Towards Cloud-Edge Collaborative Online Video Analytics with Fine-Grained Serverless Pipelines

Miao Zhang*, Fangxin Wang◇, Yifei Zhu†, Jiangchuan Liu*, Zhi Wang‡
* School of Computing Science, Simon Fraser University
◇ Future Network of Intelligence Institute (FNii) and School of Science and Engineering,
The Chinese University of Hong Kong, Shenzhen
† University of Michigan - Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University
‡ Shenzhen International Graduate School, Tsinghua University
mza94@sfu.ca,wangfangxin@cuhk.edu.cn,yifei.zhu@sjtu.edu.cn,jcliu@cs.sfu.ca,wangzhi@sz.tsinghua.edu.cn

ABSTRACT

The ever-growing deployment scale of surveillance cameras and the users' increasing appetite for real-time queries have urged online video analytics. Synergizing the virtually unlimited cloud resources with agile edge processing would deliver an ideal online video analytics system; yet, given the complex interaction and dependency within and across video query pipelines, it is easier said than done. This paper starts with a measurement study to acquire a deep understanding of video query pipelines on real-world camera streams. We identify the potentials and practical challenges towards cloud-edge collaborative video analytics. We then argue that the newly emerged *serverless computing* paradigm is the key to achieve fine-grained resource partitioning with minimum dependency. We accordingly propose CEVAS, a *Cloud-Edge collaborative Video Analytics system empowered by fine-grained Serverless pipelines.* It builds flexible serverless-based infrastructures to facilitate fine-grained and adaptive partitioning of cloud-edge workloads for multiple concurrent query pipelines. With the optimized design of individual modules and their integration, CEVAS achieves real-time responses to highly dynamic input workloads. We have developed a prototype of CEVAS over Amazon Web Services (AWS) and conducted extensive experiments with real-world video streams and queries. The results show that by judiciously coordinating the fine-grained serverless resources in the cloud and at the edge, CEVAS reduces 86.9% cloud expenditure and 74.4% data transfer overhead of a pure cloud scheme and improves the analysis throughput of a pure edge scheme by up to 20.6%. Thanks to the fine-grained video content-aware forecasting, CEVAS is also more adaptive than the state-of-the-art cloud-edge collaborative scheme.

CCS CONCEPTS

• **Information systems → Multimedia information systems**;
• **Networks → Cloud computing**.

KEYWORDS

Video Analytics, Serverless Computing, Cloud-Edge Collaboration

ACM Reference Format:
Miao Zhang, Fangxin Wang, Yifei Zhu, Jiangchuan Liu, Zhi Wang. 2021. Towards Cloud-Edge Collaborative Online Video Analytics with Fine-Grained Serverless Pipelines. In *12th ACM Multimedia Systems Conference (MMSys '21) (MMSys 21), September 28-October 1, 2021, Istanbul, Turkey.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458305.3463377

1 INTRODUCTION

Recent years have witnessed an explosive increase in camera deployment. For instance, it was reported that there would be one billion surveillance cameras watching around the world in 2021 [17], and the global surveillance camera market is predicted to account for US$ 45.93 billion in 2027 [45]. To fully unleash the potentials of these deployed cameras, video analytics applications have been developed to help various public and private entities (e.g., law enforcement agencies and retail stores [31]) to increase efficiency, reduce costs, and improve security. Specifically, the goal of these video analytics applications is to answer users' queries about spatial and temporal events occurring in video streams (e.g., How many cars are there in the video stream during rush hours today?). The ever-growing deployment scale of cameras and users' increasing appetite for real-time queries have inspired the need for designing high-performance online video analytics systems.

The advent of deep neural networks (DNNs) has revolutionized video analytics accuracy with higher resource demands [16, 21]. To overcome the resource challenge of high-accuracy video analytics at scale, the status quo solutions either resort to the resource-rich cloud to build virtual machine (VM) clusters or invest in powerful hardware to build private clusters [34, 55]. Nevertheless, this solution inevitably introduces latency caused by variable network conditions, especially when streaming high-definition (HD) videos through wireless networks [13]. Alternatively, analyzing video streams at the edge can provide real-time insights and reduces the cloud and bandwidth costs, but suffers severe performance degradation due to the constrained edge resources [13, 33]. An ideal solution to solve such a fundamental dilemma is to *opportunistically* push partial analytics tasks from the cloud to the edge for latency and costs reduction [13]. In real-world deployments, designing such a cloud-edge collaborative online video analytics system is easier said than done, and it presents several non-trivial challenges in:

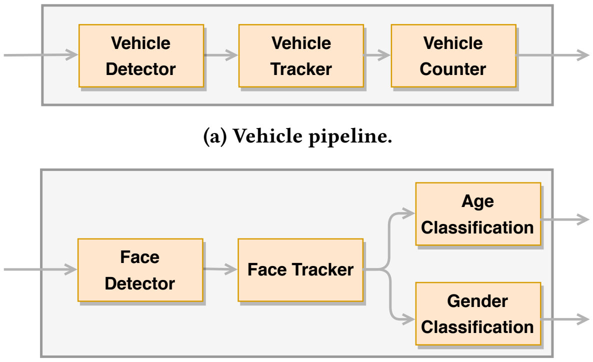
MMSys 21, September 28-October 1, 2021, Istanbul, Turkey

ACM ISBN 978-1-4503-8434-6/21/09…$15.00
https://doi.org/10.1145/3458305.3463377

- *Avoiding unnecessary resource provisioning*. Although cameras can generate video streams 24 × 7, provisioning a dedicated cluster to analyze all of them is unnecessary since the events of interest can be rare and only last for a short period (such as flame or smoke) [13, 28, 52, 56]. Thus, traditional resource provisioning strategies, such as configuring a cluster according to the peak utilization or one-time offline profiling, lead to low resource efficiency.
- *Addressing fine-grained content dynamics.* A typical video analytics query involves a set of cascade computer vision primitives [30, 34, 55], such as object detection [44], object tracking [11, 12, 50], image classification [33]. These primitives are coupled with each other through video content and form a query pipeline. The input workload and further resource demand of each primitive are affected by video content dynamics. Unfortunately, as revealed in our measurement study (details in §2.2), the video content dynamics are fine-grained and cannot be adapted by coarse-grained VM-based solutions.
- *Handling multi-tenancy at the edge.* The edge video analytics can be executed at smart cameras [32, 35] or edge servers [33]. Considering the already widely deployed low-cost non-smart cameras, we focus on the latter case. Assume multiple cameras are connecting to an edge video analytics server. For a camera stream, different query pipelines can glean information of interest simultaneously. For instance, for a camera mounted at a busy intersection, a car counting query only pays attention to cars, while a jaywalker query is interested in pedestrian behavior. As such, multiple concurrent queries on multiple streams can be submitted to the same edge server and compete for the limited resources, thereby forming a multi-tenant environment.
- *Identifying the "sweet spot" for collaboration.* For multiple concurrent video queries, which primitives of which queries should be placed on the edge server or in the cloud is a fundamental issue. Finding out the best partition strategy is tricky since different primitives have different edge-execution resource demands, cloud-edge data transfer costs, and cloud-execution expenditures.

In this paper, we propose CEVAS, a novel *Cloud-Edge collaborative online Video Analytics system with fine-grained Serverless pipelines*, to tackle the aforementioned challenges. To the best of our knowledge, CEVAS is the *first* video analytics system that introduces *serverless computing* to address the fundamental challenges in cloud-edge collaboration. Specifically, the fine-grained and automatic resource management provided by serverless computing helps CEVAS dramatically improve the resource efficiency in the cloud. The fine-grained, scalable, and lightweight compute infrastructure of serverless computing also facilitates the real-time *workload-aware* revision of the cloud-edge partition strategy. It simultaneously creates the possibility of handling *unpredictable* video query patterns and *fine-grained* video content dynamics.

CEVAS is a video analytics system spanning the cloud and edge that can automatically and dynamically identify the optimal cloud-edge partition points for concurrent serverless video analytics pipelines and orchestrates their executions between the cloud and edge. In particular, for each online serverless video analytics pipeline, it effectively forecasts the video content-dependent resource demand, cloud expenditure, and data transfer overhead for different cloud-edge partition points. Then, it identifies the best partition strategy for all online pipelines according to the prediction results and available edge resources to minimize the cloud execution costs while maximizing the edge resource efficiency. With the optimized design of individual modules and their integration, CEVAS achieves real-time responses as well as fine-grained scalability and adaptation. In general, our contributions can be summarized as follow:



(a) Vehicle pipeline.

(b) Face pipeline.

Figure 1: Two representative types of video query pipelines.

- To acquire a deep understanding of how video content affects the input workloads of cascaded primitives in video query pipelines, we analyze massive camera videos collected from the real world. By introducing two metrics for characterizing video content dynamics, our measurement study justifies the considerable benefits of designing a fine-grained content-aware pipeline partition strategy and provides insights for predicting the content-dependent resources and costs of each pipeline primitive.
- We present CEVAS, a video analytics system that integrates the edge and cloud resources to have the best of both worlds and leverages serverless computing to tackle the challenges that arise in the collaboration. We further present the detailed design of the *Controller* of CEVAS. Through the video content-aware forecasting and fine-grained workload-aware runtime scheduling, the *Contoller* can dynamically adjust the cloud-edge partition strategy for online video query pipelines, thus handling the ever-changing input workloads.
- To validate the real-world performance of CEVAS, we overcome the challenges of integrating it into the real-world cloud computing platforms and implement a prototype with Amazon Web Services (AWS) [5] and off-the-shelf edge devices. Extensive evaluations demonstrate that CEVAS can handle multi-tenant video queries in near real-time with significantly reduced cloud expenditure (13.1% of *PureCloud*) and data transfer overhead (25.6% of *PureCloud*). Furthermore, it is more adaptive than the content-agnostic cloud-edge collaborative video analytics scheme.

2 MEASUREMENT AND MOTIVATION

In this section, we first provide an introduction to typical video query pipelines. Then, based on two representative video query pipelines and two real-world camera streams, we reveal remarkable insights about how to characterize video content dynamics and

Primitive	Model	DNN	Input	Output
Vehicle Detector	YOLOv3 [44]	Yes	F	BB
Vehicle Tracker	SORT [12]	No	BB	T
Vehicle Counter	Self-implement	No	T	R
Face Detector	CenterFace [53]	Yes	F	BB + I
Face Tracker	SORT [12]	No	BB + I	I
Age Classification	AgeNet [40]	No	I	R
Gender Classification	GenderNet [40]	No	I	R

Table 1: The default model choices and I/O specifications of vehicle and face query pipelines. "F" refers to frames; "BB" refers to bounding boxes; "T" refers to tracklets of vehicles; "I" refers to face images; "R" refers to the query results.

design a high-performance video analytics system. Finally, we discuss the opportunities brought by serverless computing to handle fundamental challenges in building online video analytics systems.

2.1 Video Query Pipelines

Common computer vision primitives involved in advanced video analytics pipelines include object detection [44], object tracking [11, 12, 50], image classification [33], recognition [24], etc. Figure 1 displays two representative video query pipelines discussed in this paper. These two query pipelines have widespread usage in our daily life. For example, the vehicle pipeline can help manage traffic by counting vehicles appearing in a camera installed in a crossroad or a parking lot. The face pipeline can help profile customers by classifying the age and gender of customers appearing in the video stream of a store camera.

When a video stream is fed to the vehicle pipeline, the detector primitive is first executed to extract vehicles. It then sends the detected result (the bounding boxes of detected vehicles) to the following tracker primitive. After processing, the tracker primitive sends the entire movement trajectory (a series of centroid coordinates) of the tracked vehicle to the subsequent counter primitive for directional counting. For the face pipeline, the face detector primitive is responsible for extracting all faces appearing in the input video stream and sending cropped face images and their bounding boxes to the face tracker, which excludes duplicated faces. After that, the deduplicated face images are sent to downstream primitives for age or gender classification. The I/O specifications of each primitive in the vehicle and face pipelines are shown in Table 1.

Each primitive in a video query pipeline has multiple configuration knobs (such as frame rate, resolution, and implementation algorithms). The choices of these knobs determine resource demands (i.e., resources required to catch up with the input data rate) and the analytical accuracy. Depending on the use case, one can tune these knobs according to the performance goals. Generally, knob values leading to high accuracy require high resource demands (e.g., 30 FPS, 1080p, and a state-of-the-art DNN based algorithm). In contrast, knob values with lower resource demands tend to sacrifice accuracy (e.g., 1 FPS, 240p, and a classical computer vision algorithm).

How to configure the configuration knobs to strike a balance between resource demand and accuracy is a hot topic in data center resource management, and several frameworks or systems [30, 34, 55] have been proposed to provide guidelines for users. In this paper, we assume users have tuned these knobs according to their accuracy expectations in advance and will focus on designing a cloud-edge collaborative video analytics system. In particular, the default implementation model or algorithm choices for the vehicle and face pipelines are shown in Table 1.

Stream Name	*Crossroad*	*Restaurant*
Frame Rate	10 FPS	1 FPS
Resolution	1080p	1080p
Query Type	Vehicle pipeline	Face pipeline
Video Source	YouTube Live [48]	YouTube Live [47]
Description	A busy crossroad	A roadside restaurant

Table 2: Real-world camera streams used in this paper.

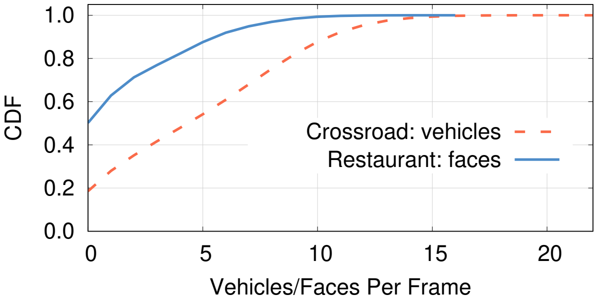
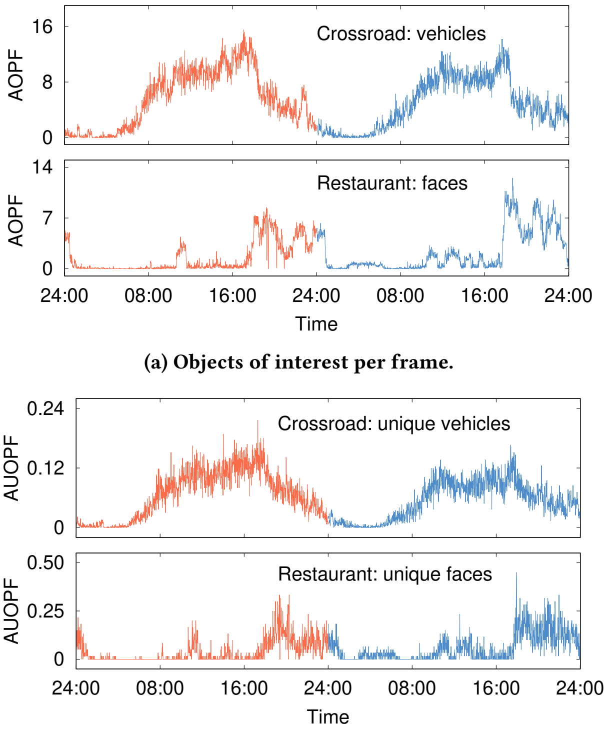


Figure 2: CDF of objects of interest.

2.2 Characterizing Real-world Video Streams

To identify real-world video stream characteristics towards supporting the design of cloud-edge collaborative video analytics systems, we analyze two consecutive 48-hour video clips collected from two real-world cameras (details in Table 2). To ensure reasonable analytical accuracy, we set the resolution of both streams to 1080p and the frame rate for *Restaurant* stream at 1 FPS and *Crossroad* stream at 10 FPS (as the mobility and speed of vehicles are higher than humans.). Concretely, by analyzing these two real-world video streams, we are trying to answer the following questions: (1) How often do the objects of interest appear in 24 × 7 video streams? (2) How does video content change with time? (3) How do video content dynamics affect the input workloads of cascaded primitives in a video query pipeline?

Vehicles and human faces are the most frequent object classes for the *Crossroad* stream and *Restaurant* stream, respectively. Despite this, as demonstrated in Figure 2, with state-of-the-art DNN detectors (details in Table 1), there are still 18.58% frames of the *Crossroad* video clip without vehicles being detected and 50.19% frames of the *Restaurant* video clip without faces being detected. This observation indicates that the streams have no content worth querying for a notable portion of the time. *This means there is no need to allocate or provision dedicated resources for video steams and suggests a considerable potential to save network and compute resources by developing a content-aware resource scheduler.*



(a) Objects of interest per frame.

(b) Unique objects of interest per frame.

Figure 3: Camera stream characteristics (statistics of two consecutive days.)

The number of detected objects of interest (vehicles or faces) directly impacts the tracker primitives' input workloads as it determines how many trackers the tracker primitives need to maintain, thereby influencing their resource demands. For detectors that employ cascade models, the number also has a considerable effect on the detector's resource demand. The number of deduplicated vehicles indicates the input workload of the counter primitive since it determines how many vehicle tracklets need to be classified. For a similar reason, the number of deduplicated faces indicates the input workload of the downstream classifier primitives. To further quantify video content dynamics, we define the *average objects per frame (AOPF)* metric and the *average unique objects per frame (AUOPF)* metric as follows:

$$AOPF = \frac{\sum_{i \in T} \#\ of\ detected\ objects\ in\ frame\ i}{\#\ of\ frames\ in\ time\ window\ T} \tag{1}$$

$$AUOPF = \frac{\#\ of\ deduplicated\ objects\ in\ time\ window\ T}{\#\ of\ frames\ in\ time\ window\ T} \tag{2}$$

We present the variations of these two metrics over time in Figure 3a and Figure 3b (T is set to 1 minute in these two figures). As one can see, both metrics show apparent variations across the day on both video streams, which indicates one-time offline or coarse-grained online profiling is not able to capture the content dynamics. *Thus, fine-grained workload profiling or prediction is necessary to adapt to the video content and achieve the most benefits.* Fortunately, we also observe fine-grained temporal correlation, which can be attributed to the fact that vehicles or faces usually occur across multiple frames, and it takes time for them to fade from the camera's field. *This observation inspires us to predict these two metrics by incorporating time-series dependency information.*

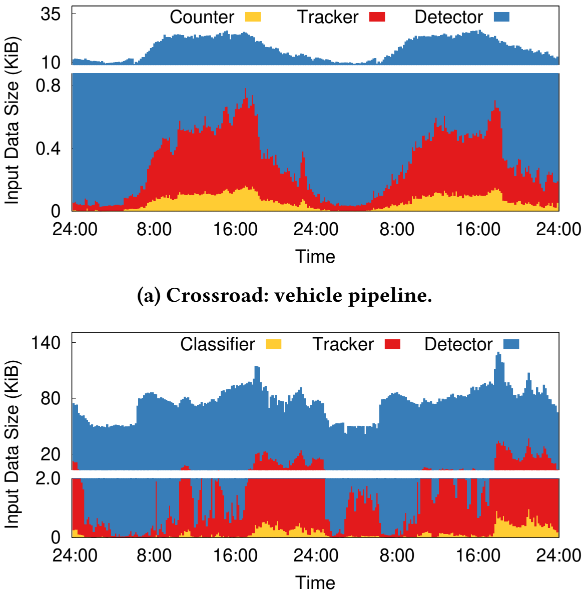
Video content dynamics can also affect the input data size of each primitive, which is crucial for a cloud-edge collaborative system as it determines the amount of data to be transferred between decoupled primitives. Figure 4 shows the input data size variations for cascaded pipeline primitives. One can observe that as the stream goes down the query pipeline, the generated intermediate data size dramatically decreases, indicating the great potential of cloud-edge collaboration in reducing bandwidth consumption. In addition, compared this figure to Figure 3a and Figure 3b, we observe notable positive correlations between the input data size of the tracker (counter) primitive and *AOPF* (*AUOPF*). However, we failed to fit them with simple linear functions since other variables (such as the cropped image size and the length of vehicle trajectories) can also affect the input data size. *This fact means the object-related variables (AOPF and AUOPF) and the input data size variables cannot substitute for each other. They are all valuable metrics to characterize video content dynamics.*

2.3 Opportunities Brought by Serverless Computing

The bursty input workloads and fine-grained video content dynamics that widely exist in video analytics inspire us to introduce serverless computing [4, 36, 51]. As a general-purpose compute abstraction, serverless computing has emerged to minimize the tedious administration operations and provide *fine-grained autoscaling* computing infrastructures [36]. Recent years have witnessed the commercial success of serverless computing represented by *Function as a Service (FaaS)* offerings (e.g., AWS Lambda [38], Google Cloud Functions [25], and Microsoft Azure Functions [23]). For instance, it is reported that AWS Lambda has already been adopted by nearly half of the companies with infrastructure in AWS [18].

Serverless computing is a good fit to build a wide variety of applications, such as web microservices and backends for IoT applications [2]. Recent efforts have been made to extend its application to heavy jobs, such as data analytics [42], machine learning [14], and video processing [7, 22]. As the public cloud extends to the edge, developers can further deploy and run serverless functions in the content delivery network (CDN) (e.g., AWS Lambda@Edge [39]) and IoT devices (e.g., AWS IoT Greengrass [26]), facilitating the design of cloud-edge collaborative serverless systems.

In FaaS platforms, monolithic application codes are decoupled into a series of stateless *functions*. Each function implements a microservice. It can be configured and invoked independently. The same function code can be executed by multiple *function instances* typically implemented by lightweight containers or sandboxes [2, 4]. Thanks to the lightweight virtualization technology, function instances can scale up or down automatically in milliseconds based on their input workloads, leading to rapid and flexible responses. This advantage empowers serverless functions to handle fine-grained input workload variations without resource management and monitoring, achieving fine-grained scalability and adaption. Moreover,



(a) Crossroad: vehicle pipeline.

(b) Restaurant: face pipeline.

Figure 4: Input data size variations of cascade primitives, averaging on all frames in a 10-minute window. The input data sizes for detector primitives are compressed frame sizes.

the pay-as-you-go pricing strategy of FaaS can ensure no money wasted on idle resources, thereby reaching high cost-efficiency. Motivated by the advantages of serverless computing in handling fine-grained input workloads and building cost-effective systems, we advocate introducing serverless computing in cloud-edge collaborative video analytics.

3 SYSTEM DESIGN

In this section, we first introduce the serverless-based system design of CEVAS. Then, we provide an overview description of CEVAS' architecture. Finally, we show how CEVAS coordinates the edge and cloud resources to handle the dynamic video query workloads.

3.1 Serverless-based System Design

To deploy a video analytics application with serverless computing, we first need to break the monolithic code into a set of serverless functions. For example, we can implement each primitive in Figure 1a as a standalone serverless function. As such, the whole video query pipeline becomes a serverless pipeline of three cascade functions. Based on these serverless video analytics pipelines, we propose CEVAS, a novel *Cloud-Edge collaborative online Video Analytics system with fine-grained Serverless pipelines.*

To have the best of both cloud computing and edge computing, CEVAS is designed to be a cloud-edge collaboration system that can handle multiple concurrent video queries in real-time. Each query requires executing a video query pipeline, a directed acyclic graph (DAG) of serverless functions, on its input video stream. The main task of CEVAS is to flexibly partition and orchestrate the execution of concurrent query pipelines between the cloud and edge. Specifically, for each query, CEVAS finds a partition point, where serverless functions before the point are executed on the edge server, and functions after the point are executed in the cloud. All partition points of concurrent query pipelines constitute a *partition strategy*. By dynamically tuning the partition strategy, CEVAS aims to (1) achieve high-throughput and cost-effective video analytics, (2) achieve adaption to dynamic input workloads.

To achieve fine-grained scalability and adaptation, CEVAS operates in a small timeslot-based way, which allows CEVAS to revise the partition strategy every timeslot in response to the input workload variations flexibly. Being serverless provides readily available infrastructures to facilitate this fine-grained revision because after processing its input workload, the serverless function instance will automatically terminate and release occupied resources. The length of a timeslot can be set by users and is advised to be small enough to enable fine-grained responses. For example, in our implementation, the default value is 5 seconds, which is the group of pictures (GOP) length of the videos we used.

3.2 Architecture of CEVAS

We define the *stream query pipeline* as executing a specific serverless pipeline on a specific camera stream. We regard the same query pipeline executed on distinct camera streams or different query pipelines executed on the same camera stream as distinct stream query pipelines. Figure 5 shows the architecture of CEVAS, which involves four entities (Users, *Controller*, Edge server, and Cloud). For illustration purposes, we assume that three cameras connect to the edge server and continually stream their captured videos to it. We further assume that there have been 3 stream query pipelines deployed to the system, i.e., a face query pipeline for camera stream #2, a vehicle query pipeline for camera stream #1, and a vehicle query pipeline for camera stream #3.

It should be noted that stream query pipelines are pipelines of serverless functions, which will not be executed unless responding to a user request. We say a stream query pipeline is online at a timeslot if there is a request for executing the stream query pipeline at the timeslot. For instance, Figure 5 shows a snapshot of a timeslot where the face query for stream #2 and the vehicle query for stream #1 are online while the vehicle query for stream #3 is offline. Thanks to the small timeslot, it is reasonable to allow the *Controller* of CEVAS to schedule only once in a timeslot. We assume that users submit queries at the end of each timeslot so that the *Controller* can schedule the partition strategy for the next timeslot at the end of the current timeslot.

Specifically, CEVAS operates in the following way: at the end of timeslot t, the *Controller* identifies all stream query pipelines that are online at timeslot $t + 1$ (Step 1 in Figure 5). It then schedules the partition strategy for timeslot $t + 1$ according to online streams' characteristics and available resources on the edge server. At the beginning of timeslot $t + 1$, the partition strategy is then sent to the edge server (Step 2 in Figure 5). Based on the partition strategy, the edge server starts to execute the stream query pipelines. When encountering the partition point of a stream query pipeline, the edge server sends the partial (intermediate) execution results (Step 3 in Figure 5) to the cloud for further execution.Once the executions complete, the cloud and the edge server send their execution

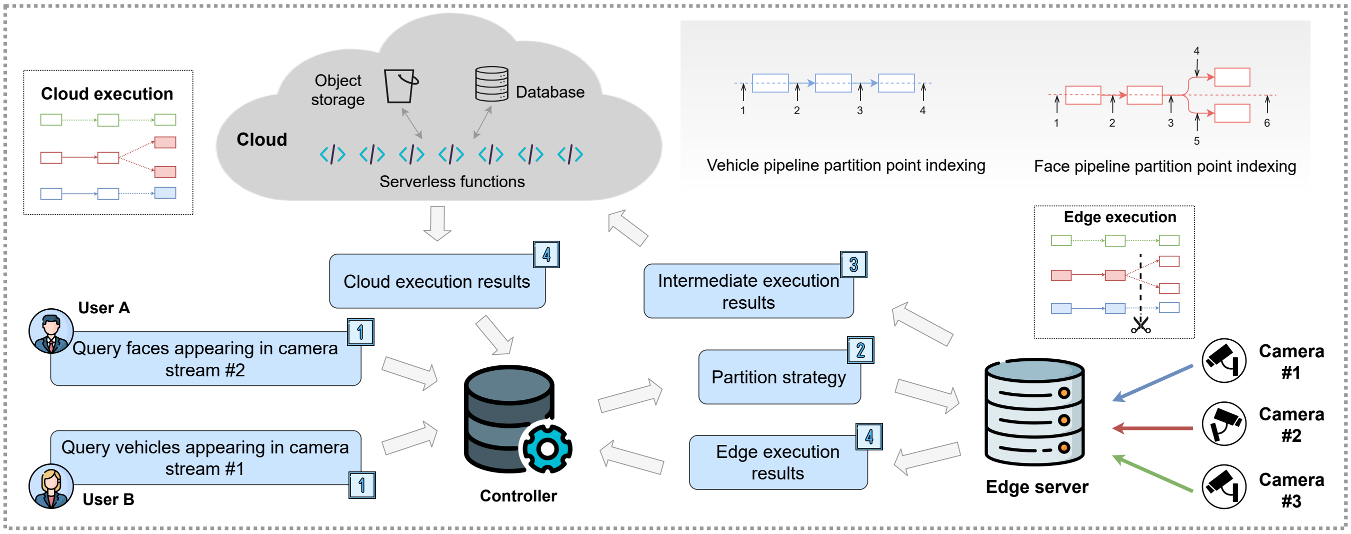


Figure 5: Architecture of CEVAS. In this snapshot, user A issues face query on camera stream #2 while user B issues vehicle query on camera stream #1. The scissors icon represents selecting partition point #3 for both online stream query pipelines, i.e., the first two primitives (functions) of both online query pipelines are executed on the edge server, and the other primitives (functions) are offloaded to the cloud for execution.

information of timeslot $t+1$ to the *Controller* for its future partition scheduling (Step 4 in Figure 5). Although there is only one edge server in Figure 5, CEVAS can be easily extended to a scenario with multiple edge servers by simply allowing the *Controller* to make partition strategies for them at the same time.

3.3 Workload-Aware Runtime Scheduling

CEVAS relies on the *Controller* to address the dynamic input workload challenge caused by time-variant online queries and video content. As shown in Figure 6, the *Controller* entails three components, which work together to provide fine-grained workload-aware scheduling for cloud-edge collaborative video analytics. Specifically, the *Monitor* component is responsible for extracting content-related metrics from the execution information reported by the cloud and edge server. The extracted information is then exploited by the *Predictor* component to forecast the edge-execution resource demand and cloud-execution cost of future timeslots. Finally, the forecasting results are passed to the *Scheduler* component for selecting the best partition strategy. In this section, we will introduce how to search for the best partition strategy in detail.

3.3.1 *Partition Problem Formulation.*

In our design, at the end of each timeslot, the *Scheduler* picks partition points for all online stream query pipelines. Each online stream query pipeline only has one partition point. The *Scheduler* indexes different partition points to distinguish between different partition options. Figure 5 demonstrates the partition points indexing examples of the vehicle and face pipelines. The indexing scheme integrates two special cases: #1 partition points mean executing the whole query pipelines in the cloud (*PureCloud* in §5.2). In contrast, the last indices (#4 for the vehicle pipeline and #6 for the face pipeline) mean executing the whole pipeline on the edge server (*PureEdge* in §5.2).

The number of online stream query pipelines for each timeslot t is determined since we have assumed that all query requests are submitted at the end of timeslot $t-1$. Let I_t denote the set of online stream query pipelines at timeslot t. For each stream query pipeline $i \in I_t$, let $x_{i,j}$ be a binary variable that gets 1 when the jth partition point of this pipeline is chosen, and let $P_{i,j}$ and $D_{i,j}$ respectively denote the corresponding cloud expenditure and edge-cloud data transfer overhead for the jth partition point in time slot t. According to the system goals, *the best partition strategy should be the one that minimizes the cloud expenditure and edge-cloud data transfer overhead while maintaining the desired analysis throughput.* Therefore, the problem of finding the best partition strategy for timeslot t can be formulated as follows:

$$\min_{x_{i,j}} \quad \sum_{i,j} (\alpha \cdot P_{i,j} \cdot x_{i,j} + \beta \cdot D_{i,j} \cdot x_{i,j})$$
$$s.t. \quad \begin{cases} \sum_{i,j} C_{i,j} \cdot x_{i,j} \le C \\ \sum_{i,j} G_{i,j} \cdot x_{i,j} \le G \\ \sum_{i,j} M_{i,j} \cdot x_{i,j} \le M \\ \sum_{i,j} M'_{i,j} \cdot x_{i,j} \le M' \\ \sum_{j} x_{i,j} = 1, \quad x_{i,j} \in \{0,1\} \end{cases} \tag{3}$$

where α and β are weights used to make a trade-off between the two optimization goals. $C_{i,j}$, $G_{i,j}$, $M_{i,j}$ and $M'_{i,j}$ are the edge-side CPU, GPU, memory and GPU memory demand respectively for the stream query pipeline i to reach the desired analysis throughput if the jth partition point is chosen. C, G, M, and M' are the CPU, GPU, memory, and GPU memory capacity of the edge server. The first four constraints ensure that the resource demands of all functions that are scheduled to execute at the edge server do not exceed the server's capacity. The last constraint ensures only one partition point for a stream query pipeline. Note that the solution to the

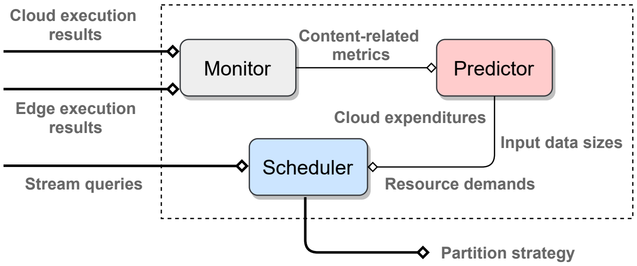


Figure 6: Components of the *Controller*.

problem is the partition strategy that is only valid for timeslot t. The *Scheduler* will reschedule the partition strategy for timeslot $t+1$ as new stream queries may be submitted, and old online stream queries may be terminated.

3.3.2 ***Handling Video Content Dynamics***. According to our observations in §2.2, video content dynamics can impact the input workloads of cascaded functions in the same pipeline. This means $P_{i,j}$, $D_{i,j}$, $C_{i,j}$, $G_{i,j}$, $M_{i,j}$ and $M'_{i,j}$ may vary with the analyzed video content. Unfortunately, when the *Scheduler* component schedules the partition strategy for timeslot t, $P_{i,j}$, $D_{i,j}$, $C_{i,j}$, $G_{i,j}$, $M_{i,j}$ and $M'_{i,j}$ are unknown since the corresponding video content has not been analyzed yet. In CEVAS, this issue is addressed by the *Predictor* component, which is responsible for predicting these content-dependent variables. Specifically, the *Predictor* component forecasts the resource demand, input data size, and cloud expenditure of each function in the pipeline.

Formally, for stream query pipeline i, we define $c_{i,k}$ ($g_{i,k}$, $m_{i,k}$, and $m'_{i,k}$) as the CPU (GPU, memory, and GPU memory) demand of its kth function being executed at the edge server. Let $d_{i,k}$ denote the input data size of its kth function and $p_{i,k}$ denote the cloud expenditure of the kth function being executed in the cloud. Note that for those content-agnostic variables, we execute offline profiling instead of relying on the *Predictor* component to predict (e.g., the max GPU memory demand of the vehicle detector function). Additionally, if one kind of edge resource is not used by a function, we set the corresponding values to 0 (e.g., $g_{i,2}$ and $m'_{i,2}$ are set to 0 since the tracker functions do not utilize the GPU resources in our implementation). With these predicted resource demands and costs of each function, the *Scheduler* can easily obtain the values of $P_{i,j}$, $D_{i,j}$, $C_{i,j}$, $G_{i,j}$, $M_{i,j}$ and $M'_{i,j}$. For instance, for a vehicle query pipeline processing stream i, $C_{i,3} = c_{i,1} + c_{i,2}$ and $D_{i,3} = d_{i,3}$.

Figure 7 shows an overview of the *Predictor* component, which solves an *multivariate time series forecasting* problem. The component predicts input data sizes (data transfer overhead), edge-execution resource demands, and cloud-execution expenditures in future timeslots for each serverless function with historical execution information. As demonstrated in our measurement study §2.2, the *AOPF* and *AUOPF* metrics are capable of characterizing the video content dynamics, which inspires us to incorporate *AOPF* and *AUOPF* metrics as parts of the *Predictor*'s inputs. In addition, the input data size of each function $d_{i,k}$ is also included as it is highly content-dependent based on our measurement. By contrast, we do not incorporate the historical edge resource demands and cloud expenditure of a function in the inputs of the *Predictor* component as we cannot fully obtain values of these variables for each timeslot. To be specific, if a function is executed in the cloud at timeslot t, its edge resource demands for timeslot t are unavailable. On the other hand, if it is executed at the edge server, its cloud expenditure will be unavailable for the timeslot. As a result, the input variables of the *Predictor* are historical values of *AOPF*, and *AUOPF* and $d_{i,k}$.

After determining the inputs and outputs of the *Predictor* component, we have to decide which prediction approach to use. There is a multitude of approaches to solving the multivariate time series prediction problem. One approach is building traditional autoregressive models, e.g., vector autoregression (VAR). Another method is converting the multivariate time series prediction problem into a supervised machine learning problem and then applying classical machine learning models (e.g., Random Forest (RF)) to make predictions. Recent advances in multivariate time series prediction focus on using various DNNs, such as convolutional neural networks (CNN) and long short-term memory (LSTM), for prediction.

In our scenario, video queries can occasionally be online and only executed for a short period, so collecting large training datasets to train complex models is costly and impractical. Moreover, the relationship between output and input variables can be very different for distinct stream query pipelines. The differences can be attributed to the executed pipeline structures (e.g., vehicle pipeline vs. face pipeline) and the queried video content. Thus, the *Predictor* has to maintain an independent model for each stream query pipeline to ensure accurate predictions. As such, the ideal prediction model should be *lightweight*, *accurate*, and *fast*. CEVAS exposes interfaces for users to specify which prediction models to use. After experimenting with several different models, we finally choose the multilayer perceptron model (MLP) in our implementation. Despite its simplification, it can perform well for our problem.

In our design, the *Scheduler* component sends the partition strategy to the edge server at the beginning of each timeslot t, say at time T_t. This means the *Predictor* component has to pass the forecasting results of timeslot $t+1$ to the *Scheduler* between time T_t and T_{t+1}. However, when the *Predictor* begins to make predictions for timeslot $t+1$, the *Monitor* component may not have received the execution results of the video chunk that begins to be processed at time T_t, e.g., the actual values of *AOPF* and *AUOPF* for timeslot t. Hence, when making predictions for timeslot $t+1$, the *Predictor* does not refer to the execution results of timeslot t, but only depends on earlier execution results from timeslot $t-h$ to $t-1$, where h is the reference window size.

Note that the aforementioned solution is valid only when the execution results of timeslot $t-1$ and earlier timeslots have been received by the *Monitor* component. Suppose the execution results of these timeslots have not been received when making predictions for timeslot $t+1$. It indicates that the edge server has been experiencing significant delays due to resource contention. In this case, the *Predictor* will not perform any predictions but notify the *Scheduler* to set the partition points of all online stream query pipelines to 1. As such, all workloads of timeslot $t+1$ are offloaded to the cloud, and the resource contention of the edge server can be relieved. If the values of input variables from $t-h$ to $t-1$ are not all available since the stream query pipeline is offline, we say the forecasting

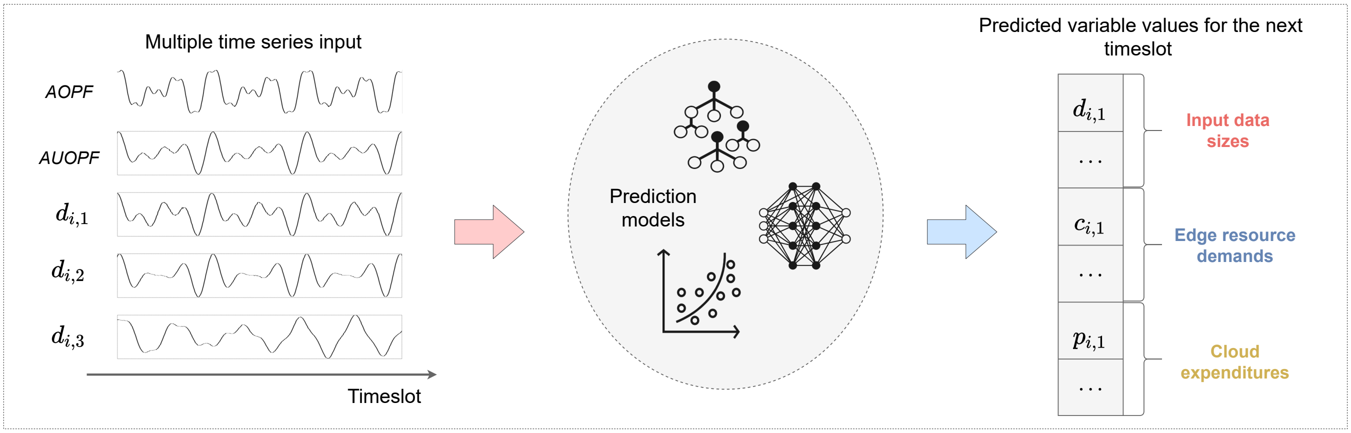


Figure 7: Overview of the *Predictor* component.

experiences a *cold start*. We take the mean historical values of input variables as the prediction result to address this issue.

3.3.3 ***Runtime Partition Scheduling.*** In our design, the models employed by the *Predictor* component are trained on datasets that are collected without multi-tenant pipelines competition. However, in a multi-tenant environment, the common resource contentions can increase the resources required to maintain desirable performance, especially for CPU and GPU resources. Therefore, once received the predicted values of $c_{i,k}$, $g_{i,k}$, $m_{i,k}$, $m'_{i,k}$, $p_{i,k}$ and $d_{i,k}$, the *Scheduler* component first executes *resource demand rectification* on $c_{i,k}$ and $g_{i,k}$. The *Scheduler* pessimistically rectifies them to the max possible values based on rules we empirically obtain. For example, for the vehicle detector primitive, the CPU time will increase by 10% in the worst case. After rectification, the *Scheduler* calculates the corresponding values of $C_{i,j}$, $G_{i,j}$, $M_{i,j}$, $M'_{i,j}$, $P_{i,j}$ and $D_{i,j}$ for each stream query pipeline i.

After these values are determined, the multiple stream query pipeline optimization problem (3) can be mapped to the *multi-dimensional multi-choice knapsack problem (MMKP)* [3]. For MMKP, there are n groups of items (n online stream query pipelines) and each group i has l_i items (each stream query pipeline i has l_i partition points). Each item in group i (each partition point in stream query pipeline i) has a value (the objective function value of problem (3)) and requires m resources (the edge-execution resource demand). The MMKP allows one to select exactly one item from a group, which corresponds to scheduling exactly one partition point for each online stream query. The goal of MMKP is to maximize the total value of selected items, which is equivalent to maximizing the opposite of the objective function value of the problem (3). Although MMKP is an NP-hard problem, several highly effective heuristics algorithms have been designed for solving it [3, 27, 49].

In our implementation, we choose the algorithm based on constructing convex hulls [3]. The running time complexity of this algorithm is $O(nlm + nl \log l + nl \log n)$, where n is the number of online stream query pipelines, m is the dimensions of the edge resources, and l is the maximum number of partition points that a stream query pipeline can have. Fortunately, one stream query pipeline has only a limited number of partition points in practice, indicating l is constant. In addition, the dimension of resources m we consider in problem (3) is 4. Thus, the time complexity of the algorithm becomes $O(n \log n)$. Due to the resource-intensive nature of video analytics pipelines, n will not be too large for a resource-constrained edge server in practice. If we extend the *Controller* of CEVAS to schedule for s independent edge servers, the total scheduling time will be $O(sn \log n)$. When n is a small constant, this only costs a linear time. Moreover, from our partition scheduling experiences for an edge server with a limited number of online stream query pipelines, brute-force searching can also provide acceptable performance.

4 SYSTEM IMPLEMENTATION

We prototype CEVAS with a public cloud computing platform: AWS. The cloud-side implementation is based on AWS's serverless computing platform: AWS Lambda. We implement each primitive of video query pipelines in a serverless function using Python language and deploy them to the platform. We configure 3008 MB for all functions and also enable parallel acceleration to deliver high-performance cloud-side processing. The parallel acceleration means, for image-based primitives, multiple function instances are invoked in parallel to accelerate the processing. This is a common trick to bring serverless's highly parallelizable superiority into full play [7, 22]. Thanks to the event-driven programming model of AWS Lambda, the instance invocation process can be automatically completed by setting appropriate event sources. We resort to cloud storage services to persist data between cascade functions that are executed in the cloud. In particular, for object-like data (e.g., video chunks, frames, and images), we leverage Amazon S3 [46] for transient storage. In contrast, for value data like bounding box coordinates, we leverage Amazon DynamoDB [19] for transient storage. We also use Amazon S3 to persistent the states of the tracker functions.

The edge server is equipped with an NVIDIA GeForce GTX 1080 GPU (with 8 GB memory), 12-core Intel Core i7-6850K CPU, and 32 GB of RAM. It runs Ubuntu 18.04 LTS. To extend the execution of AWS Lambda functions to the edge, we install AWS IoT Greengrass

Core SDK [9] on the edge device. For deployment, we create the corresponding edge-version Lambda function for each cloud-version Lambda function and deploy the edge-version functions to the edge server with AWS IoT Greengrass. The *Controller* is an off-the-shelf host (without GPU) installed AWS IoT Device SDK [8] to enable efficient and secure communication with Lambda functions on the edge server and in the cloud.

There is an additional stream ingestion Lambda function for each stream query pipeline at the edge server, which is responsible for kicking off the pipeline execution. Specifically, at the beginning of each timeslot, the *Controller* invokes the stream ingestion function for every online stream query pipeline with the pipeline partition point message. The stream ingestion function checks the partition point. It uploads the video chunk to be processed to Amazon S3 if the partition point is 1 (i.e., executing the whole pipeline on the video chunk in the cloud). Otherwise, it invokes the first function of the edge-version pipeline with the partition point message.

After processing the input workload, with the passed-in partition point message, the upstream function in the edge-version pipeline decides to invoke the edge-version subsequent function or the cloud-version subsequent function. If the edge-version is selected, the execution results and partition point message are both passed to the subsequent downstream function. Conversely, if the cloud-version is selected, the execution result is passed to the cloud-version subsequent downstream function. The execution information required by the *Controller* (e.g., *AOPF*) will also be sent from Lambda functions to the *Controller* if necessary. All messages are passed with the support of the corresponding AWS SDK through the MQTT protocol [1].

5 EVALUATION

In this section, we first describe the experiment setup. We then introduce the performance metrics and alternative schemes that are used for comparison. Finally, we show the experimental results.

5.1 Experimental Setup

Video Dataset: We obtain the *evaluation dataset* from the real-world camera streams where we collected our measurement datasets (shown in Table 2). Two video clips from the *Crossroad* camera stream and three video clips from the *Restaurant* camera stream are collected, and the length of each video clip is one hour. During experimentation, we stream these five pre-recorded video clips at the frame rate and resolution shown in Table 2 to the edge server, mimicking a multi-stream scene, where five cameras connect to the edge device: two cameras installed at busy crossroads, and the other three installed in stores or at the entrance. We choose to use pre-recorded videos rather than live streams to cover more complicated and challenging scenes.

Video Queries: Users can issue vehicle queries on the two *Crossroad* streams (indexed as stream query pipeline $Q1$ and $Q2$ respectively) and face queries on the three *Restaurant* streams (indexed as stream query pipeline $Q3$, $Q4$, and $Q5$, respectively). We implement and deploy these query pipelines as serverless function pipelines according to specifications in Table 1. In particular, to create a more complicated edge resources contention environment, we restrict the stream query pipeline $Q4$ and $Q5$ to use CPU merely. It means the edge-version face detector functions analyzing these two streams rely on CPU rather than GPU for DNN inference.

Pre-trained Prediction Models of the *Predictor* Component: As the prediction model should be lightweight, accurate, and fast, we choose the MLP model with one hidden layer to achieve the goals. We pre-trained an MLP model for the vehicle query pipeline on a 4-hour length clip of the *Crossroad* stream. We also pre-trained a GPU-version and a CPU-version prediction model for the face pipeline on a 6-hour length clip of the *Restaurant* stream. For simplification, the model architecture and hyperparameters are consistent for all stream query pipelines. The inference window size is 8, i.e., all models leverage the values of input variables from timeslot $t - 8$ to timeslot $t - 1$ to predict the values of output variables at timeslot $t + 1$. We tested the pre-trained models on the aforementioned *evaluation dataset*. The mean Root Relative Squared Error (RRSE)[1] of the pre-trained models is 0.064, and the mean inference time of processing an input sample is 0.028 CPU seconds.

5.2 Evaluation of CEVAS

Performance Metrics: To verify the effectiveness and efficiency of CEVAS, we define the following evaluation metrics.

- *Throughput.* CEVAS is a live video analytics system that analyzes videos as they are being recorded. As videos are being analyzed, the query results are accordingly updated. If the analysis rate cannot keep up with the incoming video data rate, the system will experience accumulated lags. The *throughput* metric is introduced to quantify if systems can achieve real-time analysis. Specifically, if a video stream is analyzed at a rate equal to its recorded frame rate, we say its *throughput* is 100%. Otherwise, the *throughput* is the percentage of the analysis frame rate over the recorded frame rate of the video stream. For example, for a 10 FPS video stream, 50% *throughput* means analyzing the video stream at 5 FPS, and in this case, the accumulated lag of the stream will be 2× its streamed duration.
- *Cloud expenditure.* The money paid for executing cloud-side serverless functions. CEVAS allows users to specify the compute method of this metric. In the evaluation, the values of this metric are computed based on the latest pricing strategy of AWS Lambda [6]. Note that this expenditure only involves the money paid for function execution and does not include the money paid for other cloud services (e.g., Amazon S3) to orchestrate serverless functions. This is because there are alternative ways to achieve function orchestration, and the orchestration cost depends on developers' specific implementation choices.
- *Transferred data.* The amount of data transferred between the edge server and the cloud, indicating how much network traffic is consumed.
- *Cost.* This is a composite metric of the *transferred data* metric and the *cloud expenditure* metric, and it is the optimization objective of Problem (3). By setting α and β, we can normalize the units of *transferred data* and *cloud expenditure* to US dollars. Unless otherwise noted, we assume the money paid for transferring 1 GB data from the edge to the cloud is \$0.1. Then we calculate the value of the *cost* metric by setting $\alpha = 1$ and $\beta = 0.1$.

[1] RRSE [37] is a normalized version of Root Mean Square Error (RMSE) that is employed to provide more readable results by eradicating the influence of data scale.

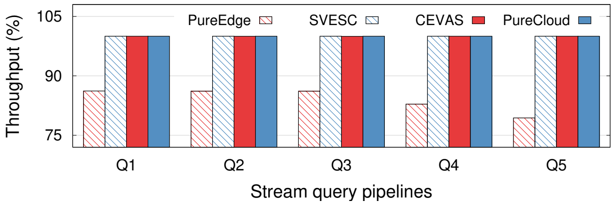


Figure 8: Throughput of different schemes on all five online stream query pipelines.

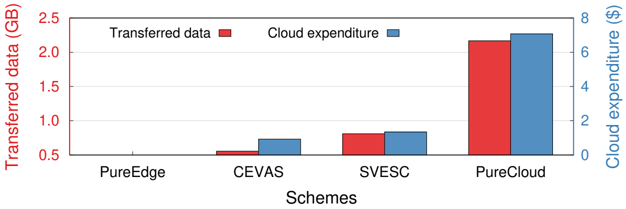


Figure 9: Transferred data and cloud expenditure of all schemes. Values shown in this figure are statistics over one-hour execution of all five stream query pipelines.

Baselines: We compare CEVAS with the following baselines:

- *PureCloud.* For this scheme, the camera streams being queried at a timeslot will be forwarded to the cloud for analysis. The edge server acts as a router, and no analytics will be executed on it. Assume the edge server has reserved sufficient bandwidth to support real-time video streaming to the cloud. Based on our current implementation of the cloud-side serverless pipelines, as long as the video streams can reach the cloud in real-time, they will be analyzed in real-time. This means the *throughput* of this scheme is always 100%.
- *PureEdge.* All online stream query pipelines are executed at the edge server. No data is forwarded to the cloud. Hence, the *cloud expenditure* and *transferred data* of the scheme are always 0.
- *SVESC.* This is a Slim version of VideoEdge [30] with *Serverless Computing* supports (*SVESC*). VideoEdge is a status quo VM cluster-based cloud-edge collaborative video analytics system. Video configuration knobs such as resolution and frame rate are fixed in this slim version, and we only tune the placement of primitives in a pipeline. To make this scheme more comparable to CEVAS, we re-implement it based on serverless functions. The main difference between this scheme and CEVAS lies in adapting to the video content dynamics or not. This scheme assumes that video content has minor influences on resource demands and costs. It leverages a small video sample as a representative to profile the edge resource demands, input data size, and cloud expenditure of each primitive in a pipeline. Then, it finds the partition strategy for each timeslot based on the same offline profiling results. In the evaluation, we select the first 5-second video chunk of each video stream as its representative sample.

5.2.1 Performance Improvement. We first present the experimental results of executing all five stream query pipelines concurrently for one hour (i.e., 720 timeslots). Figure 8 shows the *throughput* of each stream query pipeline in one hour. It can be seen that with constrained edge resources, the *PureEdge* scheme cannot catch up with the input frame rate and suffer significant accumulated lags. In contrast, all schemes that can access the cloud resources achieve real-time analytics. To be specific, with the cloud-edge collaboration, CEVAS and *SVESC* both improve the analytics throughput by up to 20.6% compared with the *PureEdge* scheme.

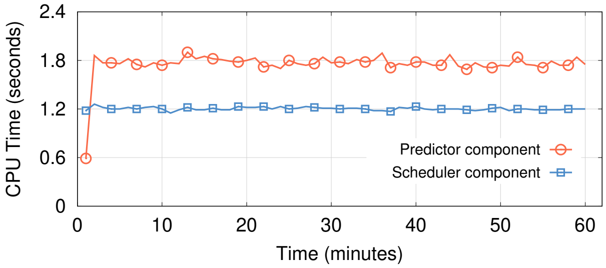
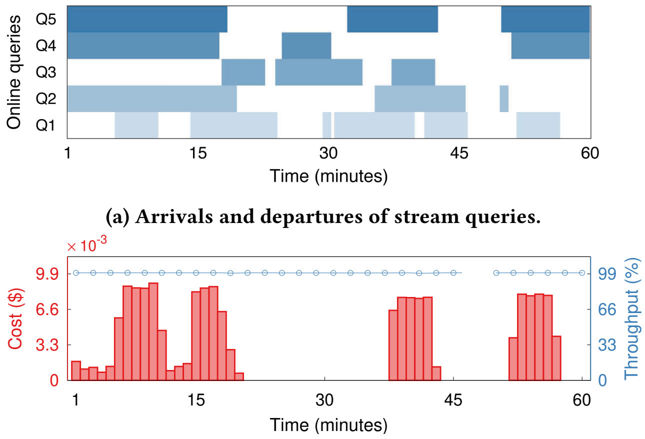


Figure 10: System overhead. The values shown in this figure are accumulated CPU times for analyzing one-minute length queries. The abnormal value of the *Predictor* component in minute 1 is caused by cold start (details in §3.3.2).

Figure 9 demonstrates the transferred data and cloud expenditures of all these schemes. Due to the lack of edge computing, the *PureCloud* scheme transfers nearly 2.2 GB data from the edge server to the cloud. Meanwhile, it costs about 7.1 US dollars to analyze the videos with AWS Lambda. By introducing cloud-edge collaboration and judiciously adjusts the partition points of online stream query pipelines, CEVAS reduces about 74.4% cloud-edge data transfer overhead and 86.9% cloud expenditure of the *PureCloud* scheme. This observation suggests that CEVAS can achieve real-time video analytics with only a small cost.

We can verify CEVAS's capability of handling video content dynamics by further comparing its performance with the content-agnostic *SVESC* scheme. Although these two schemes both achieve real-time analytics, compared to the *SVACES* scheme, CEVAS reduces its data transfer overhead by about 31.4% and cloud expenditure by about 30.9% (as shown in Figure 9). Unlike the *SVESC* scheme that employs one partition strategy for all timeslots, CEVAS can timely adjust its partition strategy according to its sensed fine-grained content variations. In fact, during the 720 timeslots of the experiment, 250 partition strategy updates are made by CEVAS to adapt to the fine-grained video content dynamics.

5.2.2 System Overhead. The primary system overhead of CEVAS is incurred by the *Predictor* and *Scheduler* components of its *Controller*. We measurement the CPU resource consumption of these two components of executing five stream query pipelines concurrently for one hour and show the results in Figure 10. It can be seen that in one minute, CEVAS consumes only about 1.8 CPU seconds to predict the resources and costs for all online stream query pipelines. Meanwhile, it only takes about 1.2 CPU seconds for CEVAS to find out the partition strategy. The overheads are minor for our *Controller* (one off-the-shelf host, details in §4). This indicates the system design of CEVAS is highly efficient and incurs minor overheads, which also



(a) Arrivals and departures of stream queries.

(b) CEVAS performance.

Figure 11: Scalability and Adaptability of CEVAS. The missing throughput values from minute 47 to minute 49 indicate that the system is idle, i.e., no online stream queries.

provides possibilities for CEVAS to work efficiently for massively concurrent video queries and a multitude of edge servers.

5.2.3 Scalability and Adaptability. In a real-world scenario, users can issue a query at any time according to their own needs, leading to an unpredictable usage pattern and system load pressure for the edge server. To validate the performance of CEVAS under unpredictable and bursty input workloads, we formulate the arrivals of stream queries as a Poisson Process with a mean interarrival time of 12 minutes (i.e., 144 timeslots). The service time of each query is randomly selected from three options of 1 minute, 10 minutes, and 20 minutes. If a newly arrived stream query is already being executed (i.e., multiple arrived stream queries have overlapping time intervals), we simply merge the time intervals of the queries to ensure that one video chunk is only analyzed once by the same pipeline. Figure 11a shows one generated usage pattern. In this figure, colorized cells for a stream query represent that the stream query is online at the corresponding timeslots.

Figure 11b shows the performance of CEVAS under the usage pattern shown in Figure 11a. The throughput shown in Figure 11b is the average of the *throughput* of all online stream queries in the corresponding minute. It should be noted that according to our definition, the *throughput* for a stream query cannot exceed 100%. This means only when all online stream queries have a nearly 100% *throughput*, the average throughput shown in Figure 11b is nearly 100%. As can be seen, no matter what kind of system load pressure, CEVAS always keeps high analytics throughput. When there are sufficient resources to handle the input workloads at the edge, no serverless functions are executed in the cloud, and the *cost* values are zero (e.g., from minute 21 to minute 35). On the other hand, when the edge server is handling heavy input workloads (e.g., from minute 6 to minute 11), CEVAS smartly pushes partial of the workloads to the cloud to maintain a high *throughput* for all online stream queries.

6 RELATED WORK

One-side video analytics systems: Single data center resource management has been discussed by several existing video analytics systems. The core technique used by them is balancing between resource and accuracy by adjusting general configuration knobs such as frame rate, resolution, and DNN models. For instance, VideoStorm [55] profiles the resource demand and accuracy of different knob combinations for each query offline and then adjusts configuration knobs for queries according to their quality and lag goals online. Chameleon [34] leverages the underlying temporal and spatial correlation to amortize frequent profiling costs incurred by adapting configuration knobs to the ever-changing video content. In contrast to these pure data center systems, Mainstream [33] focuses on multi-tenant video processing on a fixed-resource edge device. It automatically trades off between per-frame accuracy and frame processing throughput by dynamically tuning the degrees of DNN stem-sharing among concurrent applications. Unlike these cloud-only or edge-only systems, the cloud-edge collaborative architecture of CEVAS can effectively address the constrained edge resources or scarce network bandwidth challenges.

Collaborative video analytics systems: Vigil [56] is a real-time wireless video surveillance system that executes lightweight, stateless vision algorithms at the edge to prevent unrelated frames from being uploaded to the cloud and thus save wireless capacity. As compute-intensive DNN models gain popularity, DeepDecision [43] considers offloading a deep learning model from weak front-end devices (e.g., smartphones) to powerful backend helpers (e.g., edge servers/cloud). It adjusts the general configuration knobs (e.g., frame rate, bitrate) together with the placement knob (smartphone or edge/cloud processing) to achieve real-time and high-accuracy analytics. VideoEdge [30] harnesses the same configuration knobs tuning technique to allocate hierarchical clusters resources among concurrent video query pipelines to strike the desirable balance between resource demand and accuracy. Although VideoEdge solves the same multi-queries partitioning problem as CEVAS, it relies on a one-time offline resource profiling that cannot capture the content-dependent resource demand variations.

Serverless video processing: With the development of serverless computing, several efforts have been made to apply it for video processing tasks. For example, based on AWS Lambda, ExCamera [22] provides a general-purpose fine-grained parallel computation framework and further implements a low-latency serverless video encoder. Sprocket[7], a serverless video processing system, delivers high parallelism, low-latency, and cost-effective processing by leveraging the intra-video parallelism. Yet, both of them are pure cloud-based video processing systems that are more suitable for batch processing workloads. Different from these works, CEVAS attempts to introduce serverless computing into cloud-edge collaborative systems to unlock the potential of serverless computing in achieving scalable real-time video analytics.

7 DISCUSSION AND FUTURE WORK

Adaptation for more complicated DNN models: In our current prototype, we implement a computer vision primitive (e.g., object detector) as a standalone serverless function. However, a serverless function's resources may be insufficient to support real-time inference for resource-intensive DNN models, especially for those taking a video sequence as inputs (e.g., models typically developed for action recognition tasks [15]). For example, the maximum configurable memory size for a cloud function in AWS Lambda is 10 GB, and there is a lack of hardware accelerator support [10].

The easiest way to enable CEVAS to address this issue is DNN model splitting [20, 29, 41]. To be specific, we can split a large DNN model into multiple parts and implement each part with a serverless function. Consequently, one function with a large DNN model is converted to multiple cascaded functions with partial model layers and reduced resource demands. From the perspective of CEVAS, the only difference is that the converted pipeline has more cascade functions, which is still addressable. Although beyond the scope of this work, decoupling a monolithic video analytics application into a serverless pipeline is an exciting topic worth further exploration.

Adaption for network bandwidth: In this work, we assume sufficient bandwidth is provisioned for the edge server to stream videos to the cloud in real-time, i.e., the network bandwidth is not the bottleneck of the entire system. Although large organizations or entities provision dedicated network links for their video analytics systems, small businesses or personal users may resort to public or wireless networks to reduce costs. Unfortunately, wide-area network (WAN) bandwidth has proved to be scarce, variable, and expensive [54]. The network conditions between edge servers and clouds or even between cameras and edge servers can be highly unstable. In order to adapt to the network bandwidth variations, jointly optimizing the cloud-edge partition strategies with other pipeline configuration knobs (e.g., input frame rates, primitive model choices) can be necessary to achieve the most benefits. The key to realizing desirable performance goals is trading off between analytical throughput, accuracy, and cloud expenditure. This shows non-trivial challenges, and we leave this for future work.

Collaborations between edge servers: In our scenario, edge servers can be any devices that have onboard computation resources and are located in the proximity of cameras. Compared with clouds that can provide virtually unlimited resources, edge servers have physically constrained resources that tend to be heterogeneous. By enabling collaborations between individual edge servers and the cloud, CEVAS breaks the limitation of individual edge server's resources at the cost of cloud-edge network bandwidth consumption and additional cloud expenditure. Despite this, there is still room for improvement by further integrating resources at the edge.

A recent study has shown that cameras' workloads have a substantial heterogeneity [32], which will lead to heterogeneous workloads for edge servers. This creates possibilities for edge servers to share resources. For instance, a function running on an overloaded edge server can be rescheduled to run on a nearby idle server, thus improving the resource efficiency and scalability of the entire system while reducing cloud expenditure. Despite the promising prospects, introducing collaborations between edge servers also introduce new challenges. For instance, how to orchestrate serverless pipelines across geo-distributed infrastructures; how to strike throughput-cost trade-offs between local execution, peer-edge-server execution, and cloud execution. Overall, introducing collaborations between edge servers in CEVAS is an interesting future research direction.

8 CONCLUSION

In this paper, we present a cloud-edge collaborative serverless video analytics system CEVAS. By judiciously integrating the cloud and edge resources, CEVAS overcomes the drawbacks of pure cloud or pure edge schemes, realizing scalability and cost-effectiveness. To address the fundamental challenges in designing such a cloud-edge collaborative online video analytics system, CEVAS introduces serverless computing for fine-grained and flexible resource management. In particular, by leveraging video content-aware forecasting and fine-grained workload-aware runtime scheduling, the *Controller* of CEVAS solves a cloud-edge partitioning problem for multiple concurrent serverless pipelines with ever-changing input workloads. Experiments on AWS show that CEVAS can significantly reduce the data transfer overhead and cloud expenditure while maintaining high throughput. It is also more adaptive than the content-agnostic cloud-edge collaborative scheme.

ACKNOWLEDGMENTS

This work is supported by an Industry Canada Technology Demonstration Program (TDP) Grant. Fangxin Wang's work is supported in part by the Key Area R&D Program of Guangdong Province with Grant No. 2018B030338001. Yifei Zhu's work is supported by an SJTU Explore-X Research Grant. Zhi Wang's work is supported by a Shenzhen Science and Technology Program with Grant No. RCYX20200714114523079.

REFERENCES

[1] 2021. MQTT: The Standard for IoT messaging. https://mqtt.org. [Online; accessed 28-Apr-2021].

[2] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D. Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *Proceeding of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*. 419–434.

[3] M. M. Akbar, M S. Rahman, M. Kaykobad, E. G Manning, and G. C Shoja. 2006. Solving the Multidimensional Multiple-choice Knapsack Problem by constructing convex hulls. *Computers & operations research* 33, 5 (2006), 1259–1273.

[4] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt. 2018. SAND: Towards High-Performance Serverless Computing. In *2018 Usenix Annual Technical Conference (ATC'18)*.

[5] Amazon. 2021. Amazon Web Services (AWS) - Cloud Computing Services. https://aws.amazon.com. [Online; accessed 28-Apr-2021].

[6] Amazon. 2021. AWS Lambda Pricing. https://aws.amazon.com/lambda/pricing/. [Online; accessed 1-Mar-2021].

[7] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter. 2018. Sprocket: A Serverless Video Processing Framework. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'18)*.

[8] AWS. 2021. AWS IoT Device SDK. https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html. [Online; accessed 28-Apr-2021].

[9] AWS. 2021. AWS IoT Greengrass Core SDK. https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html. [Online; accessed 28-Apr-2021].

[10] AWS. 2021. Lambda quotas. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html. [Online; accessed 28-Apr-2021].

[11] D. Becker, M. Schmidt, F. B. da Silva, S. Gül, C. Hellge, O. Sawade, and I. Radusch. 2018. Visual Object Tracking in A Parking Garage Using Compressed Domain Analysis. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys'18)*. 513–516.

[12] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. 2016. Simple Online and Realtime Tracking. In *2016 IEEE International Conference on Image Processing (ICIP'16)*. IEEE, 3464–3468.

[13] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor. 2019. Scaling Video Analytics on Constrained Edge Nodes. In *Proceedings of the 2nd Conference on Systems and Machine Learning (SysML'19)*.

[14] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz. 2019. Cirrus: A Serverless Framework for End-to-end ML Workflows. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'19)*. 13–24.

[15] J. Carreira and A. Zisserman. 2017. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*. 6299–6308.

[16] Z. Chen, K. Fan, S. Wang, L. Duan, W. Lin, and A. Kot. 2019. Lossy Intermediate Deep Learning Feature Compression and Evaluation. In *Proceedings of the 27th ACM International Conference on Multimedia (MM'19)*.

[17] CNBC. 2019. One billion surveillance cameras will be watching around the world in 2021, a new study says. https://www.cnbc.com/2019/12/06/one-billion-surveillance-cameras-will-be-watching-globally-in-2021.html. [Online; accessed 28-Apr-2021].

[18] Datadog. 2020. The State of Serverless. https://www.datadoghq.com/state-of-serverless/. [Online; accessed 2-September-2020].

[19] Amazon Dynamodb. 2021. Fast and flexible NoSQL database service for any scale. https://aws.amazon.com/dynamodb/. [Online; accessed 28-Apr-2021].

[20] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein. 2019. Cracking open the DNN black-box: Video Analytics with DNNs across the Camera-Cloud Boundary. In *ACM HotEdgeVideo*.

[21] Z. Fang, D. Hong, and R. K Gupta. 2019. Serving Deep Neural Networks at the Cloud Edge for Vision Applications on Mobile Platforms. In *Proceedings of the 10th ACM Multimedia Systems Conference (MMSys'19)*. 36–47.

[22] S. Fouladi, R. S. Wahby, B. Shacklett, K. Balasubramaniam, W. Zeng, et al. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *Proceeding of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*. 363–376.

[23] Azure Functions. 2021. More than just event-driven serverless compute. https://azure.microsoft.com/en-us/services/functions/. [Online; accessed 28-Apr-2021].

[24] S. Ge, S. Zhao, X. Gao, and J. Li. 2019. Fewer-Shots and Lower-Resolutions: Towards Ultrafast Face Recognition in the Wild. In *Proceedings of the 27th ACM International Conference on Multimedia (MM'19)*.

[25] Google. 2021. Cloud Functions. https://cloud.google.com/functions. [Online; accessed 28-Apr-2021].

[26] AWS IoT Greengrass. 2021. Bring local compute, messaging, data management, sync, and ML inference capabilities to edge devices. https://aws.amazon.com/lambda/edge/. [Online; accessed 28-Apr-2021].

[27] M. Hifi, M. Michrafy, and A. Sbihi. 2006. A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications* 33, 2-3 (2006), 271–285.

[28] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*.

[29] C. Hu, W. Bao, D. Wang, and F. Liu. 2019. Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge. In *IEEE Conference on Computer Communications (INFOCOM'19)*. IEEE, 1423–1431.

[30] C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC'18)*.

[31] intuVision Retail. 2021. Retail Video Analytics. https://www.intuvisiontech.com/intuvisionVA_solutions/intuvisionVA_retail [Online; accessed 28-Apr-2021].

[32] S. Jain, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. Gonzalez. 2019. Scaling Video Analytics Systems to Large Camera Deployments. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications (HotMobile'19)*. 9–14.

[33] A. H. Jiang, D. L-K Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger. 2018. Mainstream: Dynamic Stem-Sharing for Multi-Tenant Video Processing. In *2018 Usenix Annual Technical Conference (ATC'18)*.

[34] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'18)*. 253–266.

[35] J. Jiang, Y. Zhou, G. Ananthanarayanan, Y. Shu, and A. A. Chien. 2019. Networked Cameras Are the New Big Data Clusters. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo'19)*.

[36] E. Jonas, J. Schleier-Smith, V. Sreekanti, C. Tsai, A. Khandelwal, et al. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. *arXiv preprint arXiv:1902.03383* (2019).

[37] G. Lai, W. Chang, Y. Yang, and H. Liu. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 95–104.

[38] AWS Lambda. 2021. Run code without thinking about servers. Pay only for the compute time you consume. https://aws.amazon.com/lambda/. [Online; accessed 28-Apr-2021].

[39] AWS Lambda. 2021. Run your code closer to your users. https://aws.amazon.com/lambda/edge/. [Online; accessed 28-Apr-2021].

[40] Gil Levi and Tal Hassner. 2015. Age and Gender Classification Using Convolutional Neural Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops (CVPR'15)*.

[41] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh. 2019. Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo'19)*. 21–26.

[42] Q. Pu, S. Venkataraman, and I. Stoica. 2019. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. In *Proceeding of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*. 193–206.

[43] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen. 2018. Deepdecision: A mobile Deep Learning Framework for Edge Video Analytics. In *IEEE Conference on Computer Communications (INFOCOM'18)*.

[44] J. Redmon and A. Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767* (2018).

[45] Research and Markets. 2020. Surveillance Camera Market Outlook and Forecast 2020-2027: Revenues for Dome, Bullet, Box Style, PTZ, Thermal, and Other Cameras. https://www.globenewswire.com/news-release/2020/03/27/2007497/0/en/surveillance-camera-market-outlook-and-forecast-2020-2027-revenues-for-dome-bullet-box-style-ptz-thermal-and-other-cameras.html. [Online; accessed 28-Apr-2021].

[46] Amazon S3. 2021. Object storage built to store and retrieve any amount of data from anywhere. https://aws.amazon.com/s3/. [Online; accessed 28-Apr-2021].

[47] SamuiWebcam. 2020. Tropical Murphys Live Stream From Chaweng, Koh Samui, Thailand. https://www.youtube.com/watch?v=sbZNL98Z0GE. [Online; accessed 10-Mar-2020].

[48] SeeJH.com. 2021. Jackson Hole Wyoming USA Town Square Live Cam. https://www.youtube.com/watch?v=1EiC9bvVGnk. [Online; accessed 28-Apr-2021].

[49] H. Shojaei, T. Basten, M. Geilen, and A. Davoodi. 2013. A fast and scalable multidimensional multiple-choice knapsack heuristic. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18, 4 (2013), 1–32.

[50] G. Wang, Y. Wang, H. Zhang, R. Gu, and J. Hwang. 2019. Exploit the connectivity: Multi-object tracking with trackletnet. In *Proceedings of the 27th ACM International Conference on Multimedia (MM'19)*.

[51] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 Usenix Annual Technical Conference (ATC'18)*.

[52] M. Xu, T. Xu, Y. Liu, X. Liu, G. Huang, and F. X. Lin. 2019. Supporting Video Queries on Zero-Streaming Cameras. *arXiv preprint arXiv:1904.12342* (2019).

[53] Y. Xu, W. Yan, H. Sun, G. Yang, and J. Luo. 2019. CenterFace: Joint Face Detection and Alignment Using Face as Point. In *arXiv:1911.03599*.

[54] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A Lee. 2018. AWStream: Adaptive Wide-Area Streaming Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'18)*. 236–252.

[55] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *Proceeding of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*.

[56] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee. 2015. The Design and Implementation of a Wireless Video Surveillance System. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom'15)*.