Stepconf: Slo-aware dynamic resource configuration for serverless function workflows," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1868–1877.

[23] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, and et al., "ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference," Feb. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6222936

[24] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI)*, 1981, pp. 674–679.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015, pp. 1–14.

[26] 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 the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2015, pp. 155–168.

[27] K. Apicharttrisorn, X. Ran, J. Chen, S. V. Krishnamurthy, and A. K. Roy-Chowdhury, "Frugal following: Power thrifty object detection and tracking for mobile augmented reality," in *Proceedings of the 17th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2019, pp. 96–109.

[28] M. Chen, S. C. Liew, Z. Shao, and C. Kai, "Markov approximation for combinatorial network optimization," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6301–6327, 2013.

[29] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 36, no. 10, pp. 2333–2345, 2018.

[30] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[31] Z. Xiao, Z. Xia, H. Zheng, B. Y. Zhao, and J. Jiang, "Towards performance clarity of edge video analytics," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2021, pp. 148–164.

[32] T. Elgamal, S. Shi, V. Gupta, R. Jana, and K. Nahrstedt, "Sieve: Semantically encoded video analytics on edge and cloud," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 1383–1388.

[33] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "VideoEdge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[34] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, P. Bahl, and J. Gonzalez, "Spatula: Efficient cross-camera video analytics on large camera networks," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 110–124.

[35] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 363–376.

[36] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A serverless video processing framework," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2018, pp. 263–274.

[37] F. Romero, M. Zhao, N. J. Yadwadkar, and C. Kozyrakis, "Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2021, pp. 1–17.

[38] S. Y. Jang, B. Kostadinov, and D. Lee, "Microservice-based edge device architecture for video analytics," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE Computer Society, 2021, pp. 165–177.

[39] M. Yu, Z. Jiang, H. C. Ng, W. Wang, R. Chen, and B. Li, "Gillis: Serving large neural networks in serverless functions with automatic model partitioning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 138–148.