Distream: Scaling Live Video Analytics with Workload-Adaptive Distributed Edge Intelligence

Xiao Zeng[†], Biyi Fang[†], Haichen Shen[‡], Mi Zhang[†]
[†]Michigan State University, [‡]Amazon Web Services

ABSTRACT

Video cameras have been deployed at scale today. Driven by the breakthrough in deep learning (DL), organizations that have deployed these cameras start to use DL-based techniques for live video analytics. Although existing systems aim to optimize live video analytics from a variety of perspectives, they are agnostic to the workload dynamics in real-world deployments. In this work, we present Distream, a distributed live video analytics system based on the smart camera-edge cluster architecture, that is able to adapt to the workload dynamics to achieve low-latency, high-throughput, and scalable live video analytics. The key behind the design of Distream is to adaptively balance the workloads across smart cameras and partition the workloads between cameras and the edge cluster. In doing so, Distream is able to fully utilize the compute resources at both ends to achieve optimized system performance. We evaluated Distream with 500 hours of distributed video streams from two real-world video datasets with a testbed that consists of 24 cameras and a 4-GPU edge cluster. Our results show that Distream consistently outperforms the status quo in terms of throughput, latency, and latency service level objective (SLO) miss rate.

CCS CONCEPTS

• **Computing methodologies → Distributed artificial intelligence**; **Computer vision**.

KEYWORDS

Distributed Deep Learning Systems, On-Device AI, Large-Scale Live Video Analytics, Workload Adaptive, Scheduling, Edge Computing

ACM Reference Format:
Xiao Zeng[†], Biyi Fang[†], Haichen Shen[‡], Mi Zhang[†]. 2020. Distream: Scaling Live Video Analytics with Workload-Adaptive Distributed Edge Intelligence. In *The 18th ACM Conference on Embedded Networked Sensor Systems (SenSys '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3384419.3430721

1 INTRODUCTION

Video cameras are ubiquitous. Today, cameras have been deployed at scale at places such as traffic intersections, university campuses, and grocery stores. Driven by the recent breakthrough in deep learning (DL) [26], organizations that have deployed these cameras start to use DL-based techniques for *live video analytics* [19, 22, 39]. Analyzing live videos streamed from these distributed cameras is the backbone of a wide range of applications such as traffic control and security surveillance. As many of these applications require producing analytics results in real-time, achieving *low-latency*, *high-throughput*, and *scalable* video stream processing is crucial [36].

Live video analytics systems require high-resolution cameras to capture high-quality visual data for analytics. As camera number scales up, these always-on cameras collectively generate hundreds of gigabytes of data every single second, making it infeasible to transmit such gigantic volume of data to data centers in the cloud for real-time processing due to insufficient network bandwidth and long transmission latency between cameras and the cloud [40].

The key to overcoming this bottleneck is to *move compute resources close to where data reside.* Status quo live video analytics systems hence stream camera feeds to local edge clusters for data analytics where much higher network bandwidth is provided [22, 25, 31, 39]. To move compute resources *even closer* to data sources, major video analytics providers (e.g., Avigilon, Hikvision, NVIDIA) are replacing traditional video cameras with "smart cameras". Equipped with onboard DL accelerators, these smart cameras are not only able to perform basic video processing tasks such as background subtraction and motion detection, but also capable of executing complicated compute-intensive DL-based pipelines to detect and recognize the objects and a variety of their attributes [15, 16, 23, 37]. Since each smart camera brings extra compute resources to process video streams generated by itself, this *smart camera-edge cluster architecture* is the key to enabling live video analytics at scale [9, 19, 23].

Motivation & Limitations of Status Quo. In real-world deployments, depending on what areas the cameras are covering, the number of objects of interest (e.g., people, vehicles, bikes) captured by each camera is different and can vary significantly over time. For example, for surveillance systems deployed on university campuses, a camera that covers the building entrance captures much larger numbers of people before and after classes than any other time; a camera that points at the emergency exit where people rarely visit has no objects of interest captured most of the time. As a consequence, the workload of recognizing the captured objects and their attributes produced at each camera is different and inherently dynamic over time.

As listed in Table 1, in recent years, live video analytics systems such as VideoStorm [39], Chameleon [22] and NoScope [25] have emerged. These systems enable efficient processing of a large number of camera streams, but are designed to process the streams within a centralized cluster.

With the emergence of smart cameras, recent systems start to leverage the compute resources inside smart cameras for distributed

SenSys '20, November 16–19, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-7590-0/20/11…$15.00
https://doi.org/10.1145/3384419.3430721

	Objective	Architecture	Video Analytics Pipeline Partition	Workload Adaptive	Cross-Camera Workload Balancing
VideoStorm [39]	Schedule video stream processing workloads with resource-accuracy tradeoff	Centralized	N/A	No	N/A
Chameleon [22]	Reduce the overhead of searching for the optimal resource-accuracy configuration	Centralized	N/A	No	N/A
NoScope [25]	Accelerate video analytics using resource-accuracy tradeoff	Centralized	N/A	No	N/A
FilterForward [9]	Identify important frames on smart cameras	Camera-Only	Fixed	No	No
VideoEdge [19]	Partition the video analytics pipeline across camera and cluster to optimize resource-accuracy tradeoff	Distributed	Dynamic	No	No
Distream	Enable workload adaptation for scalable, low-latency and high-throughput live video analytics without sacrificing accuracy	**Distributed**	**Dynamic**	**Yes**	**Yes**

Table 1: Comparison between Distream and status quo live video analytics systems.

live video analytics. FilterForward [9] proposes a camera-only solution which identifies important video frames and filter out unimportant ones directly on smart cameras. VideoEdge [19], on the other hand, proposes a fully distributed framework that partitions the video analytics pipeline across cameras and cluster with the objective to optimize the resource-accuracy tradeoff. While these systems aim to optimize live video analytics from a variety of perspectives, they are *agnostic* to the workload dynamics in real-world deployments described above, making them fall short in two situations: on one hand, failing to utilize the compute resources inside idle cameras could considerably jeopardize system throughput; on the other hand, failing to alleviate the workloads from cameras that are overloaded by bursty workloads could incur significantly high latency, causing the system not able to meet the latency service level objective (SLO) imposed by the live video analytics applications.

Overview of the Proposed Approach. In this paper, we present Distream – a distributed framework based on the smart camera-edge cluster architecture – that is able to *adapt* to the workload dynamics in real-world deployments to achieve low-latency, high-throughput, and scalable DL-based live video analytics. The underpinning principle behind the design of Distream is to adaptively balance the workloads across smart cameras as well as partition the workloads between cameras and the edge cluster. In doing so, Distream is able to fully utilize the compute resources at both ends to jointly maximize the system throughput and minimize the latency *without sacrificing the video analytics accuracy.*

The design of Distream involves three key challenges.

- **Challenge#1: Cross-Camera Workload Balancing**. One key obstacle to achieving high-throughput low-latency live video analytics is caused by the imbalanced workloads across cameras. Therefore, the first challenge lies in designing a scheme that balances the workloads across cameras. However, the cross-camera workload correlation, the heterogeneous onboard compute capacities of smart cameras, and the overhead of workload balancing altogether make designing such a scheme not trivial.
- **Challenge#2: Camera-Cluster Workload Partitioning**. Another key obstacle to achieving high-throughput low-latency live video analytics is caused by the imbalanced workloads between smart cameras and the edge cluster. To balance the workloads between cameras and edge cluster, the video analytics pipeline should be partitioned based on the workload ratio of the two sides. However, the possible options to partition the video analytics pipeline are quite limited in number, making workload partitioning between camera and edge cluster by nature *coarse-grained.* As a result, the partitioned video analytics pipeline may not match the workload ratio of the two sides.
- **Challenge#3: Adaptation to Workload Dynamics**. Given the dynamics of workloads in real-world deployments, the optimal solutions for cross-camera workload balancing and camera-cluster workload partitioning *vary over time.* Being able to adapt to such workload dynamics is a must for high-performance live video analytics systems. Designing such an adaptation scheme, however, is not trivial, as the optimal pipeline partitioning solution for each camera can be *different.* More importantly, since the workloads are jointly executed between cameras and edge cluster, for the whole system to achieve the best performance, the optimal pipeline partitioning solutions for all the cameras need to be *jointly* determined. This is a much more challenging problem compared to the single-pair workload partitioning problem tackled in the literature [10, 12].

To address the first challenge, Distream incorporates a cross-camera workload balancer that takes the cross-camera workload correlation, heterogeneous compute capacities of smart cameras, as well as the overhead of workload balancing into account, and formulates the task of cross-camera workload balancing as an optimization problem. In particular, the proposed cross-camera workload balancer incorporates a long-short term memory (LSTM)-based recurrent neural network which is able to enhance the performance of cross-camera workload balancing by predicting incoming workloads in the near future to avoid migrating workloads to cameras that are going to experience high workloads.

To address the second challenge, Distream incorporates a stochastic partitioning scheme that partitions the video analytics pipeline in a *stochastic* manner. In doing so, it provides much more partition flexibility and much finer partition granularity. As such, Distream is able to partition the pipeline to match the workload ratio of the smart camera and edge server.

To address the third challenge, Distream incorporates a workload adaption controller which triggers the cross-camera workload balancer when cross-camera workload imbalance is detected. Moreover, it formulates the task of jointly identifying the optimal pipeline partitioning solutions for all the cameras as an optimization problem with the objective to maximize the overall system throughput subject to the latency SLO imposed by the live video analytics applications.

System Implementation & Summary of Evaluation Results. We implemented Distream and deployed it on a self-developed testbed that consists of 24 smart cameras and a 4-GPU edge cluster. We evaluate Distream with 500 hours of distributed video streams from two real-world video datasets: one from six traffic cameras deployed in Jackson Hole, WY [7] for traffic monitoring application, and the other from 24 surveillance cameras deployed on a university campus for security surveillance application. We compare Distream against three baselines: Centralized, which processes all the workloads on the edge cluster; Camera-Only, which processes all the workloads on smart cameras; and VideoEdge [19]. Our results show that Distream consistently outperforms the baselines in terms of throughput, latency, and latency SLO miss rate by a large margin due to its workload adaptation schemes. Moreover, our scaling experiments show that Distream is able to scale up system throughput nearly linearly while maintaining a low latency with negligible overheads. Finally, we show that the workload adaptation techniques proposed in Distream could benefit existing live video analytics systems and enhance their performance as well.

Summary of Contributions. To the best of our knowledge, Distream represents the first distributed framework that enables workload-adaptive live video analytics under the smart camera-edge cluster architecture. It identifies a key performance bottleneck and contributes novel techniques that address the limitations of existing systems. We believe our work represents a significant step towards turning the envisioned large-scale live video analytics into reality.

2 BACKGROUND AND MOTIVATION

2.1 Live Video Analytics Pipeline

Modern live video analytics pipelines typically adopt a cascaded architecture which consists of a front-end object detector followed by a back-end task-specific module to perform a variety of analytics tasks on each of the detected object of interest within a video frame.

There are two types of object detectors that have been commonly used in existing live video analytics systems [22]. The first type is the CNN-based object detector (*e.g.*, YOLO [32], SSD [27] and Faster-RCNN [33]) which extracts and identifies all the objects of interest in a frame with one single inference. However, such an object detector has to be constantly extracting features from frames and performing inference even if there is no object of interest appearing in video streams. In many scenarios in real-world deployments, however, the objects of interest may only appear in video streams for short periods of time. In such case, a significant amount of compute resources is wasted. The second type of object detector is to first use a light-weight background subtractor [41] to extract the regions where objects of interest reside from the frame. It sends these regions to a classifier to identify the object within each region. As such, it produces objects of interest only when they appear in the frame. While Distream is a generic framework which supports both types of object detectors, in this work, we focus on the background subtraction-based detector to illustrate our ideas.

The task-specific module in general can be represented as a directed acyclic graph (DAG). In this work, we use attribute recognition as a concrete example of the task-specific module to illustrate our ideas. Specifically, each vertex in the DAG represents a DNN classifier that recognizes a particular attribute of the object; each directed edge represents a data flow from one classifier to another. For example, the task-specific DAG in Figure 1 consists of three branches. Based on the type of the detected object (vehicle, person, or others), the task-specific module selects one of the three branches, each of which employs a cascaded sequence of classifiers to further identify the attributes of the object[1].

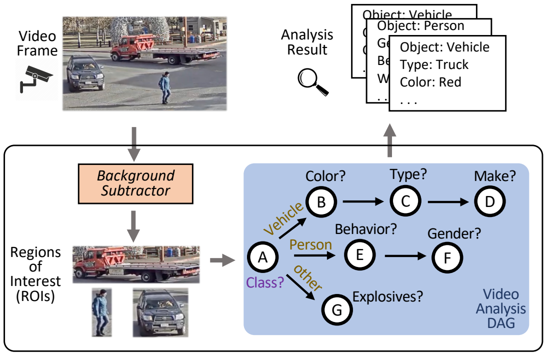


Figure 1: Live video analytics pipeline used in Distream.

Based on the pipeline illustrated in Figure 1, going through each classifier within the DAG is regarded as an individual *workload*[2]. Therefore, identifying an object of interest and its attributes within a video frame produces multiple workloads to be processed by the live video analytics system.

2.2 Workload Dynamics in Real-world Deployments

To illustrate the workload dynamics in real-world deployments, we collected a dataset from 24 surveillance cameras deployed on a university campus (§5.1). Given the limited space, we pick a representative video clip to make our points. Specifically, Figure 2 shows the workloads generated by four surveillance cameras deployed at a university building on a weekday between 12pm to 12:30pm. Among them, CAM1 monitors a square next to the building entrance; CAM2 monitors a sideway outside the building; CAM3 and CAM4 cover two different corridors inside the building.

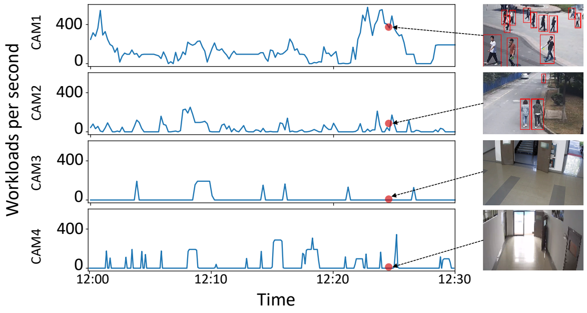


Figure 2: Workload dynamics in real-world deployment.

[1] Although we adopted the design choice of treating each task independently, Distream is flexible to support multi-task learning where a DNN inference produces multiple results. It should be noted that not all the tasks can be combined into a single multi-task DNN model. Thus our DAG formulation is general enough to support all the cases.

[2] We did not include capturing images and background extraction as workloads mainly because these steps consume much less computation compared to DNN-based inference and thus can be executed locally fast enough without offloading.

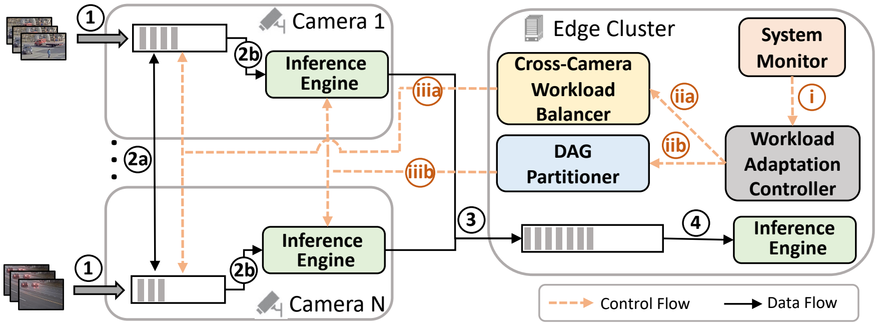


Figure 3: System architecture of Distream.

As illustrated, depending on the views captured by the cameras, the workloads of real-world video streams have three key characteristics. (i) *The workloads are different across cameras*: for CAM1 that monitors busy areas, more workloads are generated as pedestrians pass by more often; for CAM3 that monitors a less busy indoor corridor, much less workloads are generated sporadically and it has no workloads most of the time. (ii) *The workload generated at each camera is dynamic over time*: this is obvious because the content captured by each camera is changing over time. (iii) *The amount of workloads generated at each camera can vary significantly*: the workload difference between bursty and non-bursty durations can be more than 1000×.

2.3 Need for Workload Adaptation

Existing live video analytics systems based on the smart camera-edge cluster architecture, however, are agnostic to the real-world workload dynamics illustrated above. To demonstrate how existing systems fall short under such workload dynamics, we compare the system performance between Distream that is workload-adaptive and a workload-agnostic baseline that allocates compute resource proportional to the compute capacity. We use the same dataset in §2.2 and a testbed that consists of four smart cameras (2 Jetson-TX1 and 2 Jetson-TX2) and 1-GPU edge cluster (§4) to conduct our experiments.

Figure 4 illustrates our comparison results. As shown, Distream achieves significant improvement in terms of throughput and latency compared to the workload-agnostic scheme. Specifically, the median and peak throughputs of Distream are 302 IPS and 562 IPS respectively, which is 1.4× and 2.3× improvement over the workload-agnostic baseline, which only achieves 213 IPS median and 240 IPS peak throughputs. Distream also achieves 0.23s median and 3.92s maximum latency, which reduces medium and maximum latency by 57× and 40.7× compared to the workload-agnostic scheme which achieves 13.1s median latency and 159.73s maximum latency.

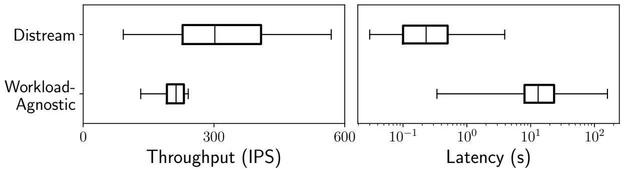


Figure 4: Comparison of throughput (left) and latency (right) between workload-agnostic and workload-adaptive (ours) schemes.

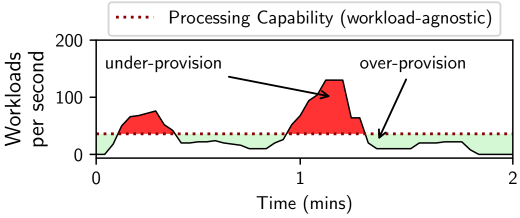


Figure 5: Illustration on where the workload-agnostic scheme falls short under dynamic workloads in real-world deployments.

To understand why the workload-agnostic scheme falls short, we use one smart camera as an example and plot its workloads generated over a two mins video clip in Figure 5. Specifically, the black solid line depicts the generated workloads over time; the red dashed line depicts the camera's processing capability (i.e., the maximum number of workloads it can process per unit time) based on the workload-agnostic scheme. As shown, under the workload-agnostic scheme, although the workload generated at the smart camera is dynamic over time, the processing capability of the camera stays the same and does not adapt to the workload variation. Consequently, when the generated workload exceeds its processing capability, the camera experiences *under-provisioning* (red region) so that it fails to process the workloads in time; when the workload is below its processing capability, the camera becomes *over-provisioning* (green region) and thus wastes its compute resources that could have been used to process workloads generated from other cameras. These observations altogether highlight the need for workload adaptation to avoid the occurrence of either under-provisioning or over-provisioning, which motivates the design of Distream.

3 DISTREAM DESIGN

3.1 Overall Architecture

Figure 3 illustrates the overall architecture of Distream. As shown, Distream is designed as a distributed framework that spans across smart cameras and the edge cluster. In the data plane, as soon as the smart camera captures a video frame, it runs the onboard background subtractor to extract regions of interest (ROIs) within the frame and appends them one by one into a local queue (①).

Each ROI in the queue is either offloaded to another camera ((2a)) or partially processed inside the local *inference engine* according to its partitioned DAG whose partition point is determined by the *DAG partitioner* ((2b)). The result of the partially-processed DAG is then transferred to the edge cluster and buffered in a queue ((3)), which will be processed by the *inference engine* inside the edge cluster ((4)) to complete the remaining unprocessed part of the DAG. All the inference results from the cameras are gathered at the edge cluster. In the control plane, the *system monitor* continuously monitors the workloads at each camera and edge cluster, and sends the collected workload information to the *workload adaptation controller* ((i)). Once the workload imbalance across cameras is detected, the *workload adaptation controller* triggers the *cross-camera workload balancer* ((iia)) to balance the workloads across cameras ((iiia)). Meanwhile, based on the current workloads at each camera and edge cluster, the *workload adaptation controller* adaptively identifies the optimal DAG partition point for each camera that balances the workload between each camera and the edge cluster. Such optimal DAG partition points are sent to the *DAG partitioner* ((iib)) for DAG partitioning, and the corresponding DAG partition result is sent to the *inference engine* at each camera ((iiib)).

In the following, we describe the three key components of Distream: *cross-camera workload balancer* (§3.2), *DAG partitioner* (§3.3), and *workload adaptation controller* (§3.4) in detail.

3.2 Balancing the Workloads across Smart Cameras

The first key component of Distream is the *cross-camera workload balancer*, which balances the workloads at the level of the extracted regions of interest (ROIs) across cameras by migrating part of the workloads from heavily loaded cameras to idle or lightly loaded ones (Figure 6). To achieve this, the *cross-camera workload balancer* needs to accommodate three key considerations:

- **Consideration#1: cross-camera workload correlation**. As also being observed by many existing works [21–23], workloads on nearby cameras may exhibit strong correlations due to mobility of objects of interest: a camera may have high workload within a short time period if its nearby cameras are experiencing high workloads. The *cross-camera workload balancer* needs to take such correlations into account to avoid migrating workloads to cameras that are going to experience high workloads.
- **Consideration#2: heterogeneous compute capacities**. Smart cameras may have heterogeneous onboard compute capacities caused by different generations of compute hardware. To prevent any camera from becoming the bottleneck, the *cross-camera workload balancer* needs to balance the workloads proportional to each camera's compute capacity.
- **Consideration#3: workload balancing overheads**. In practice, migrating workloads across cameras incurs overheads. The *cross-camera workload balancer* needs to take such overheads into account such that the overheads do not overshadow the benefits brought by load balancing.

The *cross-camera workload balancer* takes these three considerations into account, and formulates the task of cross-camera workload balancing as an optimization problem.

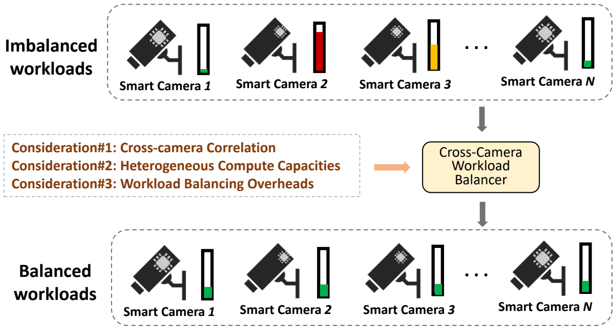

Imbalanced workloads
Smart Camera 1
Smart Camera 2
Smart Camera 3
Smart Camera N
Consideration#1: Cross-camera Correlation
Consideration#2: Heterogeneous Compute Capacities
Consideration#3: Workload Balancing Overheads
Cross-Camera Workload Balancer
Balanced workloads
Smart Camera 1
Smart Camera 2
Smart Camera 3
Smart Camera N

Figure 6: Overview of the cross-camera workload balancer.

Formally, let N denote the total number of smart cameras, w_i denote the workload of camera i before load balancing, and let $\sum_j x_{ji}$ and $\sum_j x_{ij}$ denote the workloads that are moved into and moved out of camera i respectively. Therefore, the workload of camera i after load balancing can be calculated as $u_i = (w_i + \sum_j x_{ji} - \sum_j x_{ij})$. And the goal of the *cross-camera workload balancer* is to minimize v, the workload imbalance index across smart cameras after workload balancing, which can be calculated as

$$v = (\frac{u_{max}}{\bar{u}} - 1) \tag{1}$$

where u_{max} and $\bar{u}$ represent the maximum workload and average workload of all cameras respectively. By minimizing the workload imbalance index, the *cross-camera workload balancer* is able to achieve a good balance between workload balancing efficiency and fairness without incurring thrashing.

To accommodate **Consideration#1**, we take the cross-camera workload correlation into account by incorporating a workload predictor to predict the future workload at each camera (not from migration) based on their current workloads and the cross-camera workload correlations. Specifically, we employ long-short term memory (LSTM)-based recurrent neural network [30] as the workload predictor because it shows better long-term forecasting ability in predicting time series data compared to other methods [29]. The predicted workload $\hat{w}_i$ at camera i output by the workload predictor is added to u_i as an adjustment to obtain a more accurate future workload estimate as

$$u_i = w_i + \sum_j x_{ji} - \sum_j x_{ij} + \hat{w}_i \tag{2}$$

To accommodate **Consideration#2**, we take the heterogeneous compute capacities of smart cameras into account by multiplying u_i with a normalization factor $a_i = C_i / \bar{C}$ where C_i is the compute capacity of camera i and $\bar{C}$ is the average compute capacity of all the smart cameras. As a result, the workload at each camera after load balancing gets normalized as $u_i' = u_i * a_i$ and the workload imbalance index v gets normalized as

$$v' = (\frac{u_{max}'}{\bar{u}'} - 1) \tag{3}$$

To accommodate **Consideration#3**, a triggering threshold β is added into our optimization objective. The cross-camera workload balancing is not triggered when v' is below β.

Putting all the pieces together, our cross-camera workload balancing scheme can be formulated as

$$\min_{x_{ij} \geq 0} \quad \max(v' - \beta, 0) \tag{4a}$$

$$s.t. \quad u_i = w_i + \sum_j x_{ji} - \sum_j x_{ij} + \hat{w}_i \tag{4b}$$

$$a_i = C_i / \bar{C} \tag{4c}$$

$$u_i' = u_i * a_i \tag{4d}$$

$$v' = (\frac{u'_{max}}{\bar{u}'} - 1) \tag{4e}$$

Solving the above nonlinear optimization problem is computationally hard. To enable real-time cross-camera workload balancing, we utilize a heuristic to efficiently solve the optimization problem. Specifically, our heuristic starts with all $x_{ij} = 0$ and increases x_{ij} iteratively. In each iteration, we select cameras that have the largest and the smallest workloads to form a migration pair. Then we increase the migrated workload x_{ij} by Δx (i.e., $x_{ij} = x_{ij} + \Delta x$), where $\Delta x = 1\% * u_i'$. We repeat such iteration until v' is below a threshold or v' does not improve between two consecutive iterations.

As shown in §5.6, such heuristic is able to solve the optimization problem in an efficient manner and incurs negligible overheads.

3.3 Partitioning the Workload between Smart Cameras and Edge Cluster

The second key component of Distream is the *DAG partitioner*, which partitions the workloads between smart cameras and edge cluster at the DAG level. To achieve this, the *DAG partitioner* needs to accommodate two key considerations:

- **Consideration#1: DAG is conditionally executed**. As the contents of video streams are changing, the execution flow in DAG is changing correspondingly. Take the DAG in Figure 1 as an example: if a 'Vehicle' is detected, the upper path of the DAG (i.e., the 'Color', 'Type' and 'Make' classifiers) will be executed; if a 'Person' is detected, the middle path of the DAG (i.e., 'Behavior' and 'Gender' classifiers) will be executed. However, to identify the optimal DAG partition point, the *DAG partitioner* needs to consider all the possible partition points from all the possible execution paths.
- **Consideration#2: DAG Partitioning is by nature coarse-grained**. Ideally, to balance the workloads between cameras and the edge cluster, the DAG partition point should be set based on the workload ratio of the two sides. However, the possible partition points in a DAG are the vertices in the DAG, which are discrete and limited in number. This makes DAG partitioning by nature coarse-grained. As a result, the DAG partition point may not match the workload ratio of the two sides.

The *DAG partitioner* takes these two considerations into account, and designs a two-step scheme to partition the DAG.

Step#1: Full DAG Profiling. In our first step, we follow the topological order of the DAG to extract all the possible execution paths in the DAG. As an example, Figure 7 shows three execution paths extracted from the DAG in Figure 1. For each extracted execution path, we then profile the inference cost of each classifier, and label the normalized accumulated inference cost on top of each classifier. For example, in Figure 7, the normalized accumulated inference cost of executing the 'Vehicle' path up to classifier B is 35%.

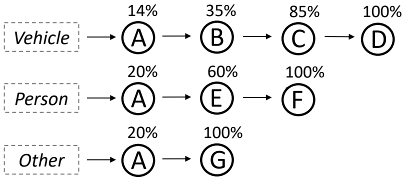


Figure 7: Full DAG profiling. The percentage on top of each vertex is the normalized accumulated inference cost.

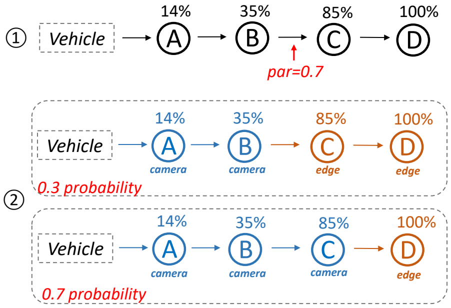


Figure 8: Stochastic DAG partition. Vertexes in blue color are classifiers allocated to run at the camera, and vertexes in brown color are classifiers allocated to run at the edge cluster.

Step#2: Stochastic DAG Partitioning. In our second step, we propose a stochastic DAG partitioning scheme to partition the DAG in a fine-grained manner such that the partitioned DAG could better match the workload ratio of the camera and the edge cluster. Specifically, for each execution path obtained in Step#1, the goal of our stochastic DAG partitioning scheme is to find a *stochastic path execution plan* such that the expected normalized accumulated inference cost of executing the path matches the optimal DAG partition point $par \in [0, 1]$ provided by the *workload adaptation controller* (§3.4).

Take the 'Vehicle' path in Figure 8 as an illustrative example. For the profiled 'Vehicle' path, assume we need to achieve $par = 0.7$ workload partitioning (i.e., 70% workload allocated to the camera and 30% workload allocated to the edge cluster) determined by the *workload adaptation controller.* Our stochastic DAG partitioning scheme first identifies two vertexes (vertex B, vertex C) along the 'Vehicle' path such that $par = 0.7$ falls between their normalized accumulated inference cost (Figure 8 (1)). Given the normalized accumulated inference costs of these two vertexes ($Cost_B = 0.35$ for vertex B, $Cost_C = 0.85$ for vertex C), our stochastic DAG partitioning scheme then determines the probabilities for partitioning the 'Vehicle' path at vertex B (P_B) and vertex C (P_C) respectively as:

$$\begin{aligned} P_B &= (par - Cost_C)/(Cost_B - Cost_C) \\ P_C &= 1 - P_B \end{aligned} \tag{5}$$

In this example, the solutions are $P_B = 0.3$ and $P_C = 0.7$. Therefore, the *stochastic path execution plan* is to partition the path at vertex B with a probability of 0.3 and to partition the path at vertex C with a probability of 0.7 (Figure 8 (2)). As such, the expected normalized accumulated inference cost of executing the path matches the optimal DAG partition point *par* $(0.3 * 35\% + 0.7 * 85\% = 0.7)$.

Compared to the discrete DAG partitioning approach, our stochastic DAG partitioning scheme provides much more partition flexibility and granularity. With such flexibility and granularity, Distream is able to better partition the DAG between smart cameras and edge cluster to match the workload ratio of two sides.

3.4 Workload Adaptation Controller

The third key component of Distream is the *workload adaptation controller*, which is the core to achieve our workload adaptation objective. The *workload adaptation controller* performs two tasks. First, when the workload imbalance index across smart cameras is greater than a threshold β, the *workload adaptation controller* triggers the *cross-camera workload balancer* to balance the workloads across the cameras as described in §3.2. Second, the *workload adaptation controller* follows a camera-cluster workload partition schedule to periodically (at interval γ) update the optimal DAG partition point for each camera to balance the workloads between each camera and edge cluster (Figure 9). Such optimal partition points are sent to the *DAG partitioner* for DAG partitioning as described in §3.3.

The DAG partition points serve as knobs to control the DAG execution across cameras and edge cluster, and thus control the workload split between them. To identify the optimal DAG partition points, the *workload adaptation controller* needs to accommodate two key considerations:

- **Consideration#1: the optimal DAG partition point is different for each camera**. The smart cameras have heterogeneous compute capacities and the workload at each camera can be different due to the discretization of cross-camera workload balancing. This requires the *workload adaptation controller* to identify the optimal DAG partition point for each camera.
- **Consideration#2: the optimal DAG partition points of all cameras need to be determined jointly**. Because the workloads are partitioned between cameras and edge cluster, the system performance (throughput and latency) is *jointly* determined by all cameras and edge cluster. Therefore, the optimal DAG partition points for all cameras are linked together and need to be jointly adjusted as well.

The *workload adaptation controller* takes these two considerations into account, and formulates the task of identifying optimal DAG partition points as an optimization problem.

Specifically, our optimization problem aims to jointly identify the optimal DAG partition points for all the cameras with the objective to maximize the overall system throughput subject to the latency service level objective (SLO) imposed by the live video analytics applications. Below, we describe how we model the throughput and the latency followed by the complete optimization formulation.

Throughput. As a DAG is partitioned into two parts and executed on a camera and edge cluster respectively, the throughput of the entire live video analytics system is the sum of the throughput of cameras and the throughput of the edge cluster.

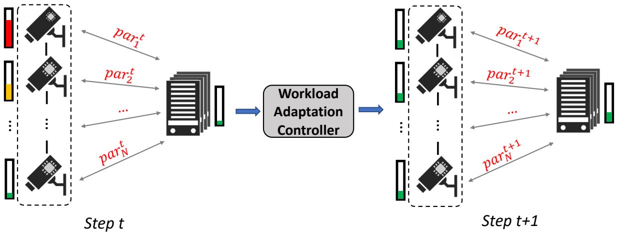


Figure 9: Overview of camera-cluster workload partition in workload adaptation controller. The optimal DAG partition points of all cameras are jointly determined

Formally, let N denote the number of cameras in the system, par_i denote the DAG partition point for camera i, and $TP^{cam_i}(par_i)$ denote the throughput of camera i given DAG partition point par_i, and TP^{edge} denote the throughput at the edge cluster which depends on the all partition points $\{par_i\}$. $TP^{cam_i}(par_i)$ and TP^{edge} can be accurately measured by monitoring the number of workloads processed per time unit at camera i and edge cluster respectively. Therefore, the throughput of N cameras TP^{cams}, the throughput of the edge cluster TP^{edge}, and the throughput of the entire system TP^{system} can be computed as:

$$TP^{cams} = \sum_{i=1}^{N} TP^{cam_i}(par_i) \tag{6a}$$

$$TP^{edge} = TP^{edge}(par_1, par_2, ..., par_N) \tag{6b}$$

$$TP^{system} = TP^{cams} + TP^{edge} \tag{6c}$$

Latency. The latency of a live video analytics system is defined as the time elapsed between receiving a workload and producing the inference result of the workload. Given that, the latency is the sum of two parts: 1) the workload queuing time, and 2) the workload processing time for producing the inference result. Because workloads are partitioned across camera and edge, the latency of two sides is thus computed separately. As such, the latency at camera i and at edge cluster are computed as:

$$L^{cam_i} = \frac{w_i}{TPcam_i} + \frac{1}{TPcam_i} \tag{7a}$$

$$L^{edge} = \frac{w_{edge}}{TPedge} + \frac{1}{TPedge} \tag{7b}$$

where w_i is the number of pending workloads to be processed at camera i, and w_{edge} is the number of pending workloads to be processed at the edge cluster.

Complete Optimization Formulation. Putting all the pieces together, the complete optimization problem can be formulated as[3]:

$$\max_{par_i \forall i} \quad TP^{system} \tag{8a}$$

$$s.t. \quad L^{cam_i} \leq L_{Max}, \forall i \tag{8b}$$

$$L^{edge} \leq L_{Max} \tag{8c}$$

where L_{Max} is the latency SLO imposed by the specific application.

[3] Distream operates at DAG node level when partitioning workloads between cameras and edge cluster. Defining the optimization objective in terms of DAG node is able to reflect the performance gain in a more straightforward manner.

As shown, since L^{cam_i} and L^{edge} are determined by TP^{cam_i}, w_i, TP^{edge} and w_{edge}, which are determined by par_i, the workloads received at cameras and the edge cluster are not independent but negatively correlated with each other. This tradeoff illustrates the significance of identifying the optimal DAG partition points to balance workloads between cameras and the edge cluster to maximize the throughput under the latency SLO.

The optimization problem formulated above is non-convex. Instead of exhaustively searching over all possible partition points for all cameras, we adopt a heuristic to solve the optimization problem efficiently. Specifically, our heuristic is iterative and each iteration has two phases. In the first phase, it starts with the objective of finding the preliminary DAG partition points that maximize the throughput without considering the latency SLO. In the second phase, we iteratively target the bottleneck camera with the largest latency and adjust its preliminary DAG partition point to meet the latency SLO. In practice, our heuristic is highly effective with negligible overheads even if the number of cameras scales up (§5.6).

4 SYSTEM IMPLEMENTATION

We implemented Distream using about 2500 lines of Golang code and 500 lines of python code. In this section, we provide details on how the key parts of Distream were implemented.

Testbed. We followed the design choice that is widely adopted by existing live video analytics systems in real-world deployments to develop our own testbed to evaluate the performance of Distream. Specifically, our testbed consists of 24 smart cameras and a single edge cluster. Among the 24 cameras, 18 were prototyped using Nvidia Jetson TX1 [3] while the other six were prototyped using Nvidia Jetson TX2 [6]. Both Jetson TX1 and TX2 are designed for embedded systems with onboard DL workloads[4]. Jetson TX2 is an upgraded version of Jetson TX1 with larger compute capacity. For the edge cluster, we use a desktop server equipped with a cluster of 4 Nvidia Titan X GPUs [1]. We followed the standard configuration of existing IP video surveillance systems to connect and set up the network bandwidth between smart cameras and edge cluster [5]. Specifically, all the 24 smart cameras and the edge cluster are wire-powered and interconnected through a single switch (D-Link DGS-1510-28X [2]) to form a local network. Each smart camera has a Fast Ethernet link (10/100 Mbps) to the switch, and the bandwidth from the switch to the edge cluster is 10Gbps. By default, we set the camera bandwidth to 10Mbps. We also evaluate the system performance under 50Mbps.

LSTM Workload Predictor. We use a two-layer LSTM with 256 neurons in each layer as our workload predictor. Specifically, the workload predictor takes the arrived workloads of all cameras as input and predicts the upcoming workloads of each camera in the next second. We observed that the amount of workloads generated at each camera is highly relevant to the time period in a day. Thus we divide a day into six periods as 00:00-04:00, 04:00-8:00, 8:00-12:00, 12:00-16:00, 16:00-20:00 and 20:00-24:00. For each period, we train a specialized LSTM using workloads observed from that period. We find that using six specialized LSTMs reduces 10.2% mean square error compared to using a universal LSTM. Since running a specialized LSTM is fast on CPU (less than 1ms for one step), and it only makes predictions each second, our LSTM workload predictor incurs negligible overheads. Although the predictor could potentially benefit from periodic re-training given the dynamics of workloads over time, it is not the focus of this work.

Video Analytics Pipeline. We use OpenCV 3.2.0 to read video streams and implemented the Gaussian mixture model-based background subtraction method in [4] to extract regions of interest (ROIs) with length of history set to 500 and threshold set to 60. It maintains an estimate of background image and uses subtraction operation to extract foreground regions, followed by blob detection operations to extract ROIs. The number of extracted ROIs per frame varies depending on the nature of the scenes captured by the cameras, ranging from 0 to 30. For *inference engine*, we designed an efficient DNN model based on MobileNetV2 [34] for each classifier, and we are able to compress over 90% of model weights using pruning [28] and knowledge distillation [17] without accuracy loss. Given that each DNN model has only about 2MB memory footprint, the *inference engine* is able to load all the DNN models into a single GPU, avoiding model loading or switching time. Similar to [35], we adopt batched inference [11] to improve video analytics throughput. We profiled the inference latency with batch size of 1, 8, 16, 32 and 64 respectively and set the batch size to 8 at the camera side and 32 at the edge cluster given its better performance tradeoff.

Scheduling. We implemented a monitor proxy at each camera and edge cluster to keep track of the runtime information including workload status, network bandwidth, throughput and latency. The proxy periodically reports these information to the *system monitor* in the edge cluster every 50ms. When performing the cross-camera workload balancing, the ROI is encapsulated into the migrating workload because the target camera does not have the video frame from the source camera.

Communication. Distream uses ZeroMQ [8] for high-speed low-cost inter-process communication among cameras. For remote communication between cameras and edge cluster, we implemented a low-cost remote procedure call (RPC) to transfer control data. Specifically, the cross-camera workload balancing instructions and DAG partition points are transferred to cameras immediately after they are generated by the *cross-camera workload balancer* and the *DAG partitioner* respectively. When performing cross-camera workload balancing and camera-cluster workload partitioning, we use batched RPCs for communication between two entities to amortize the communication overhead. The maximum RPC batch size is 20.

5 EVALUATION

In this section, we evaluate the performance of Distream with the aim to answer the following questions:

- **Q1 (§5.2)**: *Does Distream outperform status quo live video analytics systems? If so, what are the reasons?*
- **Q2 (§5.3)**: *How effective is each core technique incorporated in the design of Distream?*
- **Q3 (§5.4)**: *How is the performance of Distream affected by the system hyper-parameters and network bandwidth?*
- **Q4 (§5.5)**: *Does Distream scale well?*

[4] As AI chipsets evolve, we conjecture that AI hardware with such compute capacity will be widely deployed inside various edge devices such as smart cameras.

- **Q5 (§5.6)**: *How much overheads does Distream incur?*
- **Q6 (§5.7)**: *Do the techniques proposed in Distream also benefit status quo live video analytics systems?*

5.1 Experimental Methodology

Applications. We use Traffic Monitoring and Campus Surveillance as two representative applications to evaluate Distream. For each application, we use a representative real-world multi-camera dataset to benchmark the performance.

- **Application#1: Traffic Monitoring.** For this application, we use publicly available live video streams from six traffic cameras deployed at different traffic intersections and roads in Jackson Hole, WY [7]. These six live video streams have dynamic contents, reflecting diverse traffic conditions in the city across space and time. To obtain a representative dataset, we sampled 200 5-minute video clips across 48 hours of video streams from each of the six cameras at a frame rate of 15 FPS and 1280 × 720P frame resolution. The sampled video clips cover diverse traffic conditions at different time of the day including rush hours, light traffic time, and night time.
- **Application#2: Campus Surveillance.** Due to lack of publicly available large-scale datasets, we self-collected a dataset from 24 surveillance cameras deployed on a university campus. These cameras cover diverse areas of the campus, including indoor areas such as dining halls, cafeterias, and hallways as well as outdoor areas such as university entrances, building entrances, squares, streets, and traffic intersections. For security purposes, some of the surveillance cameras are particularly covering areas where people rarely visit. Similar to the Traffic Monitoring application, we sampled 200 5-minute-long video clips across 48 hours of video streams from each of 24 cameras at a frame rate of 15 FPS and 1280 × 720P frame resolution to obtain a representative dataset.

Baselines. We compare Distream against three baselines that cover all three types of architectures listed in Table 1.

- **Centralized**. This baseline represents the centralized approach adopted by a number of existing live video analytics systems such as VideoStorm [39], Chameleon [22] and NoScope [25] where video streams are sent to edge cluster for processing.
- **Camera-Only**. This baseline represents the approach at the other end of the spectrum where video streams are only processed locally on smart cameras (e.g., FilterForward [9]).
- **VideoEdge-Lossless (VideoEdge-L)**. VideoEdge [19] is the status quo live video analytics system that utilizes compute resources across cameras and cluster to process video streams in a distributed manner. Different from Distream, VideoEdge is workload-agnostic and aims to achieve optimized resource-accuracy tradeoff, which may trade for resources with loss of accuracy. Since Distream does not sacrifice accuracy, for a fair comparison, we implemented VideoEdge with the resource-accuracy tradeoff disabled, which we refer to as VideoEdge-Lossless (VideoEdge-L) and use it as our third baseline. As VideoEdge is workload agnostic and the only prior knowledge given is the compute cost of each classifier in the DAG as well as the compute capacity ratio of cameras and cluster, we partitioned the DAG and place the partitions according to the camera-cluster compute ratio to implement VideoEdge-L.

Evaluation Metrics. We use three metrics to evaluate the performance of Distream and the baselines.

- **Throughput**. Live video analytics systems need to process streaming video frames in a continuous manner, and high-throughput processing is essential to keeping up with the incoming video streams. In our case, workloads are inferences involved in the DAG. Thus, we use the number of inferences processed per second (IPS) to measure the throughput.
- **Latency**. Live video analytics applications require producing analytics results within a short period of time. We thus use latency which is defined as the elapsed time from when the workload is generated to when the inference result of the workload is produced as our second metric to measure the system's responsiveness. As such, latency can be calculated as the sum of network latency, workload queuing time, and workload processing latency.
- **Latency Service Level Objective (SLO) Miss Rate**. Lastly, we measure the latency service level objective (SLO) miss rate, which quantifies the percent of the workloads that do not meet the latency requirement set by a live video analytics application. In this work, we set the latency SLO to 3 seconds, which is a reasonable requirement for live video analytics systems.

5.2 Overall Performance

We begin with comparing the overall performance of Distream and baselines on both applications. Since the Traffic Monitoring dataset involves six video streams, we use six smart cameras (4 TX1 and 2 TX2) with each allocated to one video stream and incorporate one GPU in the edge cluster. We will scale to 24 smart cameras and 4-GPU edge cluster when evaluating the scaling performance of Distream in §5.5.

Throughput and Latency. Table 2 and Table 3 list the throughput and latency of Distream and baselines on the Campus Surveillance and Traffic Monitoring application respectively. Specifically, we report the average, 50th (median), 75th, and 99th percentile of throughput and latency performance gain over Camera-Only.

We have three major observations. First, Distream has achieved higher throughput than the baselines. Specifically, it achieves 2.9× and 1.6× average throughput gain on two applications, while the best baseline (VideoEdge-L) only achieves 2.1× and 1.2×. Second, Distream achieves higher peak throughput (99th) than the baselines. The higher peak throughput indicates that Distream is able to better utilize the distributed compute resources to survive through the workload bursts such as the ones illustrated in Figure 2. Third, Distream archives significant latency reduction compared to the baselines. In particular, Distream achieves 128.2× and 184× average latency reduction, while VideoEdge-L only achieves 1.5×.

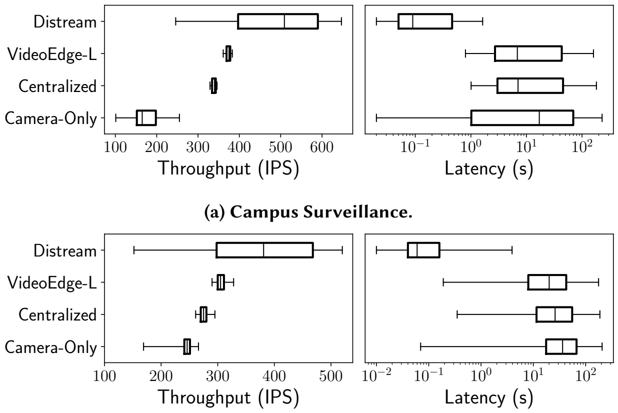
Figure 10 provides a more comprehensive view of the comparison by plotting the full distribution of throughput and latency, including the 1st, 25th, 50th (median), 75th and 99th percentiles in the box plot. In terms of throughput, Distream has a much wider throughput range, which indicates that Distream is able to handle

	Throughput				Latency			
	avg	med	75th	99th	avg	med	75th	99th
Distream	**2.9×**	**3.1×**	**2.9×**	**2.5×**	**128.2×**	**189.3×**	**147.9×**	**112×**
VideoEdge-L	2.1×	2.2×	1.9×	1.5×	1.5×	2.4×	1.6×	1.4×
Centralized	1.9×	2.0×	1.7×	1.3×	1.4×	2.2×	1.5×	1.2×

Table 2: Performance gain over Camera-Only (Campus Surveillance).

	Throughput				Latency			
	avg	med	75th	99th	avg	med	75th	99th
Distream	**1.6×**	**1.6×**	**1.9×**	**2.0×**	**184×**	**604.3×**	**446.8×**	**52.6×**
VideoEdge-L	1.2×	1.2×	1.2×	1.2×	1.5×	1.8×	1.5×	1.2×
Centralized	1.1×	1.1×	1.1×	1.1×	1.2×	1.4×	1.2×	1.1×

Table 3: Performance gain over Camera-Only (Traffic Monitoring).



(a) Campus Surveillance.

(b) Traffic Monitoring.

Figure 10: A closer look at the throughput and latency distribution.

a much wider range of amounts of workloads than the baselines. In terms of latency, the average latency of Distream for the Campus Surveillance and Traffic Monitoring application is 0.34s and 0.27s, which is much less than the baselines, which ranges from 28.54s to 43.93s for Campus Surveillance and 9.98s to 49.68s for Traffic Monitoring. This result indicates that the throughput improvement in Distream does not come at the cost of sacrificing its latency.

Latency SLO Miss Rate. Figure 11 compares the latency SLO miss rate of Distream against baselines. As shown, Distream outperforms the baselines by a large margin. In particular, Distream is able to achieve a near-zero latency SLO miss rate (0.7% and 1.5%) on the Campus Surveillance and Traffic Monitoring application respectively, while the baselines have much higher miss rates (at least 60.7% Campus Surveillance and 93.8% for Traffic Monitoring).

Why Distream Outperforms the Baselines? Distream outperforms both Centralized and Camera-Only because Distream is able to leverage the compute resources at both cameras and edge cluster sides while Centralized and Camera-Only could not. For VideoEdge-L, Distream is able to achieve better performance for the following two reasons:

Reason#1: Higher Resource Utilization. In terms of throughput, Distream outperforms the baselines because it is able to utilize

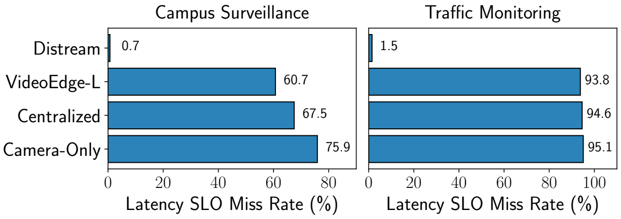


Figure 11: Latency SLO miss rate comparison.

the idle compute resources at both cameras and the edge cluster to process the workloads to enhance the throughput. To see this, we continuously track the workload imbalance index across smart cameras and the edge cluster. As shown in Figure 12 (a), Distream is able to achieve a near-zero workload imbalance for about 60% of the time. In contrast, VideoEdge-L is experiencing significant workload imbalance (more than 200% imbalance index) for 90% of the time for being agnostic to the dynamics of the workloads.

Reason#2: Less Accumulated Workloads. In terms of latency, Distream outperforms the baselines because with higher compute resource utilization, Distream is able to process the workloads at a much faster rate than the baselines, preventing the workloads from being accumulated at cameras or edge cluster. As such, Distream has much less workloads waiting in the queue, which significantly reduces the workload queuing time and hence the latency. To see this, we continuously track the accumulated workloads in the local queue at each camera and the edge cluster. As shown in Figure 12 (b), Distream is able to achieve a near-zero amount of accumulated workloads for about 80% of the time. In contrast, VideoEdge-L is experiencing significantly high amount of accumulated workloads for most of the time. As such, the workload queuing time in Distream is much lower, which is the root cause of its low latency.

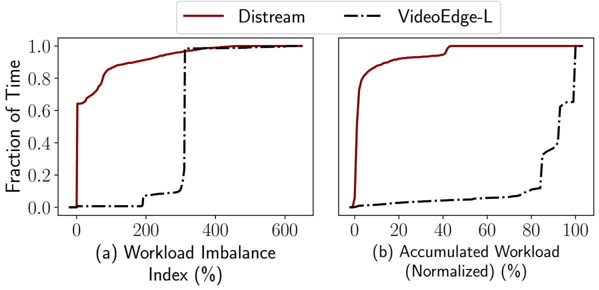


Figure 12: Illustration on why Distream outperforms baselines.

5.3 Component-wise Analysis

The design of Distream enables adaptive cross-camera workload balancing and adaptive camera-cluster workload partitioning. In this section, we implemented two breakdown versions of Distream to take a closer look at the contribution of each component.

- **Distream-L** has adaptive cross-camera workload balancing, but has static camera-cluster workload partitioning based on the compute capacity ratio of cameras and cluster.
- **Distream-P** has adaptive camera-cluster workload partitioning, but does not enable cross-camera workload balancing.

Table 4 shows the comparison results on the Campus Surveillance application. As shown, the performance gains brought by both Distream-L and Distream-P are similar and significant. On average, Distream-L achieves 2.4× throughput gain and 5.9× latency reduction respectively compared to Camera-Only. Distream-P achieve 2.5× throughput gain and 6.3× latency reduction respectively compared to Camera-Only. This result indicates the importance of both adaptive cross-camera workload balancing and adaptive camera-cluster workload partitioning to the whole system. Distream is able to combine the advantages of both Distream-L and Distream-P to achieve better performance than using only one of them.

	Throughput				Latency			
	avg	med	75th	99th	avg	med	75th	99th
Distream-L	2.4×	2.5×	2.3×	2.2×	5.9×	8.2×	5.5×	5.2×
Distream-P	2.5×	2.5×	2.3×	2.1×	6.3×	9.2×	6.6×	5.1×

Table 4: Component-wise analysis of Distream. Performance gains are over Camera-Only.

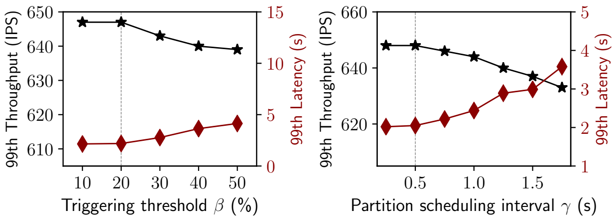
5.4 Sensitivity Analysis

The design of Distream involves two key system hyper-parameters: (i) the cross-camera workload balancing threshold β (§3.2), and (ii) the camera-cluster workload partition scheduling interval γ (§3.4). In this section, we evaluate the impact of these system hyper-parameters on the performance of Distream. In addition, since cross-camera workload balancing requires migrating workloads through the network, we also evaluate the impact of network bandwidth on the performance of Distream.

Impact of Cross-Camera Workload Balancing Threshold β. Figure 13a shows Distream's sensitivity to the cross-camera workload balancing threshold β. We only show 99th percentile throughput (left y-axis, black) and 99th percentile latency (right y-axix, red) because they are sensitive to hyper-parameter changes and can better reflect the impact on the system performance. We observe that when β is below 20%, the performance of Distream does not change much. When β exceeds 20%, the 99th percentile throughput begins to drop and 99th latency increases. This is because a larger β would trigger load balancing less often. As a result, the system may suffer from performance loss due to workload imbalance. Therefore, we set β to be 20%.

Impact of Camera-Cluster Workload Partition Scheduling Interval γ. Figure 13b shows Distream's sensitivity to the camera-cluster workload partition scheduling interval γ. We see that the 99th percentile throughput and 99th percentile latency do not change much when the interval is less than 0.5s. When γ is between 0.5s to 1.5s, the system performance has slightly declined. When γ is greater than 1.5s, the system performance significantly declines as the interval increases. Therefore, we set γ to 0.5s.

Impact of Network Bandwidth. We also examine Distream's sensitivity to network bandwidth. We evaluate Distream with network bandwidth at 10Mbps (low bandwidth) and 50Mbps (high bandwidth), which covers the bandwidth range in standard video surveillance systems on the market. Table 5 lists our evaluation results. As shown, both the throughput and latency are very similar between 10Mbps to 50Mbps. This result indicates that Distream is robust to network bandwidth changes and is able to achieve high performance even with a network bandwidth of 10Mbps.



(a) Impact of cross-camera workload balancing threshold β.

(b) Impact of camera-cluster workload partition schedule interval γ.

Figure 13: Impact of system hyper-parameters on the performance of Distream.

	Throughput (IPS)				Latency (s)			
	avg	med	75th	99th	avg	med	75th	99th
Low Bandwidth (10Mbps)	489.1	509	590	647	0.34	0.09	0.47	2.05
High Bandwidth (50Mbps)	490.9	510	593	650	0.31	0.07	0.41	1.75

Table 5: Performance of Distream under low (10Mbps) and high (50Mbps) network bandwidths.

5.5 Scaling Performance

To evaluate the scaling performance of Distream, we use the Campus Surveillance dataset and examine the throughput and latency as Distream scales up its number of smart cameras from 6 to 12, 18, and 24. To match the compute resources with new workloads brought by the new cameras, we add one GPU to the edge cluster whenever six smart cameras are added to the system.

Figure 14 illustrates the scaling performance of Distream in terms of throughput (upper) and latency (lower). As shown, Distream is able to scale its throughput up nearly linearly. Specifically, Distream achieves a 3.95× average throughput from 489.1 IPS when processing videos streamed from 6 cameras to 1931.4 IPS when processing videos streamed from 24 cameras. Meanwhile, Distream is able to maintain a similar low latency when scaling up.

5.6 System Overheads

The system overheads incurred by the design of Distream come from two sources. The first source comes from solving the optimization problems for cross-camera workload balancing in §3.2 and identifying the optimal DAG partition points in §3.4. Their overhead is 16us and 4us with six cameras and one GPU, and 21us and

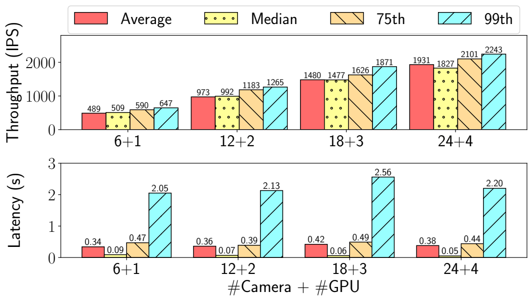


Figure 14: Scaling performance of Distream.

8us when scaled up to 24 cameras and 4 GPUs. This result indicates that our proposed heuristics are highly efficient and incur negligible overheads. The second source of overhead comes from the RPC call for migrating workloads across smart cameras (§3.4). As shown in Table 6, even if under the low bandwidth condition (i.e., 10Mbps), each RPC call only takes 2.1ms on average for workload migration. When we employ batching RPC, this overhead is amortized when the RPC batch size increases, making the overheads of cross-camera workload balancing negligible.

Batch Size	Total Migration Time	Avg RPC Overhead
1	2.1ms	2.1ms
10	7.4ms	0.74ms
20	9.2ms	0.46ms

Table 6: Cross-camera workload balancing overheads of Distream.

5.7 Benefits to Existing Systems

As summarized in Table 1, existing live video analytics systems are agnostic to the workload dynamics in real-world deployments. Instead, they focus on designing different techniques to optimize the resource-accuracy tradeoff of live video analytics systems. In this section, we add the workload adaptation techniques proposed in Distream onto VideoEdge [19] to examine how Distream could enhance its performance. We select VideoEdge instead of other status quo live video analytics systems listed in Table 1 because VideoEdge is also a fully distributed framework that partitions the video analytics pipeline across cameras and cluster.

Figure 15 compares the performance between VideoEdge and VideoEdge + Distream. As illustrated, VideoEdge + Distream is able to generate a better resource-accuracy trade-off curve than VideoEdge. In particular, Distream is able to boost the accuracy by 21% when compute resource is limited, and is able to boost the accuracy by 7% when compute resource is adequate. Such improvement is made possible due to the higher resource utilization brought by the workload adaptation mechanism of Distream. With higher resource utilization, VideoEdge does not need to sacrifice as much accuracy as before to trade for resources.

6 RELATED WORK

Live video analytics is a killer application not only for mobile vision [13, 14, 20, 24] but also for large-scale distributed settings. Our work is closely related to DL-based live video analytics systems with distributed cameras. A majority of these systems are designed to process video streams in a centralized cluster. For example, Focus [18] customizes small DL classifiers to generate object indexes in each video frame to enable fast offline video query. NoScope [25] uses specialized DL models to reduce inference overheads for throughput gain with small accuracy loss. Besides them, Chameleon [22], VideoStorm [39], and AWStream [38] leverage resource-accuracy tradeoffs for system optimization. Chameleon [22] reduces the overhead caused by searching for optimal tradeoff configuration. VideoStorm [39] exploits the variety of quality and lag goals in video analytics tasks and optimizes compute resources in GPU clusters. AWStream [38] is focused on adapting to network bandwidth variation to achieve low-latency high-accuracy wide-area streaming analytics. Different from these works, instead of trading accuracy for less compute resource consumption, Distream achieves high throughput and low latency by maximizing the resource utilization across cameras and edge cluster without sacrificing the accuracy.

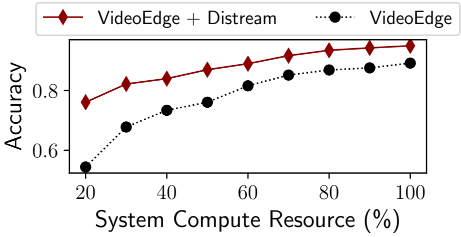


Figure 15: Resource-accuracy tradeoff curve comparison between VideoEdge and VideoEdge + Distream.

The closest work to Distream is VideoEdge [19]. Although Distream and VideoEdge are both built upon a distributed architecture that involves smart cameras and edge cluster for live video analytics, they are different in three aspects. First, Distream focuses on enabling workload-adaptive live video analytics while VideoEdge is workload agnostic. Second, VideoEdge targets optimizing resource-accuracy tradeoff while Distream focuses on maximizing throughput and attaining latency SLO without compromising accuracy. Third, VideoEdge performs video analytics pipeline placement across cameras, edge and cloud where the dynamic network bandwidth is the main bottleneck. In contrast, Distream performs workload allocation across cameras and edge cluster based on workload dynamics to eliminate the bottleneck in compute resources. In fact, Distream is complementary to VideoEdge: as shown in §5.7, Distream is able to provide a better resource-accuracy tradeoff curve when integrated with VideoEdge.

7 CONCLUSION AND FUTURE WORK

In this paper, we present the design, implementation, and evaluation of Distream, a distributed workload-adaptive live video analytics system based on the smart camera-edge cluster architecture. Distream addresses the key challenges brought by workload dynamics in real-world deployments, and contributes novel techniques that complement existing live video analytics systems. We have implemented Distream and conducted a rich set of experiments with a testbed consisting of 24 cameras and a 4-GPU edge cluster on two real-world distributed video datasets. Our experimental results show that Distream consistently outperforms the status quo in terms of throughput, latency, and latency SLO miss rate. Therefore, we believe Distream represents a significant contribution to enabling live video analytics at scale. In the current form, Distream treats cross-camera workload balancing and camera-cluster partitioning as two separate components. We plan to work on a joint solution as our future work. We will also work on designing common programming and network abstraction to support live video analytics application development based on our framework, and plan to expand our framework to the wireless settings.

8 ACKNOWLEDGEMENT

We sincerely thank the anonymous reviewers and our shepherd for their valuable feedback. This work was partially supported by NSF Awards CNS-1617627, CNS-1814551, and PFI:BIC-1632051.

REFERENCES

[1] 2016. Nvidia Titan X. https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/

[2] 2017. 24-Port Gigabit Stackable Smart Managed Switch with 4 10GbE SFP+ ports. http://us.dlink.com/products/business-solutions/dgs-1510-28x/

[3] 2017. Nvidia Jetson TX1. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[4] 2017. OpenCV Background Subtraction. https://docs.opencv.org/3.4/db/d5c/tutorial_py_bg_subtraction.html

[5] 2018. Networking Solutions for IP Surveillance. https://www.netgear.com/images/pdf/IP-Video-Surveillance_Networking-Solution-Guide.pdf

[6] 2018. Nvidia Jetson TX2. https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/

[7] 2019. JacksonHole. https://www.seejh.com/live

[8] Faruk Akgul. 2013. *ZeroMQ*. Packt Publishing Ltd.

[9] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya R Dulloor. 2019. Scaling video analytics on constrained edge nodes. *arXiv preprint arXiv:1905.13536* (2019).

[10] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*. 301–314.

[11] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A low-latency online prediction serving system. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 613–627.

[12] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. 49–62.

[13] Biyi Fang, Xiao Zeng, Faen Zhang, Hui Xu, and Mi Zhang. 2020. FlexDNN: Input-Adaptive On-Device Deep Learning for Efficient Mobile Vision. In *Proceedings of the 5th ACM/IEEE Symposium on Edge Computing (SEC)*.

[14] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom)*. 115–127.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2980–2988.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[18] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying large video datasets with low latency and low cost. *arXiv preprint arXiv:1801.03493* (2018).

[19] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 115–131.

[20] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. 82–95.

[21] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph Gonzalez. 2019. Scaling video analytics systems to large camera deployments. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. 9–14.

[22] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 253–266.

[23] Junchen Jiang, Yuhao Zhou, Ganesh Ananthanarayanan, Yuanchao Shu, and Andrew 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*. 1–7.

[24] Shuang Jiang, Zhiyao Ma, Xiao Zeng, Chenren Xu, Mi Zhang, Chen Zhang, and Yunxin Liu. 2020. SCYLLA: QoE-aware Continuous Mobile Vision with FPGA-based Dynamic Deep Neural Network Reconfiguration. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1369–1378.

[25] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1586–1597.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.

[28] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[29] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J Gordon. 2018. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 631–645.

[30] Xiaolei Ma, Houyue Zhong, Yi Li, Junyan Ma, Zhiyong Cui, and Yinhai Wang. 2020. Forecasting transportation network speed using deep capsule networks with nested lstm models. *IEEE Transactions on Intelligent Transportation Systems* (2020).

[31] Shadi A Noghabi, Landon Cox, Sharad Agarwal, and Ganesh Ananthanarayanan. 2020. THE EMERGING LANDSCAPE OF EDGE COMPUTING. *GetMobile: Mobile Computing and Communications* 23, 4 (2020), 11–20.

[32] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[35] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: a GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 322–337.

[36] Shivaram Venkataraman, Aurojit Panda, Kay Ousterhout, Michael Armbrust, Ali Ghodsi, Michael J Franklin, Benjamin Recht, and Ion Stoica. 2017. Drizzle: Fast and adaptable stream processing at scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 374–389.

[37] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[38] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. 2018. AWStream: adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 236–252.

[39] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance.. In *NSDI*, Vol. 9. 1.

[40] Mi Zhang, Faen Zhang, Nicholas D Lane, Yuanchao Shu, Xiao Zeng, Biyi Fang, Shen Yan, and Hui Xu. 2020. Deep Learning in the Era of Edge Computing: Challenges and Opportunities. *Fog Computing: Theory and Practice* (2020), 67–78.

[41] Zoran Zivkovic and Ferdinand Van Der Heijden. 2006. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters* 27, 7 (2006), 773–780.