Server-Driven Video Streaming for Deep Learning Inference

Kuntai Du*, Ahsan Pervaiz*, Xin Yuan, Aakanksha Chowdhery†, Qizheng Zhang, Henry Hoffmann, Junchen Jiang
University of Chicago †*Google*

ABSTRACT

Video streaming is crucial for AI applications that gather videos from sources to servers for inference by deep neural nets (DNNs). Unlike traditional video streaming that optimizes visual quality, this new type of video streaming permits aggressive compression/pruning of pixels not relevant to achieving high DNN inference accuracy. However, much of this potential is left unrealized, because current video streaming protocols are driven by the video *source* (camera) where the compute is rather limited. We advocate that the video streaming protocol should be driven by *real-time feedback from the server-side DNN*. Our insight is two-fold: (1) server-side DNN has more context about the pixels that maximize its inference accuracy; and (2) the DNN's output contains rich information useful to guide video streaming. We present *DDS* (*D*NN-*D*riven *S*treaming), a concrete design of this approach. DDS continuously sends a low-quality video stream to the server; the server runs the DNN to determine where to re-send with higher quality to increase the inference accuracy. We find that compared to several recent baselines on multiple video genres and vision tasks, DDS maintains higher accuracy while reducing bandwidth usage by upto 59% or improves accuracy by upto 9% with no additional bandwidth usage.

CCS CONCEPTS

• **Networks → Application layer protocols**; • **Information systems → Data streaming**; **Data analytics**; • **Computing methodologies → Computer vision problems**;

KEYWORDS

video analytics, video streaming, deep neural networks, feedback-driven

ACM Reference Format:

Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, Junchen Jiang. 2020. Server-Driven Video Streaming for Deep Learning Inference. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3387514.3405887

*Both authors contributed equally to this research.

SIGCOMM '20, August 10–14, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7955-7/20/08…$15.00
https://doi.org/10.1145/3387514.3405887

1 INTRODUCTION

Internet video must balance between maximizing application-level quality and adapting to limited network resources. This perennial challenge has sparked decades of research and yielded various models of user-perceived quality of experience (QoE) and QoE-optimizing streaming protocols. In the meantime, the proliferation of deep learning and video sensors has ushered in new analytics-oriented applications (*e.g.,* urban traffic analytics and safety anomaly detection [5, 22, 27]), which also require streaming videos from cameras through bandwidth-constrained networks [24] to remote servers for deep neural nets (DNN)-based inference. We refer to it as *machine-centric video streaming*. Rather than maximizing human-perceived QoE, machine-centric video streaming maximizes for DNN *inference accuracy*. This contrast has inspired recent efforts to compress or prune frames and pixels that may not affect the DNN output (*e.g.,* [30–32, 36, 48, 76, 78, 80]).

A key design question in any video streaming system is *where to place the functionality of deciding which actions can optimize application quality* under limited network resources. Surprisingly, despite a wide variety of designs, most video streaming systems (both machine-centric and user-centric) take an essentially *source-driven* approach—it is the content source that decides how the video should be best compressed and streamed. In traditional Internet videos (*e.g.,* YouTube, Netflix), the server (the source) encodes a video at several pre-determined bitrate levels, and although the mainstream protocol, DASH [7], is dubbed a client-driven protocol, the client does not provide any instant user feedback on user-perceived QoE to let server re-encode the video. Current machine-centric video streaming relies largely on the camera (the source) to determine which frames and pixels to stream.

While the source-driven approach has served us well, we argue that it is suboptimal for analytics-oriented applications. The source-driven approach hinges on two premises: (1) the application-level quality can be estimated by the video source, and (2) it is hard to measure user experience directly in real time. Both need to be revisited in machine-centric video streaming.

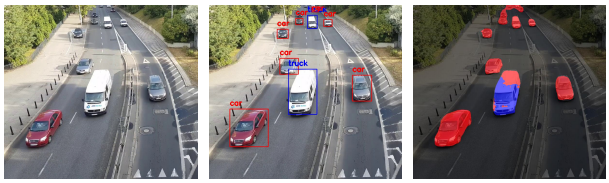
First, it is inherently difficult for the source (camera) to estimate the inference accuracy of the server-side DNN by itself. Inference accuracy depends heavily on the compute-intensive feature extractors (tens of NN layers) in the server-side DNN. The disparity between most cameras and GPU servers in their compute capability means that any camera-side heuristics are unlikely to match the complexity of the server-side DNNs. This mismatch leads to the suboptimal performance of the source-driven protocols. For instance, some works use inter-frame pixel changes [30] or cheap object detectors [80] to identify and send only the frames/regions that contain new objects, but they may consume more bandwidth than necessary (*e.g.,* background changes causing pixel-level differences) and/or cause more false negatives (*e.g.,* small objects could be missed by the cheap camera-side object detector).

Second, while eliciting real-time feedback from human users may be hard, DNN models can provide *rich* and *instantaneous* feedback. Running an object-detection DNN on an image returns not only detected bounding boxes, but also additional feedback for free, like the confidence score of these detections, intermediate features, etc. Moreover, such feedback can be extracted on-demand by probing the DNN with extra images. Such abundant feedback information has not yet been systematically exploited by prior work.

In this paper, we explore an alternative *DNN-driven* approach to machine-centric video streaming, in which video compression and streaming are driven by how the server-side DNN reacts to real-time video content. DNN-driven video streaming follows an *iterative* workflow. For each video segment, the camera first sends it in low quality to the server for DNN inference; the server runs the DNN and derives some *feedback* about the most relevant regions to the DNN inference and sends this feedback to the camera; and the camera then uses the feedback to re-encode the relevant regions in a higher quality and sends them to the server for more accurate inference. (The workflow can have multiple iterations though this paper only considers two iterations). Essentially, by deriving feedback directly from the server-side DNN, it sends high-quality content only in the minimal set of relevant regions necessary for high inference accuracy. Moreover, unlike prior work that requires camera-side vision processing or hardware support (*e.g.,* [30, 48, 80]), we only need standard video codec on the camera side.

The challenge of DNN-driven protocols, however, is *how to derive useful feedback from running DNN on a low-quality video stream.* We present *DDS* (*D*NN-*D*riven *S*treaming), a concrete design which utilizes the *feedback regions* derived from DNN output on the low-quality video and sparingly uses high-quality encoding for the relatively small number of regions of interest. We apply DDS to three vision tasks: object detection, semantic segmentation, and face recognition. The insight is that the low-quality video may not suffice to get sufficient DNN inference accuracy, but it can produce surprisingly accurate feedback regions which intuitively require higher quality for the DNN to achieve desirable accuracy. Feedback regions are robust to low-quality videos because they are more akin to binary-class tasks (*i.e.,whether* a region might contain an object and need higher quality) than to more difficult tasks such as classifying *what* object is in each region. Moreover, DDS derives feedback regions from DNN output without extra GPU overhead.

DDS is not the first to recognize that different pixels affect DNN accuracy differently, *e.g.,* prior works also send only selected regions/frames to trigger server-side inference [54, 80]. But unlike DDS, these regions are selected either by simple camera-side logics [80] which suffer from low accuracy, or by region-proposal networks (RPNs) [54] which are designed to capture where objects are likely present, rather than where higher quality is needed (*e.g.,* large targeted objects will be selected by RPNs but they do not need high video quality to be accurately recognized). Using RPNs also limits the applications to object detection and does not generalize to other tasks such as semantic segmentation. In a broader context, DDS is related and complementary to the trend in deep learning of using *attention* mechanisms (*e.g.,* [61, 74])—attention improves DNN accuracy by focusing *computation* on the important regions, while DDS improves *bandwidth efficiency* by sending only a few regions in high quality to achieve the same DNN accuracy as if the whole video is sent in the highest quality.



(a) Input (b) Object detection (c) Sem. segmentation

Figure 1: *The input and output of object detection and semantic segmentation on one example image. We use red to label the car and blue to label the truck.*

We evaluate DDS and a range of recent solutions [30, 54, 76, 78, 80] on three vision tasks. Across 49 videos, we find DDS achieves same or higher accuracy while cutting bandwidth usage by upto 59%, or uses the same bandwidth consumption while increasing accuracy by 3-9%. This work does not raise any ethical issues.

2 MOTIVATION

We start with the background of video streaming for distributed video analytics, including its need, performance metrics, and design space. We then use empirical measurements to elucidate the key limitations of prior solutions.

2.1 Video streaming for video analytics

Vision tasks under consideration: We consider three computer vision tasks—object detection, semantic segmentation, and face recognition. Figure 1 shows an example input and output of object detection (one label for each bounding box) and semantic segmentation (one label for each pixel). These tasks are widely used in real-world scenarios to detect/segment objects of interest and their results are used as input to high-level applications (*e.g.,* vehicle collision detection).

Why streaming videos out from cameras? On one hand, computer vision accuracy has been improved by deep learning at the cost of increased compute demand. On the other hand, low prices of high-definition network-connected cameras make them widely deployed in traffic monitoring [27], video analytics in retail stores [12], and inspection of warehouses or remote industrial sites [38]. Thus, the camera operators must scale out the compute costs of analyzing ever more camera feeds [2, 6, 21]. One solution is to *offload the compute-intensive inference (partially or completely) to centralized GPU servers.* (Sometimes, video feeds must be kept local due to privacy regulations, but it is beyond our scope.) For the sake of discussion, let us calculate the costs of 60 HD cameras each running ResNet50 classification at 90FPS. We use ResNet50 classifier because our applications require more complex DNN models (*e.g.,* FasterRCNN-ResNet101) cannot run on Jetson TX2 [9] at 30FPS. Now, buying 60 Raspberry Pi 4 Cameras and an NVIDIA Tesla T4 GPU (with a throughput of running ResNet50 at 5,700FPS [17]) costs \$23 × 60(cameras)[19]+\$2000(GPU)[13]= \$3.4K. Buying 60 NVIDIA Jetson TX2 cameras (each running ResNet50 at 89FPS [16]) costs about \$400[15]×60 = \$24K, which is one order of magnitude more expensive. These numbers may vary over time, but the price gap between two approaches is likely to remain. The calculation

does not include the network bandwidth to send the videos to a server, which is what we will minimize.

Performance metrics: An ideal video streaming protocol for video analytics should balance three metrics: accuracy, bandwidth usage, and freshness.

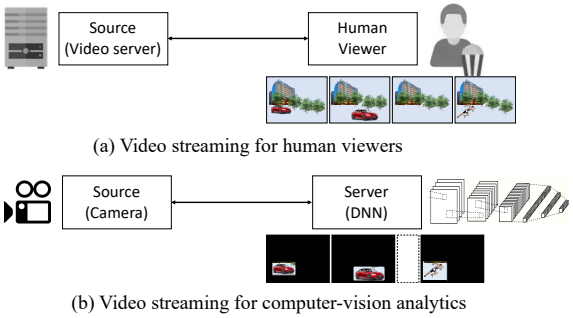
- *Accuracy:* We define accuracy by the *similarity* between the DNN output on each frame when the video is streamed to the server under limited bandwidth and the DNN output on each frame when the original (highest quality) video is streamed to the server. By using the DNN output on the highest-quality video (rather than the human-annotated labels) as the "ground truth", we can reveal any negative impact of video compression and streaming on DNN inference, without being affected by any errors made by the DNN itself. This is consistent with recent work (*e.g.,* [45, 78, 79]). We measure the accuracy by *F1 score* in object detection (the harmonic mean of precision and recall for the detected objects' location and class labels) and by IoU in semantic segmentation (the intersection over union of pixels associated to the same class).
- *Bandwidth usage:* In general, the total cost of operating a video analytics system includes the camera cost, the network cost paid to stream the video from the camera to the server, and the cost of the server. In this paper, we focus on reducing the network cost through reducing the bandwidth usage. §2.4 will highlight the deployment settings in which the total costs of a video analytics system are dominated by the network cost and thus reducing bandwidth usage is crucial. We measure the bandwidth usage by the size of the sent video divided by its duration.
- *Average response delay (freshness):* Finally, we define freshness by the average processing delay per object (or per pixel for semantic segmentation), *i.e.,* the expected time between when an object (or a pixel) first appears in the video feed and when its region is detected and correctly classified, which includes the time to send it to the server and to run inference on it.[2]

2.2 Design space of video analytics systems

Next, we discuss the design space of how video analytics systems can potentially navigate the tradeoffs among these performance metrics along five dimensions:

- *Leveraging camera-side compute power:* Since the camera can naturally access the raw video, one can leverage the camera's local compute power (if any) to discard frames [30, 48] or regions [54, 80] that may not contain important information. As we will elaborate in §2.3, the accuracy of such local filtering heuristics may cause significant accuracy drops.
- *Model distillation:* DNNs are often trained on large datasets, but when used exclusively for a specific category of video scenes, a DNN can be shrunk to a much smaller size (*e.g.,* via knowledge distillation), in order to save compute cost (GPU cycles) without hurting accuracy (*e.g.,* [48]). This approach is efficient only in training smaller DNNs that work well on less expensive hardware.

[2]Average response delay is meaningful if the follow-up analysis can by updated when a new objects/pixel is detected/classified (*e.g.,* estimating the average speed of vehicles on a road). That said, this definition does not apply to applications that are sensitive to *worst-case* delays rather than average delay, *e.g.,* if one queries for the total number of vehicles, the answer will not be completed until all vehicles are detected.



(a) Video streaming for human viewers

(b) Video streaming for computer-vision analytics

Figure 2: *Unlike video streaming for human viewers, machine-centric video streaming has unique bandwidth-saving opportunities.*

- *Video codec optimization:* Unlike traditional video codecs that optimize for human visual quality, video analytics emphasizes inference accuracy and thus opens up possibility to more analytics-oriented video codecs (*e.g.,* analytics-aware super resolution [76]).
- *Temporal configuration adaptation:* To cope with the temporal variance of video content, one can adapt the key configurations (*e.g.,* the frame rate, resolution and DNN model) to save compute costs [45] or network costs [78]. That said, it fails to exploit the uneven spatial distribution of important information in videos.
- *Spatial quality adaptation:* Information of interest (*e.g.,* target objects) is sparsely distributed in each frame, so some pixels are more critical to accurate DNN inference than others. One can save bandwidth usage by encoding each frame with a spatially uneven quality distribution (*e.g.,* region-of-interest encoding [54]) so that high video quality is used only where pixels are critical to DNN inference [54, 80].

In this paper, we take a pragmatic stance to focus on a specific point in the design space—no camera-side frame-dropping heuristics, no model distillation (use the server-side DNN as-is), and no change to the video codec; instead, we use the server-side DNN output to drive spatial quality adaptation.

2.3 Potential room for improvement

Traditional video streaming maximizes human quality of experience (QoE)—a high video resolution and smooth playback (minimum stalls, frame drops or quality switches) [35, 46, 50]. For machine-centric video streaming, however, it is crucial that the server-received video has sufficient video quality in the regions that heavily affect the DNN's ability to identify/classify objects; however, the received video does not have to be smooth or have high quality everywhere.

This contrast has a profound implication—machine-centric streaming could achieve high "quality" (*i.e.,* accuracy) using much less bandwidth. Each frame can be spatially encoded with *non-uniform quality levels*. In object detection, for instance, one may give low quality to (or even blackout) the areas other than the objects of interest (Figure 2(b))[3]. While rarely used in traditional video streaming, this scheme could significantly reduce bandwidth consumption and response delay, especially because objects of interest usually only occupy a fraction of the video size. Figure 3 shows that across

[3]This may look like region-of-interest (ROI) encoding [59], but even ROI encoding does not completely remove the background either, and the ROIs are defined with respect to human perception.

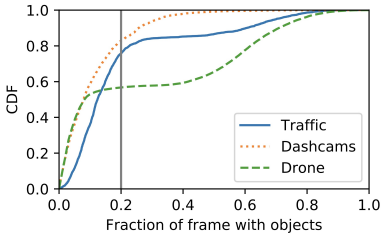


Figure 3: *Bandwidth-saving opportunities: Over 50-80% of frames, the objects (cars or pedestrians) occupy less than 20% of the frame area, so most pixels do not contribute to the accuracy of video analytics.*

three different scenarios (the datasets will be described in §5.1), in 50-80% of frames, the objects of interest (cars or pedestrians) only occupy less than 20% of the spatial area of a frame. We also observe similar uneven distributions of important pixels in face recognition and semantic segmentation. The question then is how to fully explore the potential room for improvement?

2.4 Preliminary comparison of existing solutions

We present a framework to compare the performance, in accuracy, total cost, and response delay, of four baselines: camera-side local inference ("Camera-only"), server-side inference ("AWStream"), and selecting frames/regions by the camera and sending them to server for further analysis ("Vigil" and "Glimpse"). We then analyze the sources of their (suboptimal) performance in §2.5. The tests are performed on the traffic videos in our dataset (§5.1). We will give more details about their implementations and include more baselines in the full evaluation (§5).

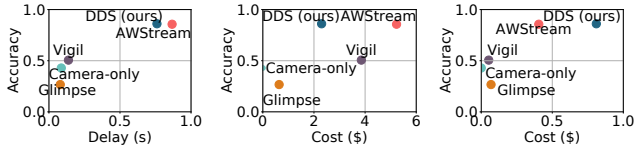
~~For each solution s, we use a fixed camera-side logic $Local_s$~~ and a fixed server-side DNN $Remote_s$. We use P_s to denote the data (frames or videos, depending on the solution) sent from the camera to the server. They together determine the accuracy of s: $Acc(Local_s, Remote_s, P_s)$.[4] Note that P_s is tunable by changing the internal configurations of s, and with fixed $Local_s$ and $Remote_s$, the cost-delay-accuracy tradeoff of s will be governed by P_s. We use the same server-side DNN (FasterRCNN-ResNet101) to make sure the accuracies are calculated with the same ground truth.

Figure 4a shows the delay-accuracy tradeoffs of the four solutions (and our solution which will be introduced in next section). Here, the delay is the average response delay per frame as measured in our testbed. (We explain the hardware choice in §5.1.) Note that the local model running on the camera ("Camera-only") has relatively lower accuracy than Vigil (which uses both the local DNN and the server DNN) and AWStream (which fully relies on the server DNN results). We will explain the reasons in §2.5.

Figure 4b and Figure 4c show the costs to achieve their respective performance in Figure 4a under two price settings. We measure the cost by the average total cost of analyzing a 720p HD video at 30FPS (~5Mbps) for an hour.

- *Setting 1 (Total cost is dominated by network):* A camera is connected to an *in-house* server through an *LTE* network. Since the camera and the server are purchased upfront, their costs amortized per frame will approach zero in the long run, but the LTE cost is paid by time. Here, we consider the AT&T 4G LTE plan, $50 per month [3] for 30GB data (before the speed drops to measly 128kbps) [10], or equivalently $0.75 for streaming at 1Mbps for one hour. Thus, the per-hour total cost of a solution s is $Cost_s \approx \$0.75 \cdot Size(P_s)$, where $Size(P)$ is the total bandwidth usage (in Mbps) to send P.
- *Setting 2 (Total cost is dominated by server):* A camera is connected to a *cloud* server through *cheap* wired network. Unlike the previous setting, the cloud server is paid by usage so its cost grows with more server-side compute, but the network cost is negligible compared to 4G LTE plans. To run the server-side DNN at 30FPS, we assume that we need 3 NVIDIA Telsa K80 cards and it costs $0.405 per hour [11] (and other cloud providers have similar price ranges). The per-hour total cost of s, therefore, is $Cost_s \approx \$0.405 \cdot Frac(P_s)$, where $Frac(P)$ is the number of frames in P divided by all frames.

[4] Of course, the value P_s and accuracy are video-dependent, but we omit it for simplicity since we compare solutions on the same videos.



(a) Delay-accuracy **(b) Cost vs accuracy (Setting 1)** **(c) Cost vs accuracy (Setting 2)**

Figure 4: *The trade-offs among cost, delay, and accuracy on the traffic videos in our dataset under two settings. The cost in setting 1 is dominated by the network cost, so schemes that save bandwidth usage are more favorable. The cost in setting 2 is dominated by the server cost, so saving bandwidth does not yield better solutions.*

In the first setting (Figure 4b, where the total cost is dominated by the network cost), prior solutions show unfavorable cost-accuracy tradeoffs (when compared with our solution). However, in the second setting (Figure 4c, where the total cost is dominated by the server cost), prior solutions in general strike good cost-accuracy tradeoffs (compared with ours). This is largely because some of them (Vigil and Glimpse) are designed to minimize server-side compute cost, which this paper does not explicitly optimize.

2.5 Sources of the limitations

Existing solutions for video streaming are essentially *source*-driven—the decisions of which pixels/frames should be compressed and sent to the server are made by the source (camera), with little real-time feedback from the server-side DNN that analyzes the video. The fundamental issue of source-driven protocol is that any heuristic that fits camera's limited compute capacity is hard to precisely identify the minimum information that is needed by the server-side DNN to achieve high accuracy. The result is a unfavorable tradeoff between bandwidth and accuracy (*e.g.*, Figure 4b): any gain of accuracy comes at the cost of considerably more bandwidth usage.

This problem manifests itself differently in two types of source-driven solutions. The first type is *uniform-quality streaming*, which modifies the existing video protocols and adapts the quality level to maximize inference accuracy under a bandwidth constraint. For instance, AWStream [78] uses DASH/H.264 and periodically re-profiles the relationship between inference accuracy and video quality. CloudSeg [76] sends a video at a low quality but upscales the video using super resolution on the server. They have two limitations. First, they do not leverage the uneven distribution of

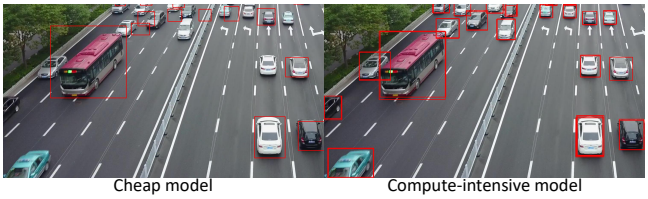


Figure 5: *Contrasting the inference results between a cheap model (SSD-MobileNet-v2) and a compute-intensive model (FasterRCNN-ResNet101) on the same image. The compute-intensive model is more accurate when the video content is challenging (e.g., having many small objects).*

important pixels; instead, the videos are encoded by traditional codecs with the same quality level on each frame. Second, while they get feedback from the server DNN, it is not based on real-time video content, so it cannot suggest actions like increasing quality on a specific region in the current frame.

The second type is *camera-side heuristics* that identifies important pixels/regions/frames that might contain information needed by the server-side analytics engine (*e.g.,* queried objects) by running various local heuristics (*e.g.,* checking significant inter-frame difference [30, 54], a cheap vision model [31, 32, 48, 80]), or some DNN layers [36, 72]. These solutions essentially leverage the camera-side compute power to save server compute cost and network cost [36]. However, these cheap camera-side heuristics are inherently less accurate than the more complex DNN models on the server, especially when the video content is challenging (*e.g.,* consisting of many small objects, which is typical for drone and traffic videos, as illustrated in Figure 5). Any false negatives of these camera-side heuristics will preclude the server from detecting important information; any false positives (*e.g.,* pixel changes on the background) will cost unnecessary bandwidth usage.

3 DNN-DRIVEN VIDEO STREAMING

In this section, we present the design of DDS and discuss its design rationale and performance tradeoffs.

3.1 Overall workflow

We explore an alternative approach, called *DNN-driven streaming* (DDS). In DDS, the compression and streaming behaviors are driven by the feedback judiciously generated by the server-side DNN, rather than the low-complexity local heuristics on the camera side, in order to capture what the analytics engine needs from the real-time video content. Figure 6 contrasts the workflow of DDS with that of the traditional source-driven approach: source-driven streaming is "single-shot" (*i.e.,* camera using simple heuristics to determine how the video should be streamed out), but DDS is *iterative* and logically contains two streams:

- **Stream A (passive, low quality):** The camera continuously sends the video in low quality to the server.
- **Stream B (feedback-driven):** The server frequently (*e.g.,* every handful of frames) extracts the *feedback regions* from the DNN outputs on the Stream A video and sends them back to the camera as feedback. Upon receiving the feedback from the server, the camera then re-encodes the recent history video accordingly and sends it to the server for a second-round inference on these "zoomed-in" images.

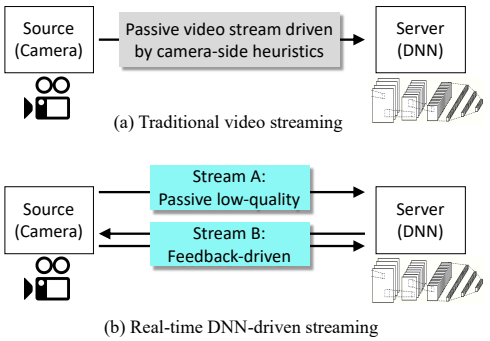


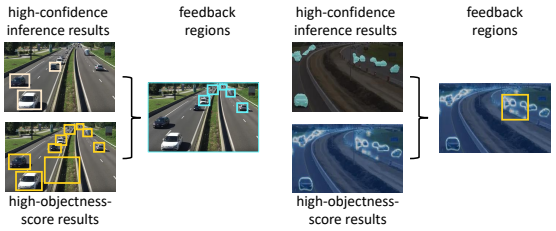
Figure 6: *Contrasting the new real-time DNN-driven streaming (iterative) with traditional video streaming in video analytics.*

The key to DDS's success is the design of the feedback regions, which we discuss next.

3.2 Feedback regions

High-level framework: DDS extracts the feedback regions by utilizing the information naturally returned/generated by the server-side DNN, rather than a wholesale change on the DNN architecture. To deal with a variety of DNNs with different outputs, DDS uses a custom logic to extract feedback regions from each DNN. But these logics share the same framework (explained next) and are integrated with DNNs using a similar interface (explained in §4.1). For convenience, we use the term "elements" to denote the unit of a vision task—a bounding box (in object detection and face recognition) and a pixel (in semantic segmentation). At a high level, given the DNN output on the low-quality video, we first identify the elements that are likely to be in the DNN output on the high-quality video but not in the DNN output on the low-quality video, and we then pick a small number of rectangles (for encoding efficiency) as the feedback regions to cover these elements. Next, we present how this high-level logic is used in two classes of vision tasks.

Object detection (based on bounding boxes): Most bounding-box-based DNNs are anchor-based (though some are anchor-free [29]). This means that a DNN will first identify regions that might contain objects and then examine each region. Each proposed region is associated with an objectness score that indicates how likely an object is in the region. For DNNs (*e.g.,* FasterRCNN-ResNet101 [68]) that use region proposal networks (RPNs), each proposed region is directly associated with an objectness score. However, not all object-detection DNNs use RPNs. For instance, Yolo [66] does not and instead, it assigns a score for each class in each region in the final output. In this case, we sum up the scores of non-background classes as the objectness score, which indicates how likely a region includes a non-background object. We keep regions with objectness score over a threshold (*e.g.,* 0.5 for FasterRCNN-ResNet101, and Figure 17 will show DDS's performance under different objectness thresholds). From these high-objectness regions, we apply two filters to remove those that are already in the DNN output on the low-quality video (Stream A). First, we filter out those regions that have over 30% IoU (intersection-over-union) overlap with the labeled bounding boxes returned by DNN on the low-quality video. We empirically pick 30% because it works well on all the videos



(a) Bounding-box-based tasks **(b) Pixel-based tasks**

Figure 7: *Illustration on how DDS generates feedback regions on two types of applications.*

in our dataset. Second, we remove regions that are over 4% of the frame size (roughly 20% of each dimension), because we empirically find that if an object is large, the DNN should have successfully detected it. The remaining region proposals (bounding boxes) are used as feedback regions.

Figure 7a shows an example: there are nine bounding boxes in high-objectness-score results, three of which overlap with inference results in Stream A and one of which is too large. The remaining five regions are the feedback regions.

Semantic segmentation (based on pixels): Semantic segmentation DNNs assign each pixel a class label (see Figure 1), and in addition, they also give a score of each class for each pixel (the class with the highest score is the class label). We first assign a score of $1 + max' - max$ to each pixel, where max is the largest score among classes of interest and max' is the second largest. Intuitively, the higher the score is, the more "indecisive" the DNN is about which class a pixel belongs to. We then pick the feedback regions by creating k rectangles that cover as many high-score pixels as possible. We repeatedly pick the $n \times n$ rectangle in which the pixels have maximum average score and zero out the scores of corresponding pixels until we found k rectangles. The values of n and k control the total number of pixels in the feedback regions ($k \cdot n^2$). We use $n = 32$ and $k = 16$, though we do not claim them to be optimal values. (Figure 18 shows the performance under different k values.) Figure 7b shows an example and the selection of one feedback region. We can see that the high-score pixels typically lie at the boundaries of objects.

We notice three properties of the above logic.

- First, the feedback regions are likely in the DNN output but not yet in DNN's output on low-quality video. This provides useful clue about where video quality should be increased in Stream B.
- Second, to save bandwidth of Stream B, the feedback regions are created with shapes that can be efficiently encoded. Thus, they are different from any direct (intermediate/final) output of DNNs (*e.g.,* region proposals).
- Third, the algorithms to extract feedback regions only assume the format of the DNN output, rather than particular DNN architectures or parameters.

3.3 Handling bandwidth variation

Like other video streaming protocols, DDS must adapt its bandwidth usage to handle fluctuations in available bandwidth. There are several effective control knobs that affect the bandwidth usage of DDS. However, we empirically find that these knobs affect the

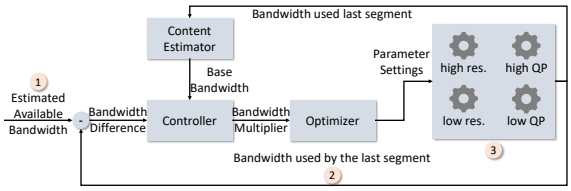


Figure 8: *DDS's adaptive feedback control system dynamically tunes the low and high quality configurations based on the difference between the estimated available bandwidth for the next segment and that used for the previous segment.*

bandwidth-accuracy tradeoff in a similar way (*i.e.,* on the same Pareto boundary; §5.4), so DDS only tunes low quality level and high quality level.

To tune the low and high quality levels, we implement a feedback control system (illustrated in Figure 8). Our controller is based on prior work that proposes a virtual, adaptive control system that can be customized for specific deployments [28, 60]. To instantiate this controller, DDS needs to specify three things: a bandwidth constraint to be met, feedback for monitoring bandwidth usage, and the tunable parameters that affect bandwidth usage. For DDS, the bandwidth constraint is the estimated available bandwidth for the next segment (labelled (1) in the figure), the feedback is the total bandwidth usage (for both low and high quality) from the last segment (2), and the tunable parameters are the resolution and quantization parameters (*i.e.,* the QP in Figure 8) of both the low and high quality (3). The controller continually estimates the *base* bandwidth usage; *i.e.,* the last segment's bandwidth usage if the *default* parameter settings had been used. The controller then takes this base behavior as well as the difference between the desired bandwidth constraint for the next segment and the achieved bandwidth usage for the previous segment and computes a scaling factor for the base bandwidth. This scaling factor is passed to an optimizer which finds the low and high quality settings that deliver the scaled bandwidth usage while maximizing F1 score.

DDS's dynamic bandwidth adaptation has several useful formal properties based on its use of feedback control.

First, the content estimator can handle *dynamic* video content which changes the relationship between the parameters and bandwidth usage. The adaptation mechanism uses a Kalman Filter to continually estimate the *base* bandwidth usage. Hence, when the video content changes, the control model—that captures the relationship between the parameters and bandwidth usage—will update itself, allowing DDS to capture unmodeled non-linearities in the relationship between quality settings and bandwidth use. Intuitively, we can think of the relationship between bandwidth usage and quality parameters as a curve and the base bandwidth (estimated by the Kalman filter) as a tangent to that curve. When adjusting the quality parameters, the DDS controller uses this tangent as a linear approximation to the true behavior. Using this formulation, the bandwidth usage converges to the bandwidth constraint in time proportional to the logarithm of the error in this estimation [60]. This adjustment technique provides robustness in the face of shifts and variations in the system including when there does not exist a single control model that captures all system dynamics [37].

Second, the optimizer finds the highest quality given the bandwidth usage specified by the controller. This optimality is achieved

by scheduling configurations over multiple segments. As the system has a small, constant number of constraints (simply respecting the bandwidth requirement), an optimal solution can be found in constant time [49].

3.4 Design rationale and performance analysis

Why driven by server? At first glance, the idea of server-driven region extraction seems similar to Vigil [80] and EAAR [54], which also identifie and send only regions likely with objects to the server. But we argue that the region extraction methods of Vigil and EAAR spend extra bandwidth on objects that can be detected at low quality, and they do not generalize to applications like semantic segmentation. Moreover, both of them do not leverage modern video codec to save bandwidth. Furthermore, Vigil's camera-side local model uses simpler feature extractor than the server-side DNN, and thus might miss objects when analyzing challenging video content (as illustrated in Figure 5). As we will show in §5.2, even if Vigil uses a model (MobileNet-SSD) that runs only 3× faster than the server-side DNN [20], it still misses about 40% more objects of interest than DDS and sends over 30% more data; EAAR consumes 4× bandwidth and still less accurate than DDS. §5.2 will give more analysis.

Analysis of DDS's network usage: The bandwidth usage of DDS is governed by two factors: (1) the quality levels of Stream A and Stream B, and (2) the areas of the feedback regions of Stream B. If Stream A uses a high quality level, the bandwidth usage will be dominated by Stream A and the feedback regions selected in Stream B will be less relevant. But if Stream A uses a very low quality level, DDS cannot extract meaningful feedback regions from the DNN output on the low-quality video. (§5.1 gives the detail configurations of Stream A and B.) The areas of feedback regions have a complex relationship with the video content. Intuitively, feedback regions will be smaller when less objects/pixels are associated with small objects or hard-to-classify boundaries. When feedback regions are so large that Stream B is almost the same size of the original video, then DDS will not save much bandwidth.

To use the analysis in §2.4, when the total cost of a video analytics system is dominated by the network cost (Setting 1), DDS will reach better cost-accuracy tradeoffs than the baselines, although it will do poorly when the cost is dominated by the server cost (Setting 2).

Delay analysis of DDS: One concern of DDS is the extra delay in Stream B. We introduce an optimization in §4.2 to reduce the average response delay by reporting the objects/pixels that are already detected in Stream A. This allows DDS to achieve a lower average response delay than the baselines at similar accuracy (see Figure 4a), since Stream A has a low response delay and many objects/pixels will not need Stream B.

4 IMPLEMENTATION

We implement DDS mostly in Python and the code is available and will be regularly updated in [8].

4.1 DDS Interface

DDS sits between the low-level functions (video codec and DNN inference) and the high-level applications (*e.g.,* object-detection queries). It provides "south-bound" APIs and "north-bound" APIs, both making minimum assumptions about the exact implementation of the low-level and high-level functions.

The south-bound APIs interact with the video codec and DNN. Our implementation uses the APIs already exposed by the x264 MPEG video, such as `x264_encoder_encode` [25]. From DNN, DDS implements two functions: (1) feedback regions, each with a specified location; and (2) detection results including the detected pixels/bounding boxes each with a specified location and a detection confidence score.

The north-bound APIs implement the same analyst-facing (north-bound) APIs as the DNNs (DDS can simply forward any function call to DNNs), so the high-level applications (*e.g.,* [51, 58]) do not need to change and DDS can be deployed transparently from the analysts' perspective. The only difference is that DDS runs the DNN twice on the same video segment, so the two DNN inference results must be merged into a single result, which is logically similar to how DNNs internally merge redundant results (*e.g.,* [73]).

4.2 Optimization

Saving bandwidth by leveraging codec: A naive implementation of Stream B would encode each feedback region as a separate high-quality image. But we found that the total size of these images would be much greater than the original video without cropping out the regions! The reason is that the video codecs (*e.g.,* H.264/H.265), after decades of optimization, are very effective in exploiting the spatial redundancies within a frame and the temporal redundancies between frames to reduce the encoded video size. DDS leverages such encoding effectiveness. It sets the pixels outside of the feedback regions in the high quality image to black (to remove spatial redundancies), and encodes these images into a video file (to remove temporal redundancies).

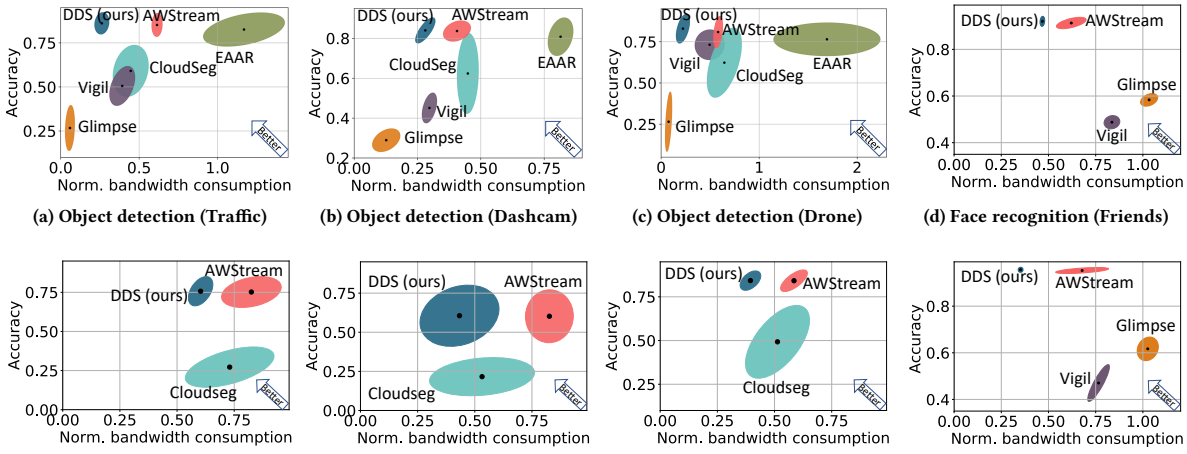
Reducing average delay via early reporting: The cost that DDS pays to get better performance is the worst-case response delay: the result of Stream B will wait for two rounds of inference before it can be returned. We leverage the observation that a substantial fraction of the DNN output from the low-quality video (Stream A) already has high confidence and thus can be returned without waiting for Stream B. While this optimization does not change the bandwidth consumption or worst-case response delay, it substantially reduces the delay of many inference results. In object detection, we empirically found that over 90% of all final detected objects could have been detected in Stream A. These objects can be returned much faster than any prior approach, because Stream A uses a quality level much lower than what other work (*e.g.,* [31, 32, 78]) would need to achieve the same accuracy. Similarly, in semantic segmentation, we found that the label of over 93% of all pixels can be returned by Stream A, without the need of Stream B.

Camera-side heuristics for fault tolerance: When the connection to the server is poor or the server is disconnected, DDS will leverage camera-side compute (if available). Like Glimpse [30], DDS can use a camera-side tracking logic to generate inference results on new frames based on the results of the previous frames.

5 EVALUATION

The key takeaways of our evaluation are:

- On three vision tasks, DDS achieves same or higher accuracy than the baselines while using 18-58% less bandwidth (Figure 9) and 25-65% lower average response delay (Figure 11).



(a) Object detection (Traffic) (b) Object detection (Dashcam) (c) Object detection (Drone) (d) Face recognition (Friends)

(e) Semantic segmentation (Traffic) (f) Sem. segmentation (Dashcam) (g) Semantic segmentation (Drone) (h) Face recognition (TBBT)

Figure 9: *The normalized bandwidth consumption v.s. inference accuracy of DDS and several baselines on various video genres and applications. DDS achieves high accuracy with 55% bandwidth savings on object detection and 42% on semantic segmentation, and 36% on face recognition. Ellipses show the 1-σ range of results.*

Name	*Vision tasks*	*Total length*	*# videos*	*# objs/IDs*
Traffic	obj detect / segment	2331s	7	24789
Drone	obj detect / segment	163s	13	41678
Dashcam	obj detect / segment	5361s	9	24691
Friends	face recog	6000s	10	15022
TBBT	face recog	6000s	10	12109

Table 1: *Summary of our datasets.*

- DDS sees even more improvements on certain video genres where objects are small (Figure 12) and on applications where the specific target objects appear rarely (Figure 13).
- DDS's gains remain substantial under various bandwidth budgets (Figure 14) and bandwidth fluctuation (Figure 16).
- DDS poses limited additional compute overhead on both the camera and the server (Figure 19).

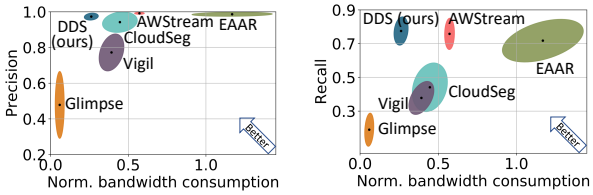
5.1 Methodology

Experiment setup: We build an emulator of video streaming that can measure the exact analytics accuracy and bandwidth usage. Although existing video analytics platforms might support DDS, we implement and test DDS and all baselines in our emulator for a fair comparison. It consists of a client (camera) that encodes/decodes locally stored videos and a fully functional server that runs any given DNN and a separate video encoder/decoder. We run DNN inference on RTX 2080 super and all other computations on Intel Xeon Silver 4100. Unless stated otherwise, we use FasterRCNN-ResNet101 [68] as the server-side DNN for object detection, InsightFace [14, 34] for face recognition and FCN-ResNet101 [57] for semantic segmentation. As we will see in §5.3, different choices of DNNs will not qualitatively change the takeaways. When needed, we vary video quality along the quantization parameter (from {26,28,30,32,34,36,38,40}, we call it QP for short) and the resolution (from scale factors of {0.8,0.7,0.5}), and DDS uses 36 (QP) as low quality and 26 (QP) as high quality, with resolution scale set to 0.8 in object detection and 1.0 in semantic segmentation[5]. We do not consider the network cost of AWStream to profile the accuracy-bandwidth relationships under different QP-resolution combinations. This makes the AWStream bandwidth usage is strictly less than its actual one. In most graphs, we assume a stable network connection, but in §5.4, we will test DDS under different network bandwidth and latency and a few real network traces.

Datasets: To evaluate DDS over various video genres, we compile five video datasets each representing a real-world scenario (summarized in Table 1 and their links can be found in [4]). These videos are obtained from two public sources. First, we get videos from *aiskyeye* [23], a computer-vision benchmark designed to test DNN accuracies on drone videos. Nonetheless, DDS and the baselines can be affected by factors such as fraction of frames with objects of interest or size of the regions with objects. Therefore, we try to cover a range of values along these factors (including objects of various sizes and frames with various number of objects) by adding YouTube videos as follows. We search keywords (*e.g.,* "highway traffic video HD") in private browsing mode (to avoid personalization biases); among the top results, we manually remove the videos that are irrelevant (*e.g.,* news report that mentions traffic), and we download the remaining videos in their entirety or the first 10-minutes (if they exceed 10 minutes). The vision tasks are (1) to detect (or segment) vehicles in traffic and dashcam videos, (2) to detect humans in drone videos, and (3) to recognize identities in sitcom videos. Because many of these videos do not have human-annotated ground truth, for fairness, we use the DNN output on the full-size original video as the reference result to calculate accuracy. For instance, in object detection, the accuracy is defined by the F1 score with respect to the server-side DNN output in highest resolution (original) with over 30% confidence score.

Baselines: We use five baselines to represent two state-of-the-art techniques (see §2.3): camera-side heuristics (Glimpse [30], Vigil [80], EAAR [54]) and adaptive streaming (AWStream [78], CloudSeg [76]). We made a few minor modifications to ensure the comparison is fair. First, all baselines and DDS use the same

[5]We use full resolution in semantic segmentation to keep the same number of labeled pixels as in the ground truth which assigns each pixel a class label.



(a) Precision on traffic video (b) Recall on traffic video

Figure 10: *The normalized bandwidth consumption v.s. precision and recall of DDS and several baselines on traffic camera.*

server-side DNN. Second, although DDS needs no more camera-side compute power than encoding, camera-side heuristics baselines are given sufficient compute resource to run more advanced tracking [40] and object detection algorithm [69] than what Glimpse and Vigil[6] originally used, so these baselines' performance is strictly better than their original designs. Third, all DNNs used in baselines and DDS are pre-trained (*i.e.,* not transfer-learned with samples from the test dataset); In particular, our implementation of CloudSeg uses the pre-trained super-resolution model [26] from the website [18]. This ensures the gains are due to the video streaming algorithm, not due to DNN fine-tuning, and it also helps reproducibility. Finally, although DDS could lower the frame rate, to ensure that the accuracies are always calculated on the same set of images, we do not sample frames and only vary the resolution and QP in DDS.

Performance metrics: We use the definition of accuracy and average response delay from §2.1. To avoid impact of video content on bandwidth usage, we report bandwidth usages of DDS and the baselines after normalizing them against the bandwidth usage of each original video.

5.2 End-to-end improvements

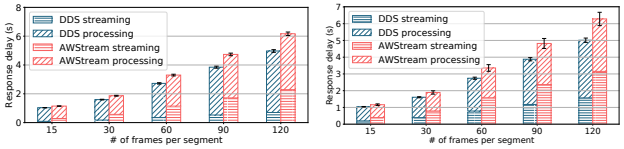
We start with DDS's overall performance gains over the baselines along bandwidth savings, accuracy, and average response delay.

Bandwidth saving: Figure 9 compares the bandwidth-accuracy tradeoffs of DDS with those of the baselines. In each application, we use a fixed DDS configuration and normalize the bandwidth usage against the size of the highest-quality videos (which we use to derive the ground truth). We also lower the frame rate to 1 FPS to speed up the experiments and confirm on a randomly sampled set of videos that the 1 FPS optimization has minimal impact on DDS's relative bandwidth savings and accuracy gains. Across three vision tasks, DDS achieves higher or comparable accuracy than AWStream but uses 55% less bandwidth in object detection and 42% in semantic segmentation. Glimpse sometimes uses less bandwidth but has much lower accuracy. Vigil, Glimpse, CloudSeg and EAAR consumes more bandwidth than DDS with lower accuracy. Overall, even if DDS is less accurate or uses more bandwidth, it always has an overwhelming gain on the other metric.

We explain this result from two perspectives.

- *Precision and recall:* Figure 10 corroborates our intuition (§2.5) that the camera-side heuristics, used by Vigil, Glimpse and EAAR to select frames/regions, are limited by camera-side compute power and thus tend to either miss objects (as illustrated in Figure 5) and produce spurious objects (*e.g.,* Vigial and Glimpse), or use too much bandwidth to achieve decent accuracy (*e.g.,* EAAR). We notice that CloudSeg has a lower recall than AWStream (whereas the original paper shows otherwise on a different dataset). We speculate that this is because we use a pre-trained super-resolution model (in consistent with the implementation of other baselines), whereas the original paper fine-tunes the super-resolution model to the dataset. DDS has a higher precision because it uses the output of the server-side DNN model; it achieves high recall because it re-examines the uncertain regions from the low-quality video.
- *Encoding:* Glimpse, Vigil, and EAAR all send individual frames separately, whereas DDS uses a video codec to leverage the temporal similarities across frames. Moreover, AWStream and CloudSeg do use video codec, but they do not leverage the non-uniform quality distribution, and DDS (Stream B) only encodes the difficult-to-detect regions/pixels at a higher quality.

[6]Our implementation of Vigil does not include the optimization of setting the background pixels in same RGB color, but in a separate experiment, we find that on the object detection videos (Table 1), this optimization only reduces Vigil's bandwidth usage by 10-20% and leads to similarly low accuracy.



(a) Object detection (b) Semantic segmentation

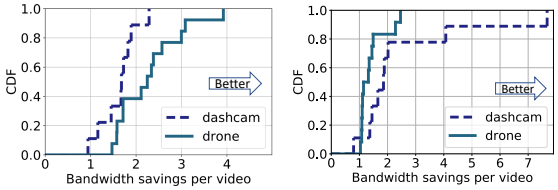
Figure 11: *Response delay of DDS is consistently lower than AWStream under various lengths of video segment.*

Response delay: Figure 11 shows the response delay of DDS and AWStream (the baseline whose accuracy is the closest to DDS) with the same length of a segment (number of consecutive frames encoded as a segment before sent to the server). In this experiment, we use a fixed bandwidth at the bitrate of the highest-quality video, which we use to derive the ground truth. Note that we exclude the buffering delay for the camera to accumulate the frames in each segment (which is the same between AWStream and DDS) as well as the delay for DDS to concatenate the pixel matrices in Stream A and Stream B (which can be sped up by standard libraries, such as openCL). For the same segment length, we see that DDS reduces the average response delay by about 5-25% compared with AWStream, despite that DDS needs two iterations per frame. This is because DDS detects most of the objects in Stream A whereas AWStream sends a single video stream and spends more time to transmit that stream through the network. To put it into perspective, popular video sites use 4-second to 8-second segments [81] (*i.e.,* over 120 frames per segment), a range in which DDS's gains are considerable.

5.3 Sensitivity to application settings

Impact of video genres: Next, Figure 12 shows the distribution of per-video bandwidth savings with respect to AWStream (dividing AWStream's bandwidth usage by DDS's when DDS's accuracy is at least as high as AWStream) in three datasets. There is substantial performance variability, even among the videos of the same type. This is because DDS's gains depend on the size of objects missed by the low-quality encoding, which varies with videos (§3.4).

That said, the impact of content on performance gains can be task-dependent. For instance, in object detection, the dashcam videos show less improvement than the other two datasets, but



(a) Object detection (b) Semantic segmentation

Figure 12: *Distributions of per-video bandwidth savings in two datasets. The gains of DDS are video dependent.*

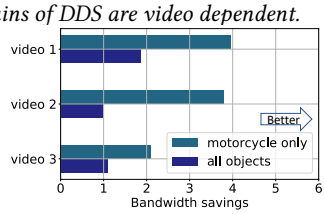


Figure 13: *Segmentation on only motorcycles achieves 2-4× more bandwidth savings than segmentation on all classes.*

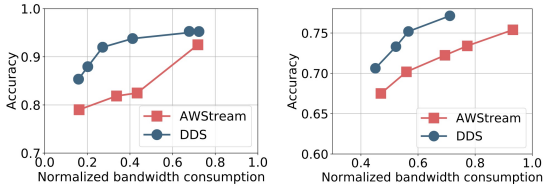
in semantic segmentation, the dashcam videos show the most improvement! This contrast highlights the difference between the two tasks. In object detection, when an object is not confidently classified, DDS will send an entire bounding box to the server. However, semantic segmentation typically sends the pixels at the boundary of objects, whose size is less affected by the size of objects. Since the dashcam videos have more large objects, they show more gains from DDS when the task is semantic segmentation.

Impact of the targeted objects: So far we have evaluated DDS when the target object classes appear frequently in video, but an advantage of DDS is that it saves more bandwidth when the server-side DNN only detects particular objects and these objects appear less frequently. To show it, we change the segmentation task from detecting all objects to detecting only motorcyles. Figure 13 shows the DDS's bandwidth savings (when achieving same or higher accuracy than AWStream) on three traffic videos in which motorcyles do appear but only in a small fraction of frames, and compare the bandwidth savings with those when the task is over all classes. DDS's gains are more significant when only motorcyles are the target objects, and the gains are higher when the motorcyles take a smaller fraction of pixels and frames (*e.g.,* Video 2).

Impact of DNN architecture: Last but not least, we test different DNN architectures on a randomly-selected traffic video (5-minute long). We find that DDS achieves substantial bandwidth savings under different server-side DNN models: FasterRCNN-ResNet101 (44%) and FasterRCNN-ResNet50 [20] (54%) has the same architecture but different feature extractors, while Yolo [66] (51%) uses a different architecture and feature extractor (§3.2). This implies that the benefit of DDS is agnostic to the server-side DNN architecture. We leave a full examination of different DNN architectures (*e.g.,* MaskRCNN [39]) to future work.

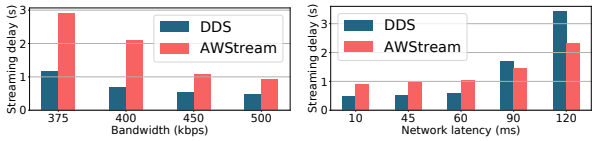
5.4 Sensitivity to network settings

Accuracy vs. available bandwidth: We then vary the available bandwidth and compare DDS with AWStream, which is performance-wise the closest baseline. Figure 14 shows that given different available bandwidth, DDS can adapt its configurations to cope with



(a) Object detection (b) Semantic segmentation

Figure 14: *DDS outperforms AWStream (the closest baseline) in accuracy under various bandwidth consumption budgets.*



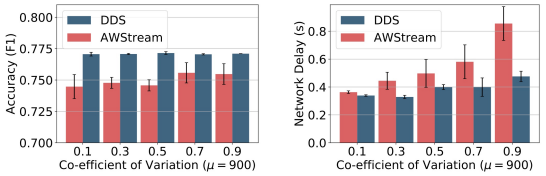
(a) Bandwidth vs. streaming delay (b) Latency vs. streaming delay

Figure 15: *The response delay of AWStream, camera-only approach and DDS with respect to different network bandwidth and network latency. DDS is more sensitive to network latency.*

them while achieving higher accuracy. We notice that the accuracy of semantic segmentation is lower than that of object detection (*i.e.,* the segmentation model is more sensitive to quality degradation). We speculate that this is because the segmentation accuracy is highly sensitive to the pixels around the object edges, which tend to be modified with slight quality degradation (this effect seems more pronounced when small objects are close to each other).

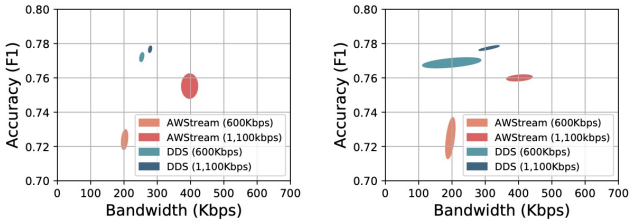
Impact of network bandwidth and latency: Figure 15a shows the impact of varying network bandwidth (while keeping the network latency at 10ms) on the streaming delay of DDS and AWStream. We ensure that the accuracy of DDS is always higher than the accuracy of AWStream. We use Linux netem to vary the network bandwidth and latency. We see that DDS has lower streaming delay (*i.e.,* less time is spent on the network) than AWStream. This is because DDS sends less data over the network than AWStream and many objects/pixels need only one iteration to be detected and classified. Similarly, Figure 15b shows the impact of network latency (while keeping the bandwidth at 500kbps) on the streaming delay of DDS and AWStream. We see that when latency is over a threshold (~90ms in this experiment), DDS has higher streaming delay than AWStream. This is because for those objects detected in Stream B, they experience two iterations, which makes the streaming delay of DDS more sensitive to long network latency than AWStream.

Impact of bandwidth variance: Figure 16 compares DDS with AWStream under an increasing bandwidth variance. We use synthetic network bandwidth traces to evaluate the impact of bandwidth variance in a controlled manner. The available bandwidth of this trace is drawn from a normal distribution of $900 \cdot N(1, \sigma^2)$Kbps while we increase σ from 0.1 to 0.9. We observe that DDS maintains a higher accuracy than AWStream. Although DDS and AWStream use the same bandwidth estimator (average of the last two segments), DDS uses the available bandwidth more efficiently because DDS's feedback control system continually adapts the model configuration parameters to bandwidth. Thus, DDS adaptively selects the best possible configuration parameters at each time instant.



(a) Accuracy (b) Network delay

Figure 16: *DDS can handle bandwidth variance and maintain a sizeable gain over the baseline of AWStream even under substantial bandwidth fluctuation.*



(a) Network trace 1 (b) Network trace 2

Figure 17: *Impact of available bandwidth on performance under re-scaled real network trace.*

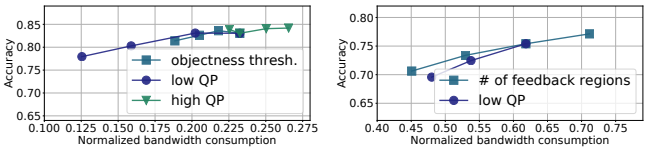
Even when the variance in available bandwidth is high ($\sigma > 0.7$), DDS maintains a relatively low response delay while AWStream's delay increases.

Under real network traces: Next, Figure 17 evaluates DDS against AWStream on two real network traces [1]. Since the available bandwidth in the traces on average exceeds the bandwidth needed to stream the original video, we stress-test DDS by scaling the available bandwidth by a constant factor to mimic settings where multiple cameras share the bottleneck bandwidth (TCP-induced variances are ignored). In particular, we scale the average bandwidth of trace to 1,100kbps and 600kbps, while retaining the relative bandwidth variance in the trace. We can see DDS consistently achieves higher accuracy under different mean available bandwidth.

Impact of parameter settings: Figure 18 shows the impact of key parameters of DDS on its accuracy/bandwidth tradeoffs: the QP in Stream A ("low QP"), the QP in Stream B ("high QP"), the objectness threshold, and the number of feedback regions (*i.e.,* the k introduced in §3.2). We vary one parameter at a time and test them on the same set of traffic videos. The figures show that by varying these parameters, we can flexibly trade accuracy for bandwidth usage. Overall, their effect roughly falls on the same Pareto boundary, so there may not be significant difference between the choices of parameters to vary when coping with bandwidth fluctuation.

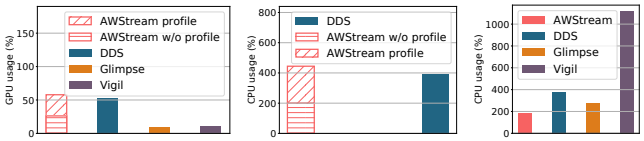
5.5 System microbenchmarks

Camera-side and server-side overheads: Figure 19 compares the systems overheads of DDS with the baselines. We benchmark their performance on our server, with one RTX 2080 Super and one 16-core Intel Xeon Silver 4100. We scale the CPU and GPU usage by normalizing their runtime (*e.g.,* 2x runtime on the same number of fully used CPUs will be translated to 2x more CPU usage). Figure 19 shows that compared to AWStream (when the profiling cost is excluded), DDS has 2x more overheads at both client-side



(a) Object detection (b) Semantic segmentation

Figure 18: *Sensitivity analysis of DDS's parameters. By varying these parameters, DDS can flexibly adapt itself to reach desirable accuracy for a given bandwidth constraint.*



(a) Server GPU (b) Server CPU (c) Client CPU

Figure 19: *Compared to prior solutions, DDS has low additional systems overhead on both client and server.*

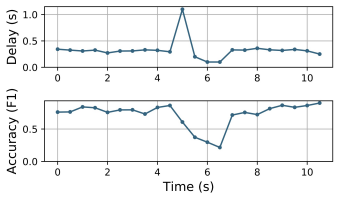
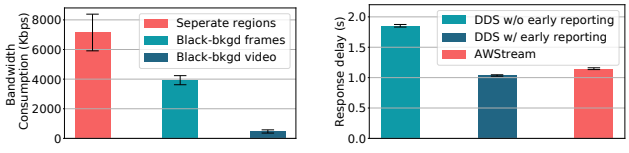


Figure 20: *DDS can handle server disconnection (or server failure) gracefully by falling back to client-side logic*

CPU (the CPU usage may exceed 100% since it may leverage more than one CPU cores), server-side CPU and server-side GPU. This is because DDS invokes extra encoding, decoding and inference costs in Stream B. However, the profiling cost of AWStream is a substantial server-side CPU and GPU costs. We estimate it by profiling 30 configurations (compared to 216 claimed in [78]) over a 10-second video every 4 minutes (which is much less often than profiling a 30-second video every 2 minutes as used in [78]), the server-side CPU and GPU costs of AWStream have already become higher than DDS. That said, we acknowledge that if AWStream updates the profile less frequently (*e.g.,* every tens of minutes), its GPU usage could be lower than DDS, but that might cause its profile to be out of date and less accurate. On the server side, both Vigil and Glimpse incur minimal CPU overheads (since they do not need to decode the video) and much less GPU overheads than DDS and AWStream (since their camera-side logics reduce the need for server-side inference).

Fault tolerance: We stress test DDS with temporary server-side disconnection. By default, the camera runs DDS protocol, and it also has a local object tracking algorithm as a backup. Figure 20 shows the time-series of response delay and accuracy. First, DDS maintains a desirable accuracy, but at $t = 5$ second, the server is disconnected. We see that DDS waits for a short time (until server times out at $t = 5.5$) and falls back to tracking the last detection results from the server DNN. This allows for a graceful degradation in accuracy, rather than crashing or delaying the inference indefinitely. Between the server disconnection and the timeout, the segments will be placed in a queue, and when the local inference begins, the queue



(a) Reducing bandwidth usage by smarter encoding **(b) Reducing response delay by early reporting**

Figure 21: *System refinements to (a) reduce Stream B bandwidth and (b) reduce response delay (both of them are introduced in §4.2).*

will be gradually cleaned up. When the server is back online (at $t = 13$), the camera will be notified with at most a segment-worth of delay (0.5 second), and begin to use the regular DDS protocol to resume video analytics. Meanwhile the camera will continue to send a liveness probe every video segment.

Performance optimization: Figure 21 examines two performance refinements. First, figure 21(a) shows that (1) putting the proposed regions on a black background frame yields about 2× bandwidth savings over encoding each region separately; and (2) compressing these frames in an mp4-format leads to another 10× bandwidth savings. Second, figure 21(b) shows that returning the first-iteration output (*i.e.,* the high-confidence results in Stream A before Stream B starts), we reduce the average response delay by about ∼ 40%.

6 RELATED WORK

We discuss the most closely related work in three categories.

Video analytics systems: The need to scale video analytics has sparked much systems research: DNN sharing (*e.g.,* [42, 44]), resource allocation (*e.g.,* [52, 79]), vision model cascades (*e.g.,* [48, 71]), efficient execution frameworks (*e.g.,* [51, 58, 64]), as well as camera/edge/cloud collaboration (*e.g.,* [30, 32, 54, 63, 72, 76, 78, 80], see §2.3 for a detailed discussion) or multi-camera collaboration (*e.g.,* [43, 45]). The most related work to DDS is AWStream [78] which shares with DDS the ethos of using a server DNN-generated profile. The key distinction is that such feedback is not real-time video content, so it cannot zoom in on specific regions on the current frames. Vigil [80] sends cropped regions, but it is bottlenecked by the camera computing power (See §2.3). DDS shares the concept of server-driven streaming with its own preliminary design [63] and the partially server-driven solutions [30, 54]. But it is the first solution that achieves high accuracy (by correcting objects mislabeled *and* missed by the low-quality video or the camera-side model) in multiple vision tasks and DNN models, and fully utilizes the video streaming codec to minimize bandwidth usage.

Vision applications: Computer vision and deep learning have a substantial body of research (*e.g.,* [34, 47, 56, 65, 67, 68, 77]). Recent works on video object detection show it is inefficient to apply object detection DNN frame by frame; instead it should be augmented by tracking [41] (similar to §4.2) or by a temporal model such as LSTM [53, 55]. This work complements DDS by designing new, server-side deep learning models. DDS's distinctive advantage is that it explicitly optimizes the bandwidth/accuracy tradeoffs in a way that is largely agnostic to the server-side DNN.

Internet video encoding/streaming: Recent innovations in video encoding have provided better compression gains (*e.g.,* [33]). The closest efforts to DDS are scalable video coding [62] and region-of-interest (ROI) encoding [59]. However, these approaches optimize human-perceived quality. Scalable video coding can utilize the bandwidth more efficiently than traditional encoding methods, but it still compresses video uniformly in its entirety. ROI encoding requires the viewer to specify the region of interest, and a recent proposal [54] uses the region-proposal network (RPN) to generate ROI regions. Much work on adaptive bitrate streaming (*e.g.,* [35, 46, 50, 70]) has focused on adapting the bitrate of precoded video chunks to bandwidth fluctuation, but less on adapting encoding to the dynamic video content as DDS does.

7 LIMITATIONS AND DISCUSSION

Strict server-side resource budget: In some sense, DDS reduces bandwidth usage at the expense of relying on server-side DNN to run inference more than once per frame. Similar tradeoffs can be found in AWStream, which triggers costly reprofiling periodically, and CloudSeg, which enforces an upfront super-resolution model customization process. All these techniques may not be directly applicable where server resources have strict budgets or GPU cost is proportional to its usage (*e.g.,* cloud instances).

Implication to privacy: Privacy is an emerging concern in video analytics [75]. While DDS does not explicitly preserve privacy, it is amenable to privacy-preserving techniques. Since DDS does not send out full resolution video, it could be repurposed to denature videos before sending only a part of the video to the server.

Edge AI accelerators: Though DDS makes minimal assumption on camera's local computation capacity, it benefits from the trend of more accelerators being added on edge devices, by using camera-side heuristics as described in §4.2. We also envision DDS running alongside the camera local analytics to share the workload and provide higher inference accuracy with minimal bandwidth overheads.

8 CONCLUSION

Video streaming has been a driving application of networking research. This work argues that the emerging AI applications inspire a paradigm shift away from the traditional source-driven approach to video streaming. We have developed a concrete DNN-driven design, called DDS, that exploits opportunities unique to deep learning applications: (1) unlike user QoE, video inference accuracy depends less on pixels than on what is in the video, and (2) deep learning models offers extra information that helps us decide how video should be encoded/streamed. We believe that the development of such video streaming protocols will significantly impact not only video analytics, but also the future analytics stack of many distributed AI applications.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Paolo Costa. In this project, Junchen Jiang and Kuntai Du are supported by NSF (CNS-1901466). Junchen Jiang is also supported by Google Faculty Research Award. Moreover, Ahsan Pervaiz and Henry Hoffmann are supported by NSF (CCF-1439156, CNS-1526304, CCF-1823032, CNS-1764039), ARO (W911NF1920321), DOE (DESC0014195 0003), and DARPA BRASS program.

REFERENCES

[1] 4g/lte bandwidth logs. http://users.ugent.be/~jvdrhoof/dataset-4g/.

[2] Are we ready for ai-powered security cameras? https://thenewstack.io/are-we-ready-for-ai-powered-security-cameras/.

[3] At&t unlimited data plans with talk & text. https://www.att.com/plans/unlimited-data-plans/. (Accessed on 06/15/2020).

[4] Benchmarking videos used in dds. https://github.com/KuntaiDu/dds.

[5] Can 30,000 cameras help solve chicago's crime problem? https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html.

[6] Cloud-based video analytics as a service of 2018. https://www.asmag.com/showpost/27143.aspx.

[7] Dashjs. https://github.com/Dash-Industry-Forum/dash.js.

[8] Dds: Machine-centric video streaming. https://github.com/KuntaiDu/dds.

[9] Faster r-cnn on jetson tx2. https://jkjung-avt.github.io/faster-rcnn/. Accessed: 7/3/2020.

[10] Fastest wireless network in 2020: We tested 8 carriers to crown a winner | tom's guide. https://www.tomsguide.com/us/best-mobile-network,review-2942.html. (Accessed on 06/15/2020).

[11] Gpus pricing | compute engine documentation | google cloud. https://cloud.google.com/compute/gpus-pricing. (Accessed on 06/21/2020).

[12] How ai based video analytics is benefiting retail industry. https://www.lanner-america.com/blog/ai-based-video-analytics-benefiting-retail-industry/.

[13] Hp r0w29a tesla t4 graphic card - 1 gpus - 16 gb. https://www.amazon.com/HP-R0W29A-Tesla-Graphic-Card/dp/B07PGY6QPT/. Accessed: 2020-1-29.

[14] Insightface: 2d and 3d face analysis project. https://github.com/deepinsight/insightface.

[15] Jetson tx2. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/. Accessed: 7/3/2020.

[16] Nvidia deep learning inference technical overview, table "jetson tx2 (maxq mode)" at row resnet50 batch size 128. http://www.nextplatform.com/wp-content/uploads/2018/01/inference-technical-overview-1.pdf. Accessed: 7/3/2020.

[17] Nvidia tesla deep learning product performance (table "t4 inference performance" at row resnet50 batch size 128). https://developer.nvidia.com/deep-learning-performance-training-inference. Accessed: 7/3/2020.

[18] Official implementation of efficient cascading residual network for sr. https://github.com/nmhkahn/CARN-pytorch.

[19] Smraza raspberry pi 4 camera module 5 megapixels 1080p. https://www.amazon.com/Smraza-Raspberry-Megapixels-Adjustable-Fish-Eye/dp/B07L2SY756/. Accessed: 2020-1-29.

[20] Tensorflow detection model zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[21] Video meets the internet of things. https://www.mckinsey.com/industries/high-tech/our-insights/video-meets-the-internet-of-things.

[22] Video surveillance: How technology and the cloud is disrupting the market. https://cdn.ihs.com/www/pdf/IHS-Markit-Technology-Video-surveillance.pdf.

[23] Vision meets drones: A challenge. http://www.aiskyeye.com/.

[24] Wi-fi vs. cellular: Which is better for iot? https://www.verypossible.com/blog/wi-fi-vs-cellular-which-is-better-for-iot.

[25] x264 open source video lan. https://www.videolan.org/developers/x264.html.

[26] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 252–268, 2018.

[27] Ganesh Ananthanarayanan, Victor Bahl, Peter Bodik, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath Sivalingam, and Sudipta Sinha. Real-time video analytics - the killer app for edge computing. *IEEE Computer*, October 2017.

[28] S. Barati, F. A. Bartha, S. Biswas, R. Cartwright, A. Duracz, D. Fussell, H. Hoffmann, C. Imes, J. Miller, N. Mishra, Arvind, D. Nguyen, K. V. Palem, Y. Pei, K. Pingali, R. Sai, A. Wright, Y. Yang, and S. Zhang. Proteus: Language and runtime support for self-adaptive software development. *IEEE Software*, 36(2):73–82, March 2019.

[29] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*, 2020.

[30] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168. ACM, 2015.

[31] Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. Adascale: Towards real-time video object detection using adaptive scaling. *arXiv preprint arXiv:1902.02910*, 2019.

[32] Sandeep P Chinchali, Eyal Cidon, Evgenya Pergament, Tianshu Chu, and Sachin Katti. Neural networks meet physical networks: Distributed inference between edge devices and the cloud. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2018.

[33] High Efficiency Video Coding and ITUT Rec. H. 265 and iso, 2013.

[34] Jiankang Deng, Jia Guo, Xue Niannan, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *CVPR*, 2019.

[35] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 362–373. ACM, 2011.

[36] John Emmons, Sadjad Fouladi, Ganesh Ananthanarayanan, Shivaram Venkataraman, Silvio Savarese, and Keith Winstein. Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 27–32, 2019.

[37] Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 299–310, New York, NY, USA, 2014. ACM.

[38] Shilpa George, Junjue Wang, Mihir Bala, Thomas Eiszler, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards drone-sourced live video analytics for the construction industry. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 3–8. ACM, 2019.

[39] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[40] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):583–596, 2014.

[41] Congrui Hetang, Hongwei Qin, Shaohui Liu, and Junjie Yan. Impression network for video object detection. https://arxiv.org/pdf/1712.05896.pdf, Dec 2017.

[42] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 115–131. IEEE, 2018.

[43] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph E. Gonzalez. Scaling Video Analytics Systems to Large Camera Deployments. In *ACM HotMobile*, 2019.

[44] Angela H Jiang, Daniel L-K Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, David G Andersen, and Gregory R Ganger. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 29–42, 2018.

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

[46] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (ToN)*, 22(1):326–340, 2014.

[47] Kinjal A Joshi and Darshak G Thakore. A survey on moving object detection and tracking in video surveillance system. *International Journal of Soft Computing and Engineering*, 2(3):44–48, 2012.

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

[49] D. H. K. Kim, C. Imes, and H. Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *ICCPS*, 2015.

[50] S Shunmuga Krishnan and Ramesh K Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, 2013.

[51] Sanjay Krishnan, Adam Dziedzic, and Aaron J Elmore. Deeplens: Towards a visual data management system. *arXiv preprint arXiv:1812.07607*, 2018.

[52] Robert LiKamWa and Lin Zhong. Starfish: Efficient concurrency support for computer vision applications. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 213–226. ACM, 2015.

[53] Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7083–7093, 2019.

[54] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[55] Mason Liu and Menglong Zhu. Mobile video object detection with temporally-aware feature maps. https://arxiv.org/pdf/1711.06368.pdf, Mar 2018.

[56] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, pages 6738–6746, 2017.

[57] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[58] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 57–70. ACM, 2016.

[59] Marwa Meddeb. *Region-of-interest-based video coding for video conference applications*. PhD thesis, Telecom ParisTech, 2016.
[60] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. CALOREE: learning control for predictable latency and low energy. In *ASPLOS*, 2018.
[61] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.
[62] J-R Ohm. Advances in scalable video coding. *Proceedings of the IEEE*, 93(1):42–56, 2005.
[63] Chrisma Pakha, Aakanksha Chowdhery, and Junchen Jiang. Reinventing video streaming for distributed vision analytics. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA, July 2018. USENIX Association.
[64] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. Scanner: Efficient video analysis at scale. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.
[65] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, pages 779–788, 2016.
[66] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.
[67] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
[68] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, 2017.
[69] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.
[70] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hossfeld, and Phuoc Tran-Gia. A survey on quality of experience of http adaptive streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2015.
[71] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. Fast video classification via adaptive cascading of deep models. *arXiv preprint*, 2017.
[72] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 328–339. IEEE, 2017.
[73] Jilin Tu, Ana Del Amo, Yi Xu, Li Guari, Mingching Chang, and Thomas Sebastian. A fuzzy bounding box merging technique for moving object detection. In *2012 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 1–6. IEEE, 2012.
[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
[75] Han Wang, Yuan Hong, Yu Kong, and Jaideep Vaidya. Publishing video data with indistinguishable objects. In *Proceedings of the 22nd International Conference on Extending Database Technology (EDBT)*, 2020.
[76] Yiding Wang, Weiyan Wang, Junxue Zhang, Junchen Jiang, and Kai Chen. Bridging the edge-cloud barrier for real-time advanced vision analytics. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.
[77] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.
[78] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 236–252. ACM, 2018.
[79] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, volume 9, page 1, 2017.
[80] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 426–438. ACM, 2015.
[81] Tong Zhang, Fengyuan Ren, Wenxue Cheng, Xiaohui Luo, Ran Shu, and Xiaolan Liu. Modeling and analyzing the influence of chunk size variation on bitrate adaptation in dash. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.