Reinventing Video Streaming for Distributed Vision Analytics

Chrisma Pakha[*], *Aakanksha Chowdhery*[+], *Junchen Jiang*[*#]
[*]*University of Chicago,* [+]*Google/Princeton,* [#]*Microsoft*

Abstract

Driven by the ubiquity of camera-equipped devices and the prohibitive cost of modern vision techniques, we see a growing need for a *custom video streaming protocol* that streams videos from cameras to cloud servers to perform neural-network-based video analytics. In the past decade, numerous efforts have optimized video streaming protocols to provide better quality-of-experience to users. In this paper, we call upon this community to similarly develop custom streaming protocols for better analytics quality (accuracy) of vision analytics (deep neural networks). We highlight new opportunities to substantially improve the tradeoffs between bandwidth usage and inference accuracy. The key insight is that existing streaming protocols are essentially client (camera)-driven; in contrast, by letting the analytics server decide what/when to stream from the camera, the new protocols can directly optimize the inference accuracy while minimizing bandwidth usage. Preliminary evaluation shows that a simple protocol can reduce bandwidth consumption by 4-23x compared to traditional streaming protocols and other distributed video analytics pipelines while maintaining at least 95% inference accuracy.

1 Introduction

With the proliferation of connected cameras in smart homes, smart cities, industrial IoT, and wearable cameras, we are seeing an increased demand to generate real-time insights (e.g., traffic monitoring, video summarization, intruder detection) on the live video feeds of the cameras. Over the past decade, the accuracy of many vision tasks has improved dramatically by Deep Neural Networks (DNNs) [12, 22–24, 32] but at the high computational cost of running the vision pipelines. The resulting tradeoff between accuracy and computation complexity has necessitated *distributed processing pipelines*, in which complex inferences are offloaded to cloud servers, and (processed) videos are streamed between cameras and servers [25, 26, 34].

Prior work (e.g., [5, 10, 14]) has focused on offloading vision tasks from mobile to the cloud, but we argue that it has missed a crucial aspect that *the video streaming stack integrates the requirements of video analytics itself* (§2). Traditional encoding and streaming protocols (e.g., [17, 30, 31]) are designed to stream all frames at high resolution so that viewers can watch the video uninterrupted. Video analytics, in contrast, focus only on specific video segments with queried objects of interest. Thus, a streaming protocol is optimal as long as the frames with the queried objects of interest are sent at sufficiently high resolution. These requirements of vision tasks open up new opportunities to greatly reduce bandwidth usage while achieving high analytics accuracy.

We first set out to understand the full potential of customizing streaming protocols for vision analytics (§3). We show that the optimal streaming protocol (under idealized assumptions) could achieve almost 100% accuracy and save bandwidth by about 50-1000× over traditional streaming protocols and early examples of custom protocols [3, 36] (Figure 2). The insight underlying the unexploited potential is that the existing solutions rely only on client-side bandwidth-filtering techniques that are agnostic to the server-side DNN-based analytics logic.

Inspired by these observations, we then present a new *server-driven* framework that enables the server-side analytics logic to determine how the video should be streamed (§4). We demonstrate its practical benefits by first casting it into an active learning process and then showing that a simple design point can greatly outperform the existing streaming protocols from prior work; reducing bandwidth consumption by 4-23× with similar accuracy (Figure 5). We end the paper by highlighting open questions, and call upon the research community to develop custom streaming protocols for vision analytics to truly unleash its full potential (§5).

2 What's new about video streaming in vision analytics?

Vision research has made rapid advances leading to many new commercial applications that extract insights in real-time videos from ubiquitously available cameras. Some queries for traffic monitoring applications include 'count cars moving in a specific direction at an intersection,' or

'find all speeding cars on the highway.' Some queries for surveillance applications include 'detect intruders' or 'track an object of interest.' In general, video analytics queries require detecting and classifying the objects of interest (e.g., vehicles) in a sequence of video frames. We use object detection as a representative complex and fundamental vision task. E.g., pose estimation will require the machine to locate the human first.

Why videos need to be streamed? The accuracy of computer vision tasks has improved significantly with the advent of deep neural networks (DNNs) [12, 22–24, 32]. However, running DNNs on live video feeds incurs a high computational cost (e.g., running a standard DNN object detector on a 30 frame-per-second video requires one dedicated GPU [22]), and the camera clients capturing the video have limited computational ability. Therefore, the video needs to be streamed to cloud or edge servers (as shown in Figure 1) to offload the computationally intensive DNN-based vision tasks (e.g., [3, 5]).

Some pipelines rely on special client-side hardware (e.g., [19]) to extract image features (e.g., [2, 15]) or run the first few NN layers (e.g., [33]), and send these extracted features or intermediate results, instead of videos/frames, to the server. While deploying hardware accelerator or GPU on edge devices is practical, its high cost prohibits a scalable solution, especially if we need to deploy it in smart cities or on low-end mobile devices; even with GPU capability, most mobile devices do not run at a high frame rate (30 frames/sec). Instead, this work targets a more flexible setting where the server provides object detection as a service to which any camera can send video for analytics without a large bandwidth overhead and computational cost.

What's an ideal streaming protocol? A video streaming protocol can be seen as a function that applies some compression operation $p(\cdot)$ (e.g., resolution downsizing and frame sampling) on the original video v, and sends the compressed video $p(v)$ to the server.[1] In video analytics, the optimal video streaming protocol maximizes the inference accuracy while keeping the size of the compressed video below available bandwidth to keep the inference delay constant. Formally, it can be written as follows: given video content v, and available bandwidth w, the optimal protocol p^* is

$$p^* = \text{argmax}_{|p(v)| \leq w} F(p(v)) \quad (1)$$

where $F(p(v))$ is the inference accuracy of running the server-side NN model on $p(v)$. A video protocol can be "parameterized" by various *control knobs*:

- *Frame selection*.
- *Area cropping* (sending only a cropped area).

[1]Compressing videos without losing information critical to DNN inference is reminiscent of other signal processing settings, such as LPC compression used for speech recognition [21], which tries to retain only the signal elements necessary for correct speech recognition.

Figure 1: *Streaming videos for vision analytics.*

- *Resolution* of a selected frame (or area).
- *Compression level* of a selected frame (or area).

Why not classic video streaming protocols? Conventional video streaming (encoding standards, e.g., H.265 [17, 31], and protocols, e.g., RTMP [20], DASH [29, 30]) have been designed to maximize the user-perceived quality-of-experience (QoE). The viewing experience suffers if video resolution is low, the video re-buffers, or many frames are dropped. So, to ensure high user-perceived quality, these streaming protocols optimize the video resolution (or bitrate) within available bandwidth, avoid video stalling (rebuffering) events and excessive frame drops, and shorten the start-up delay.

Dropping frames or lowering the frame resolution does not affect video analytics in the same way as it affects user-perceived quality. Instead, the goal of video analytics is to maximize the inference accuracy. Higher frame resolution may have negligible or diminishing effect on accuracy, if the object can be detected at a lower resolution. Further, the DNN inference accuracy does not suffer when the video is "choppy" or the video stalls, as long as the dropped frames do not add new information or the missing information can be retrieved if needed. A straightforward idea is to send only key frames (I frames) of H.264 video from the client to server but it may achieve lower accuracy or higher bandwidth because the current encoding is agnostic to objects in the frame. Further, recent efforts to make video encoding aware of the objects/regions of interest (e.g., [18]) require that the client specifies the region-of-interest before encoding. In short, whether a video gives a user satisfying viewing experience, i.e., *human-perceived QoE*, is different from whether a video provides enough information for the inference of a deep neural network (or analytics algorithm in general), i.e., *DNN-perceived QoE*.

Why not existing video analytics pipelines? In the existing pipelines, the clients either use simple decision rules or a classifier to select a subset of frames to send to the server, on which the server runs DNNs. For example, Glimpse [3], designed for wearable devices, assumes that the server detects the objects, sends the bounding box back to the client, and the client tracks this object. Vigil [36] assumes the client has sufficient computational

capability to run object detection and prioritizes frames with more objects. NoScope [13] and MCDNN [11] use multiple specialized DNNs where a filter (a frame difference detector or a simpler inference model) sends only the frames that are likely to have objects to more complex NN models. We take Glimpse as an example to illustrate the limitation of relying on client-side heuristics. Glimpse selects the frames to send based on the pixel-level changes between frames, but it can arguably cause *false negatives* (e.g., an object of interest does appear but does not change enough pixels to trigger the client to send the frame), as well as *false positives* (e.g., color changes in background trigger the client to send many frames while the actual objects of interest are static).

To sum up, both traditional video streaming protocols and previous analytics pipelines are essentially *client-driven*; it is the client-side logic that determines how the video is compressed and streamed. This client-driven workflow, however, suffers from the fundamental limitation that, without any input from the server, it is difficult to predict precisely what the server needs, i.e., how much information is needed to encode all objects of interest. In other words, these approaches are oblivious to the server-side inference accuracy ($F(\cdot)$ in Eq. 1), and are thus unable to directly maximize the inference accuracy under given bandwidth constraints.

3 A case for a server-driven approach

In contrast, we make a case for *server-driven* approach, which maximizes the analytics accuracy by giving the server full (or partial) control over what to send from the client to the server. In this section, we empirically demonstrate the advantages of an optimal server-driven design over alternatives. We end the section with the key challenges toward designing a practical server-driven protocol, and the intuition why it is tractable.

3.1 Potential improvement

The distinctive feature of a server-driven streaming protocol is that the analytics server decides the operations (p in Eq. 1) that client uses to compress the video before sending it to the server. Because the server runs the DNN model, in theory it is capable of picking the operations that directly maximize the inference accuracy.

Here, we quantify the full potential of a server-driven streaming protocol using an *oracle* version. The oracle protocol (unrealistically) assumes that the server can access the original video and pick the optimal values for each of the control knobs–frame selection, area cropping, resolution, and compression level. That is, it selects the exact frames where the objects move, crops out only the spatial regions that contain objects, selects the minimal resolution at which the server-side logic can detect the object, etc.

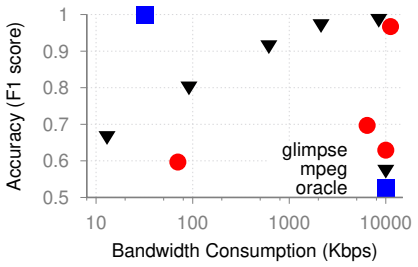

Figure 2: *The oracle protocol vs. streaming all encoded frames (MPEG) and client-side heuristics (Glimpse).*

Figure 2 shows the improvement by the oracle protocol using an example video which includes 13 vehicle objects. We compare the oracle protocol with two baselines: (1) streaming all frames pre-encoded by MPEG (with varying resolutions frome 348×128 to 1392×512, and quantization parameter from 18 to 51) to the server, and (2) Glimpse [3] (with varying degree of sensitivity at the frame different detector). Note that we use Glimpse as an example to illustrate the common limitation of client-driven heuristics shared by other similar pipelines. For fairness, we run the same server-side DNN logic in all experiments (see Table 1 for all setups), and the oracle protocol uses the same functional partition of Glimpse (i.e., client tracks all objects detected by the server). The accuracy metric could be application-dependent, we use F1 score (harmonic mean of inference precision and recall) as in [13, 35]. We can see that when all control knobs are set with the optimal values, the oracle protocol (the square point) saves bandwidth usage by about 250× while achieving almost 100% accuracy.

3.2 Challenge and opportunities

Implementing a server-driven protocol in practice, however, faces a fundamental dilemma. For the server to determine what to fetch from the client, it must first profile the NN's accuracy of the video under different configurations. In other words, the server is unable to decide what to fetch from the client, if it has never seen the original video, but sending the original video to the server negates the benefit of saving the bandwidth in the first place.

Is it possible to send the server a small amount of data that is just enough to allow the server to drive the protocol? The answer is positive. Our key insight is that *the inference result on low-quality video (e.g., low resolution, low frame rate, or aggressive compression), while not accurate, is sufficient to reveal what additional information is needed by the NN model to achieve higher accuracy*. It manifests itself in two concrete ideas.

Idea #1: Inference results on low-resolution frames can indicate where objects are likely to appear. For each detected object, the DNN-based models also returns a confidence value. Although inference results at low

Figure 3: *Inference results on low-resolution image (left) indicate where the protocol should "zoom-in" with high resolution (right) to improve the inference accuracy.*

resolution often have low confidence values, they can nevertheless indicate the areas where objects are likely to appear, so the protocol can "zoom-in" on these areas with more pixels to improve accuracy. Figure 3 illustrates it in action: on the low resolution image (left), we detect three objects, based on which we crop out three bounding boxes and run inference with high resolution (right), which reveals one of them is a false positive.

Idea #2: Inference results on sparsely sampled frames can indicate which frames are likely to contain objects. The difference between objects detected in two sampled frames can indicate whether the server needs to sample additional frames in between. For instance, if the server detects one object in the 1st frame and 3 objects in the 30th frame, this naturally suggests that there are frames with new objects that we have not sampled, and we need to sample additional frames in between.

4 Towards a practical design

We have seen that a server-driven streaming protocol enables new opportunities for bandwidth savings as well as accuracy improvement in vision analytics. This section formalizes the server-driven protocol, and presents *SimpleProto*, a concrete design that achieves substantial improvement in practice.

4.1 Formalizing server-driven protocol

Iterative workflow using superposition coding: The ideas in §3.2 inspire an interactive workflow which optimizes server-side inference accuracy by sending information *incrementally*: the client first sends a base-quality video (e.g., low resolution or frames sampled sparsely in time) to the server, and if the inference output is not confident, the server decides to get additional information so that it can recover higher-quality video by "adding" the new data to the base-quality video it already has. Fortunately, this can be achieved by *superposition coding* [4,8] where the sender communicates messages of the same signal at multiple resolutions by encoding them in different layers. The base layer represents the video frames at a low spatial resolution or sampled sparsely temporally or at a higher compression level while an enhancement layer adds information to the previous layers progressively in spatio-temporal dimensions to get better inference quality. Such coding schemes have been used in video encoding for MPEG [17] to add prediction frames (P/B frames) to the key frames, and in Scalable video coding [27] to scale video quality while broadcasting to multiple devices.

Server-driven protocol as active learning: The workflow of a server-driven protocol closely resembles an active learning process [6,7]. In active learning, given a set of unlabeled data points, one needs to train an accurate classifier by labeling as few data points as possible. Conceptually, the goal of video analytics is to separate the regions of frames that have objects of interest from other regions or other frames; in other words, a "classifier" that separates the spatio-temporal regions that include objects of interest from the rest of the video. Now, we can view a video as a group of bits ("data points"), each being a small area on a single frame, and each frame is pre-clustered into regions ("clusters") (by, for instance, region proposals commonly used in object detectors [23]). The label of each bit is which object of interest it belongs to, or that it belongs to no object. Sending a video segment at a high resolution or a higher sampling rate to the server can be seen as labeling more bits in the region. Casting the server-driven iterative workflow as active learning provides a systematic framework for applying the insight of gradually increasing video quality until the inference is accurate enough.

4.2 A concrete design and its early promise

Next, we present SimpleProto, a simple server-driven protocol that achieves substantial improvement. Extending it with complex active learning strategies is likely to bring more benefits and performance guarantees, and is an interesting next step.

SimpleProto control logic: The client in SimpleProto only passively sends data (a frame or a cropped area) when requested by the server. As in §3.1, the client runs a tracking logic, so the goal of the server is to detect objects and their locations in the frames where the object set changes. For each new frame, SimpleProto runs the basic iterative workflow (§4.1) to make decisions on each of the knobs (§2) one by one, and in the order of cropping, resolution, and frame rate. For brevity, we used a fixed compression level, but it can be optimized in almost the same way as selecting resolution. For each new frame, the logic starts with low sampling rate (or use I or P frames of H.264 to indicate if a frame is worth sampling) and low resolution. In each iteration, if a bounding box is detected with sufficient confidence at a lower resolution, SimpleProto crops out that area, and sends it at a higher resolution to server for inference. The iterations end (1) if the inference confidence is below *alpha* (default, 30%), then the area is discarded, or (2) if the confidence exceeds *beta* (default, 80%), then an object is successfully detected. Finally, if the new sampled frame

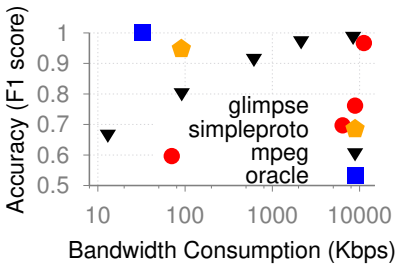

Figure 4: *Comparing SimpleProto with the oracle, and baselines of MPEG and Glimpse on an example video.*

has a different number of objects than the last sampled frame, SimpleProto runs an iterative binary search between these two frames in order to locate the exact frame where new objects first appeared. It should also be noted that SimpleProto will miss an object if it is not "noticeable" at a low resolution (e.g., the object is too small) or at the low frame rate (e.g., the object appears and disappears too quickly).

Under dynamic bandwidth: In practice, the available bandwidth may be dynamic, so SimpleProto's basic logic may not be able to complete all the server-client iterations needed to achieve sufficient accuracy. Fortunately, SimpleProto's iterative workflow is amenable to bandwidth variation. One can set a deadline for per-frame inference, and let SimpleProto exit at the last iteration before the deadline. Because each iteration progressively increases the inference accuracy (e.g., reducing false positives via increasing resolution, and reducing false negatives via sampling more frames), such early exit should strike the best accuracy-bandwidth tradeoff under a given bandwidth constraint.

Protocol interfaces: Like other video streaming protocols, a server-driven protocol for vision analytics operates at application layer. It maintains two connections (which could be multiplexed in one transport session) between the client and the server: one for image/video data transmission, and one for control messages. The server-side process exposes three APIs[2]: (1) a user-facing API, through which a user specifies the objects of interest and gets their locations and labels in the frames; (2) a DNN-facing API, through which the server gets the bounding boxes of detected objects in an image from DNN logic[3]; and (3) a client-facing API, through which the server fetches a region of a specific frame in a specific resolution. The client-side process simply receives frames from the camera (video source), compresses and sends frames as specified by the server in the control messages.

Early promise: Figure 4 compares the bandwidth-accuracy tradeoffs of SimpleProto with MPEG, Glimpse, and the oracle (§3.1) using the same video of Figure 2, and Figure 5 shows the bandwidth reduction of SimpleProto on five traffic video clips from [9] (summarized in Table 1) when the F1 score is fixed above 0.95. On one hand, SimpleProto outperforms both baselines of MPEG and Glimpse: While performance of SimpleProto does depend on the content of the video, SimpleProto achieves comparable accuracy (F1 score $>$ 0.95) using 4-23$\times$less bandwidth than MPEG or Glimpse. The overhead of running SimpleProto is not substantial; SimpleProto runs 2-3 iterations on most selected frames, and the number of frames that the client needs to store locally is bounded by the minimal frame sampling rate of one per few seconds, which means it will only need to save several second of videos. On the other hand, we see a considerable gap between SimpleProto and the oracle protocol, which shows that SimpleProto only scratches the surface and we can improve the trade-off substantially.

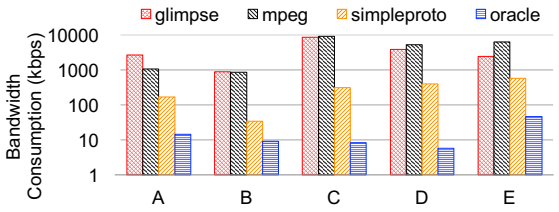

Figure 5: *Bandwidth reduction of SimpleProto over MPEG and Glimpse (F1 score $>$ 0.95) on five videos.*

5 Open questions

Tighter integration with client/server analytics stack: Although superposition coding ensures no redundant data is sent (i.e., total bandwidth usage is bounded), the total delay of these communications and server-side processing could potentially grow unbounded. For instance, the SimpleProto server logic may fetch data from the client multiple times to analyze one frame, and runs complex server logic before the next data fetching. Therefore, to make distributed video analytics practical, the video streaming protocol must be integrated with the server/client-side analytics stack (e.g., [13,35]) to jointly optimize performance tradeoffs.

Leveraging insights from computer vision literature: Like traditional video streaming, streaming video for DNN-based vision analytics faces the challenge that it is extremely complex to understand what the "viewer" (in this case, DNN) cares about. However, we see an opportunity that since vision analytics use artificial models designed by researchers, it is plausible to "open up the black box" of DNN [28]. To take a simple example, suppose we know that the server-side model detects dog by using the color features, then the client can remove all green parts of the video, since no dog is green, e.g. by using probabilistic predicates [16].

[2]Note that a server-driven protocol can be equivalently implemented by the client, as long as the server exposes interfaces to allow client to make the same decisions as if the logic is run on the server.

[3]This is the same to how DNN object detectors are run, so no additional work is needed by vision analytics designers

References

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

[2] V. Chandrasekhar, J. Lin, O. Morre, A. Veillard, and H. Goh. Compact global descriptors for visual search. In *2015 Data Compression Conference*, pages 333–342, April 2015.

[3] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, New York, NY, USA, 2015. ACM.

[4] T. M. Cover. Comments on broadcast channels. *IEEE Transactions on information theory*, 44(6):2524–2530, 1998.

[5] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[6] S. Dasgupta and D. Hsu. Hierarchical sampling for active learning. In *Proceedings of the 25th international conference on Machine learning*, pages 208–215. ACM, 2008.

[7] S. Dasgupta and J. Langford. Active learning tutorial. ICML, 2009.

[8] A. El Gamal and E. C. van der Meulen. A proof of marton's coding theorem for the discrete memoryless broadcast channel. *IEEE Trans. Information Theory*, 27(1):120–122, 1981.

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[10] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 68–81, 2014.

[11] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16. ACM, 2016.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[13] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. Noscope: Optimizing neural network queries over video at scale. *Proc. VLDB Endow.*, 10(11), Aug. 2017.

[14] R. LiKamwa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl. Energy characterization and optimization of image sensing toward continuous mobile vision. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 69–82, 2013.

[15] J. Lin, L. Y. Duan, S. Wang, Y. Bai, Y. Lou, V. Chandrasekhar, T. Huang, A. Kot, and W. Gao. Hnip: Compact deep invariant representations for video matching, localization, and retrieval. *IEEE Transactions on Multimedia*, 19(9):1968–1983, Sept 2017.

[16] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri. Accelerating machine learning queries with probabilistic predicates. In *ACM SIGMOD*. ACM, June 2018.

[17] D. Marpe, T. Wiegand, and G. J. Sullivan. The h. 264/mpeg4 advanced video coding standard and its applications. *IEEE communications magazine*, 44(8):134–143, 2006.

[18] M. Meddeb. *Region-of-interest-based video coding for video conference applications*. PhD thesis, Telecom ParisTech, 2016.

[19] S. Naderiparizi, P. Zhang, M. Philipose, B. Priyantha, J. Liu, and D. Ganesan. Glimpse: A programmable early-discard camera architecture for continuous mobile vision. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 292–305. ACM, 2017.

[20] H. Parmar and M. Thornburgh. Adobes real time messaging protocol. *Copyright Adobe Systems Incorporated*, pages 1–52, 2012.

[21] L. R. Rabiner and R. W. Schafer. Introduction to digital speech processing. *Foundations and Trends® in Signal Processing*, 1(12):1–194, 2007.

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[23] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2017.

[24] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1476–1481, 2017.

[25] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[26] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2):24–31, 2015.

[27] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the h. 264/avc standard. *IEEE Transactions on circuits and systems for video technology*, 17(9):1103–1120, 2007.

[28] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Interpretable deep learning by propagating activation differences. ICML, 2016.

[29] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[30] T. Stockhammer. Dynamic adaptive streaming over http–: standards and design principles. In *ACM conference on Multimedia systems*, pages 133–144. ACM, 2011.

[31] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[33] S. Teerapittayanon, B. McDanel, and H. 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.

[34] J. Wang, B. Amos, A. Das, P. Pillai, N. Sadeh, and M. Satyanarayanan. A scalable and privacy-aware iot service for live video analytics. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, pages 38–49, 2017.

[35] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, volume 9, page 1, 2017.

[36] T. Zhang, A. Chowdhery, V. Bahl, K. Jamieson, and S. 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.

6 Appendix

Table 1 summarizes the basic statistics of the videos from [9]. The object detector used in the experiments is Faster-RCNN ResNet101 [1] with confidence threshold of 30%. Here the definition of object change rate is the average number of how many objects leave and enter a frame within a second.

Video	Frame rate (fps)	Camera type	Resolution (pixels)	Object change rate
A	10	Static	1242 x 375	0.44
B	2	Moving	1280 x 720	0.27
C	10	Moving	1392 x 512	0.3
D	10	Moving	1242 x 375	0.23
E	10	Moving	1392 x 512	0.42
F	10	Moving	1392 x 512	2.73

Table 1: Table summarizing statistics of videos used in Figure 5. Video F is used in Figure 2 and Figure 4.