THE UNIVERSITY OF CHICAGO

ACCMPEG: OPTIMIZING VIDEO ENCODING FOR VIDEO ANALYTICS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCE
IN CANDIDACY FOR THE DEGREE OF
MASTER

DEPARTMENT OF COMPUTER SCIENCE

BY
KUNTAI DU

CHICAGO, ILLINOIS

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This master dissertation is a gift to my grandmother. I do not have the glory to stay with you before your death. You were worrying about me before your death, so hope this dissertation can reach to you and grant you relief. I'm doing well, and I miss you.

ABSTRACT

With more videos being recorded by edge sensors (cameras) and analyzed by computer-vision deep neural nets (DNNs), a new breed of video streaming systems has emerged, with the goal to compress and stream videos to remote servers in real time while preserving enough information to allow highly accurate inference by the server-side DNNs. An ideal design of the video streaming system should simultaneously meet three key requirements: (1) low latency of encoding and streaming, (2) high accuracy of server-side DNNs, and (3) low compute overheads on the camera. Unfortunately, despite many recent efforts, such video streaming system has hitherto been elusive, especially when serving advanced vision tasks such as object detection or semantic segmentation.

This paper presents AccMPEG, a new video encoding and streaming system that meets the three objectives. The key is to learn how much the encoding quality at each (16x16) macroblock can influence the server-side DNN accuracy, which we call *accuracy gradients*. Our insight is that these macroblock-level accuracy gradients can be inferred with sufficient precision by feeding the video frames through a cheap model. AccMPEG provides a suite of techniques that, given a new server-side DNN, can quickly create a cheap model to infer the accuracy gradients on any new frame in near realtime. Our extensive evaluation of AccMPEG on two types of edge devices (one Intel Xeon Silver 4100 CPU or NVIDIA Jetson Nano) and three vision tasks (six recent pre-trained DNNs) shows that compared to the state-of-the-art baselines, AccMPEG (with the same camera-side compute resources) can reduce the end-to-end inference delay by 10-43% without hurting accuracy.

CHAPTER 1

INTRODUCTION

Empowered by modern computer vision, video analytics applications running on edge/mobile devices are poised to transform businesses (retail, industrial logistics, home assistance, etc), and public policies (traffic management, urban planning, etc) TrafficVision (2021); TrafficTechnologyToday (2019); GoodVision (2021); intuVision (2021); VisionZero; Microsoft (2019). These emerging video applications use deep neural networks (DNNs) to analyze massive videos from edge video sensors, creating an explosive growth of video data SecurityInfoWatch (2012, 2016) that serve *analytical* purposes rather than being watched by human users for entertainment SecurityInfoWatch (2016); GrandViewResearch (2018); Slate (2019).

A key component of these video-analytics applications is an efficient *video compression*[1] algorithm, which compresses videos in realtime while preserving enough information for accurate inference by the *final DNN* running on a remote (cloud) server StreamingMedia (2019). An ideal video compression algorithm should meet three requirements that are key to edge video analytics applications: (1) high inference accuracy by the final DNN, (2) low end-to-end delay of encoding and streaming the video to the analytical server, and (3) low compute overhead on the camera side. There have recently been many proposals of such analytics-oriented video compression algorithms; for instance, they leverage the *spatial heterogeneity* of the final DNNs Du et al. (2020); Liu et al. (2019), such as object detection and segmentation: only in a small fraction of regions (*e.g.,* which contain important details), lowering quality tends to lower inference accuracy and renders it useless.

Unfortunately, none of the existing solutions can simultaneously meet the three requirements, especially when serving advanced tasks such as object detection or semantic segmentation. For instance, using camera-side heuristics to filter out (or lower encoding of) unuseful pixels quality is sensible Zhang et al. (2015); Li et al. (2020); Canel et al. (2019),

1. We use video compression and encoding interchangeably.

but existing designs cannot precisely lower quality of unuseful pixels without affecting useful ones, unless the heuristics themselves are almost as compute-intensive as the final DNNs. In response, some proposals achieve high accuracy and low camera-side overhead by sending the content to the server-side DNN first to extract feedback Du et al. (2020); Liu et al. (2019); Zhang et al. (2021) from the server, based on which the camera can then encode the video near-optimally or run local inference (*e.g.,* object tracking), causing high end-to-end delay. Other solutions extract and encode feature maps on the camera Duan et al. (2020); Xia et al. (2020); Emmons et al. (2019); Kang et al. (2017b); Matsubara et al. (2019), and they work well for classification models but not for advanced DNNs (object detection and segmentation models), whose feature maps are orders of magnitude larger than those of classification DNNs.

This paper presents a new video streaming system called AccMPEG that meets the three aforementioned requirements. At a high level, AccMPEG runs a cheap *quality selector* logic (a shallow neural net, MobileNet-SSD Howard et al. (2017); Sandler et al. (2018)) that determines a near-optimal encoding scheme for any frame—the encoding quality at each (16x16) macroblock, and encodes the frames using popular video codecs like h.26x with region-of-interest (RoI) encoding. The insight underpinning the cheap quality selector is that inferring the influence of the encoding quality at each macroblock on the final DNN accuracy, which we call *accuracy gradients* or *AccGrads*, is much simpler than semantic segmentation (a common vision task) for multiple reasons (§3.2): for instance, assigning high or low quality to macroblocks of a 720p frame (1280x720) is equivalent to assigning binary labels on an 80x45 image (a 720p frame has 80x45 macroblocks), which can be much simpler than most modern vision tasks.

AccMPEG's quality selection also strikes a favorable accuracy-delay balance. Prior techniques assign encoding quality at coarser granularities, such as encoding entire bounding boxes in high quality and entire background in low quality. In contrast, AccMPEG's macroblock-level quality selection could outperform them by encoding some background

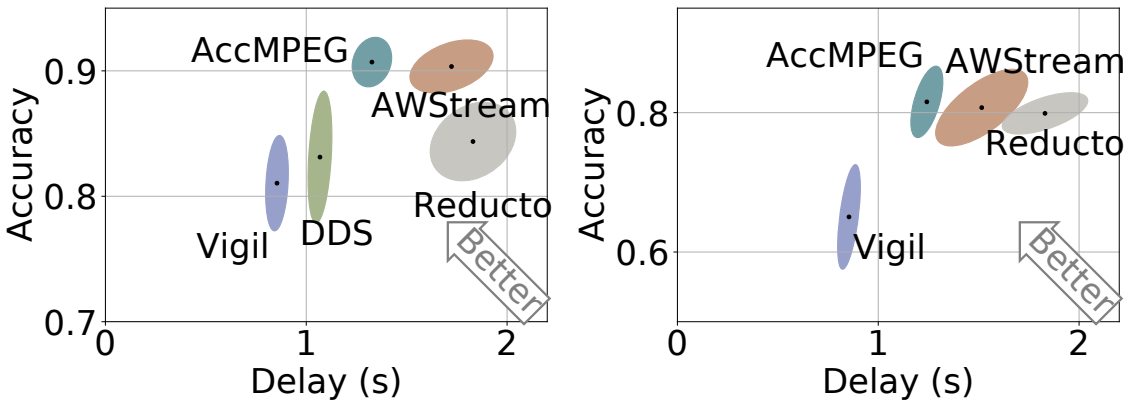


Figure 1.1: Example results of the accuracy-delay tradeoffs of AccMPEG and baselines. AccMPEG achieves 10-30% smaller end-to-end delay without sacrificing accuracy, or 1-5% higher inference accuracy than state-of-the-art solutions.

macroblocks in high quality (*e.g.,* to provide necessary context that improves accuracy) and some inside the bounding boxes in low quality (*e.g.,* if encoding other macroblocks of a large object in high quality is sufficient to achieve high accuracy).

Finally, one must be able to quickly customize the quality selector based on the need of a new final DNN. Traditionally, training such quality selector would run numerous forward and backward propagations on an entire pipeline (quality selection, encoding, and inference) over many training images. In contrast, AccMPEG *decouples* the final DNN from the training of the quality selector (§5). To this end, we directly derive the true *AccGrad* per macroblock by treating the final DNN as a differentiable blackbox, and then train the quality selector as a standalone (low-dimensional) segmentation model to infer these *AccGrads*, which also reduces the number of training iterations and training images.

Using videos of three different genres, three typical vision tasks (object detection, semantic segmentation, and keypoint detection), and five recent off-the-shelf vision DNNs, we show that on two types of edge devices (one CPU or a Jetson Nano) AccMPEG can reduce the inference delay by 10-43% without hurting accuracy compared to various state-of-the-art baselines. Figure 1.1 shows an example improvement of AccMPEG's accuracy-delay tradeoffs over the baselines (See Figure 6.1 for the complete results). AccMPEG's encoding speed (30fps with one Intel Xeon Silver 4100 CPU) is only marginally slower than basic video

encoding and is much faster than the baselines that achieve similar compression efficiency. Moreover, an AccMPEG encoder for a new DNN can be created within only 8 minutes.

Admittedly, not all techniques of AccMPEG are exactly new: it uses standard video codec libraries Wiegand et al. (2003); Coding & Rec (2013) that support RoI encoding, and many proposals in this space (*e.g.,* Wang et al. (2017); Mnih et al. (2014)) use the the spatially uneven distribution of DNN attention. Nonetheless, AccMPEG strikes a unique balance among encoding/streaming delay, inference accuracy, and low compute overhead in advanced vision tasks (*e.g.,* object detection, segmentation). Our contribution is two-fold:

- A cheap quality selector that infers accuracy gradients (how sensitive a DNN's output is to the encoding quality at each macroblock) on each frame and selects encoding quality per macroblock, with only a compute overhead on par with video encoding.
- Fast training (within several minutes) of the quality selector for any given final DNN, which degrades gracefully when the final DNN changes.

CHAPTER 2

MOTIVATION

We begin by motivating the three performance requirements that drive our design (low accuracy, low encoding and streaming delay, and low camera-side compute overhead). We then elaborate on why prior solutions struggle to simultaneously meet the three requirements.

2.1 Video encoding for edge video analytics

Distributed video analytics: As accurate analytics requires compute-intensive DNNs that cheap video sensors cannot afford, the video frames are often *compressed* by a *video encoder* and then sent to a remote server for accurate DNN-based analytics (Figure 2.1). We refer to the server-side DNN as the *final DNN*. In this work, we focus on three video analytics tasks: object detection (one labeled bounding box for each object), semantic segmentation (one label for each pixel), and keypoint detection (17 keypoints such as hand and elbow on a human body).

There are two types of video analytics. In live analytics, the video frames are continuously encoded and sent to a remote server which runs the final DNN to analyze the video in an online fashion Du et al. (2020); Liu et al. (2019); Zhang et al. (2015); Chen et al. (2015); Li et al. (2020); Zhang et al. (2021, 2018); Kang et al. (2017a); Zhang et al. (2017). In retrospective analytics, the encoded video is first stored locally on the camera, and an operator can choose to fetch part of the video for DNN-based analytics Keahey et al. (2019); Kang et al. (2018); Hsieh et al. (2018).

Performance requirements: In both live and retrospective analytics, a key component is the video encoding algorithm. An ideal video encoding algorithm for distributed video analytics should meet three goals:

- *High accuracy:* The encoded video must preserve enough information for the final DNN to

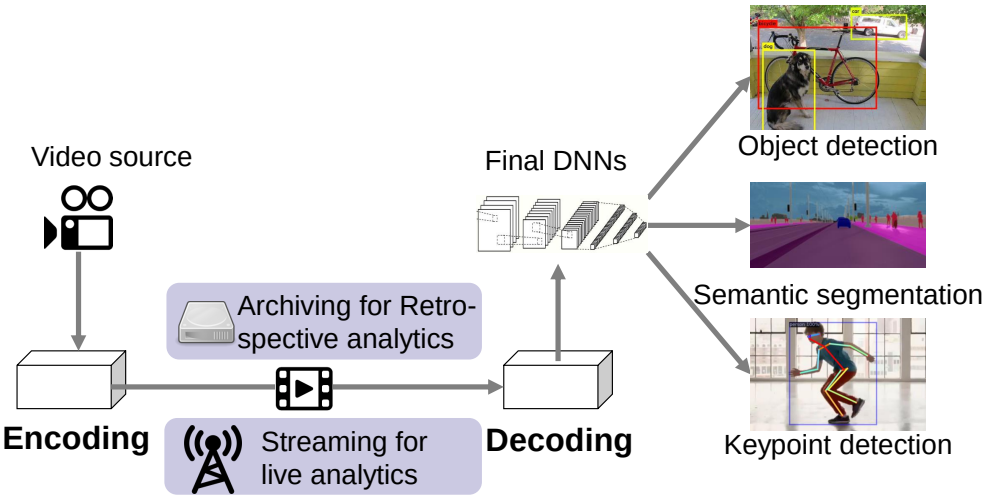


Figure 2.1: Illustration of video encoding as part of the video analytics pipelines of three tasks

return nearly identical inference results as if it runs on the original video frames.[1]

- *Low delay:* The delay of encoding the video (encoding delay) and streaming the video to the server (streaming delay) should be low.
- *Low camera cost:* The encoding algorithm should be cheap enough to run at 30fps with only marginal extra compute overhead compared to encoding videos using popular codecs such as h.26x.

2.2 Limitations of previous work

Here, we categorize previous work in four general approaches and explain why they cannot meet all performance requirements simultaneously.

Local frame-filtering schemes: One of the popular techniques is to let the camera run a simple logic to identify which frames are irrelevant to the vision task and thus can be discarded Li et al. (2020); Canel et al. (2019); Chen et al. (2015) or encoded in low

1. To calculate the accuracy of an inference result on a compressed frame, we obtain the "ground truth" results by running the final DNN on the high-quality video frames (rather than using the human-annotated labels). Thus, any inaccuracy will be due to the video stack (*e.g.,* video compression, DNN distillation), rather than errors made by the final DNN itself. This is consistent with recent work (*e.g.,* Zhang et al. (2018, 2015, 2017); Keahey et al. (2019); Mullapudi et al. (2019); Kang et al. (2017a)).

quality Zhang et al. (2018). This approach works well when the video content is relatively stationary, where the incidents/objects of interest are rare and easy to detect; *e.g.,* in wildlife camera feeds, animals are rare and readily detectable since they are the only moving objects on a static background. However, for frames that are not discarded, this approach encodes the entire frames with uniform quality, which can be suboptimal, since the objects of interest often occupy only a small fraction of each frame Du et al. (2020); Liu et al. (2019), leading to higher streaming delays than necessary.[2]

Local heuristics to lower background quality: Since objects of interest often account for a small fractions of each frame, some work (*e.g.,* Zhang et al. (2015); Dai et al. (2021)) uses local heuristics to filter out (or lower the quality of) the background pixels and sends the remaining object-related pixels in high quality to the server-side final DNN. However, these local heuristics are constrained by the limited camera-side compute resources, giving rise to false negatives—object-related pixels are treated as background and thus filtered out or sent in low quality, causing the DNN to miss objects of interest. For instance, to detect potential object-related regions, Vigil Zhang et al. (2015) relies on a low-accuracy non-convolutional Haar-cascade-classifier-based object detector, and CiNet Dai et al. (2021) uses a very shallow convolutional network (with only 2 convolutional layers, 1 average pooling layer and 2 fully-connect layers) designed to handle only few objects per frame. There are also deeper NNs such as MobileNet-SSD Howard et al. (2017) that run on resource-constrained cameras, but they have to downsize frames to low resolutions (*e.g.,* 300×300) for real-time inference, thus prone to missing small objects.

Server-driven compression: To overcome the camera-side resource constraints, another approach Du et al. (2020); Liu et al. (2019); Zhang et al. (2021) leverages the abundant server-side compute resources to generate feedback on how videos should be encoded. This approach generally compresses videos efficiently while achieving high accuracy, but it suffers

2. Though CloudSeg Wang et al. (2019b) does not perform frame filtering at the camera side, it also share the limitation since it compresses entire frames in same encoding quality.

from a high inference delay. DDS Du et al. (2020), for instance, sends a low-quality video to the server-side DNN which returns to the camera which regions must be encoded in high quality, but under high network latency, getting such server-driven feedback can take at least two network round-trip times before the camera can actually encode the video for final DNN inference, causing high inference delay on each frame.

Local DNN compression: Instead of encoding videos on the camera, some proposals also extract the final DNN's feature maps on the camera and compress the feature maps which might contain less information than the original raw frames Duan et al. (2020); Xia et al. (2020); Emmons et al. (2019); Kang et al. (2017b); Matsubara et al. (2019). While this approach has shown promise with classification or action recognition DNNs Kang et al. (2017b); Emmons et al. (2019); Duan et al. (2020), these tasks do not require the spatial locations of objects, allowing aggressive aggregation of feature maps over an entire frame. In contrast, the vision tasks that we focus on (*e.g.,* object detection, semantic segmentation) are sensitive to object locations, making the intermediate feature maps much larger and much more difficult to be compressed efficiently. For example, many state-of-the-art object detectors (*e.g.,* Detectron2) use expensive feature extractors such as ResNet101 He et al. (2016), and if we feed a 720p (1280×720) frame through even parts (*e.g.,* 90) of its convolution layers, the feature map still contains 2×10^7 floating-point numbers per frame, 20x more than the number of pixels in the original frame.

There are also proposals to train DNN autoencoders that compress video to a smaller size than the popular video codecs do, but these DNNs are much more compute-intensive than the video codecs and even the final DNN. For example, NLAM Liu et al. (2020) requires performing expensive 3D convolutions on videos for more than 30 times, while an object detector backbone (*e.g.,* ResNet34 He et al. (2016)) only performs 2D convolution for 34 times.

CHAPTER 3

OVERVIEW OF ACCMPEG

In this section, we present AccMPEG, a new video encoding algorithm that uses a *cheap* camera-side model to decide which regions should be encoded in higher quality. Here, we introduce AccMPEG's workflow and its challenges, and then present the key idea that addresses these challenges.

3.1 Workflow and challenges of AccMPEG

Figure 3.1 depicts the workflow of AccMPEG. When a video frame arrives, AccMPEG first feeds it through a cheap quality selector model, called *AccModel*, to obtain a *macroblock-level quality selection*—which macroblocks (16x16 blocks, which many modern video codecs Wiegand et al. (2003) use as the basic encoding unit) should be encoded in high quality and which should be in low quality. The camera then encodes the video frames according to the quality selection and sends the encoded video to the server for the final DNN inference. The quality selector (*AccModel*) is trained for each final DNN offline, such that when the video frames are encoded in its selected quality, the DNN can return accurate inference results. As we will see in §5, the quality selector can also be re-used among DNNs of similar tasks with only marginal performance penalty.

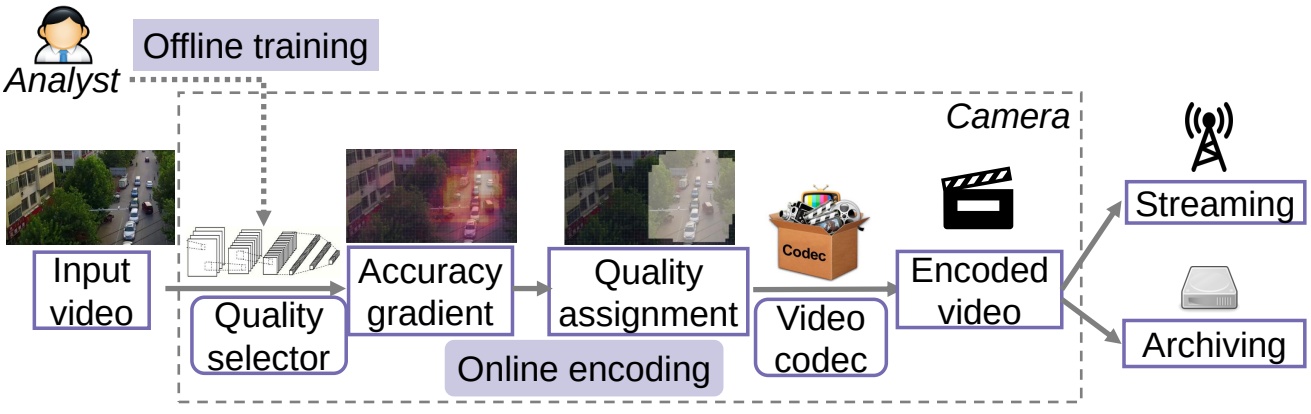


Figure 3.1: Overview of AccMPEG.

Challenges: The key component of AccMPEG is the quality selector (*AccModel*) that selects the encoding quality per macroblock. It has three challenges: *(i)* How to optimally assign encoding quality at the fine spatial granularity of macroblocks to achieve better accuracy-delay tradeoffs than baselines? *(ii)* How to minimize the per-frame compute overhead of *AccModel* to allow real-time video encoding on the camera side? And *(iii)* how to quickly train *AccModel* for each server-side DNN?

3.2 Key idea: Accuracy Gradient

The key idea to address these challenges of *AccModel* is to obtain the *Accuracy Gradient* (hereinafter *AccGrad*) of each macroblock, which measures how much the encoding quality at the macroblock can affect the DNN inference accuracy. Mathematically, the *AccGrad* of macroblock B in a given frame is defined as:

$$AccGrad_B = \sum_{i \in B} \left\| \frac{\partial Acc\big(D(\mathbf{X}); D(\mathbf{H})\big)}{\partial \mathbf{X}_i}\bigg|_{\mathbf{X}=\mathbf{L}} \right\|_1 \cdot \|\mathbf{H}_i - \mathbf{L}_i\|_1, \tag{3.1}$$

where i, $\mathbf{L}$, and $\mathbf{H}$ denote a pixel within B, the low-quality encoded frame, and the high-quality encoded frame, respectively. D denotes the server-side DNN inference function, $Acc\,(D(\mathbf{X}); D(\mathbf{H}))$ is the accuracy of inference result on $\mathbf{X}$ (*i.e.,* its similarity with the inference result on the high-quality frame $D(\mathbf{H})$), and $\|\cdot\|_1$ is the L1-norm.

Intuitively, the encoding quality of the macroblocks with higher *AccGrad* values have a greater influence on the DNN accuracy, so these macroblocks should be encoded in high quality. We will present the *AccGrad*-based quality assignment logic in §4. Appendix A.1 gives the mathematical reasoning behind Equation 3.1.

AccGrad enables a series of system optimizations that help address the challenges in §3.1.

Better accuracy-delay tradeoffs via finer-grained quality assignments: We begin with the benefit of *AccGrad*-based quality selection over the traditional region-based quality selection Du et al. (2020); Liu et al. (2019); Zhang et al. (2021), which identifies regions with

greater impact on DNN inference (*e.g.,* via region proposals Ren et al. (2015)) and then use high quality to encode some region proposals in their *entirety* and low quality to encode the *whole* background. These coarse-grained encoding schemes can be inefficient. On one hand, some surrounding pixels of the object bounding boxes can still be crucial for the final DNN to accurately detect/classify objects and demarcate their boundaries from the background. For instance, to detect the car in Figure 3.2(a), one must encode not only its bounding box in high quality but also some neighboring macroblocks too. On the other hand, some pixels inside an object's bounding box (*e.g.,* the smooth surface of a car in Figure 3.2(b)) have similar RGB values regardless of the encoding quality, so it is safe to compress them in low quality without hurting inference accuracy.

In contrast, *AccGrad* by definition can capture such fine distinction among macroblocks. For instance, the macroblocks surrounding the car's bounding box (Figure 3.2(a)) will have high *AccGrad* values in Equation 3.1, because $Acc\big(D(\mathbf{L}); D(\mathbf{H})\big)$ will have a high derivative with respect to the pixels in these macroblocks. Similarly, the smooth surface of the car in Figure 3.2(b) is likely to have low *AccGrad* (despite being part of the car object), because $\|\mathbf{H}_i - \mathbf{L}_i\|_1$ will be small on the pixels i in these macroblocks.[1]

AccModel **might not be compute-intensive:** *AccModel* can be seen as a segmentation problem, but unlike normal segmentation models that are compute-intensive, a cheaper model might suffice for *AccModel* for three reasons.

First, unlike traditional image segmentation that gives one label per pixel, *AccModel* returns one label per *16x16 macroblock* (*i.e.,* all pixels in a macroblock share the same DNN output). Therefore, unlike traditional convolutional operations which scan the image pixel by pixel, *AccModel* only needs to scan the image *macroblock by macroblock* (saving upto $16^2 = 256$x on convolutional operations).

1. *AccGrad* may look similar to the saliency maps in computer vision, but there is a key distinction. While saliency captures which pixel *values* have more influence on the DNN output, *AccGrad* captures how much changing a macroblock's *encoding quality* changes the DNN inference accuracy.

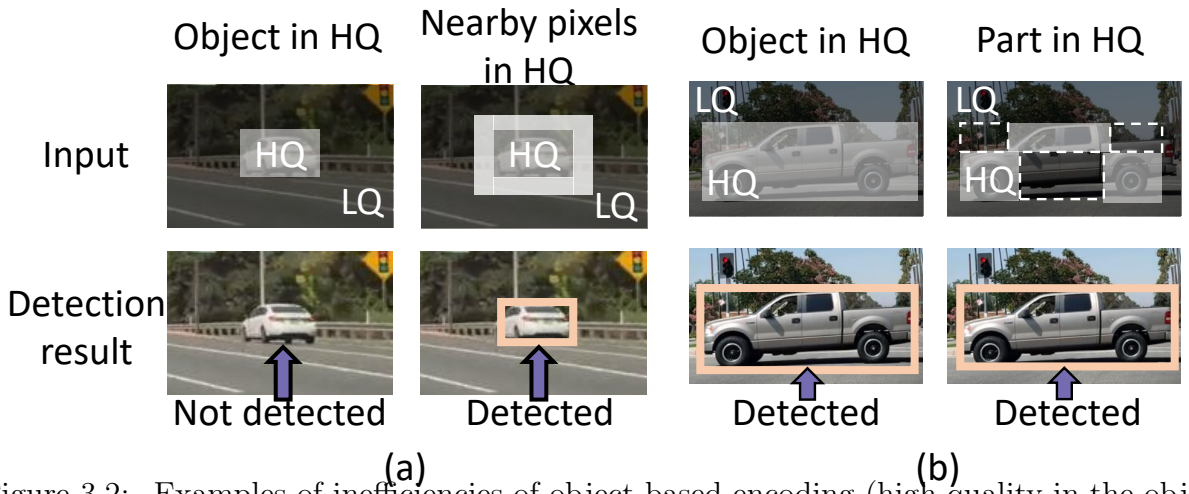


Figure 3.2: Examples of inefficiencies of object-based encoding (high quality in the object bounding box): (a) Objects not detected unless nearby pixels are in high quality; and (b) Objects still detected even if just parts are in high quality.

Second, *AccModel* only needs to do a *binary* classification on each macroblock (either high quality or low quality), rather than multi-class segmentation, which further reduces the complexity of *AccModel*.

Third, while accurate segmentation must minimize both false positives (*e.g.,* pixels misclassified as objects) and false negatives (*e.g.,* pixels misclassified as background), AccMPEG has more tolerance towards false positives (*e.g.,* macroblocks mislabeled with high *AccGrad*). Encoding a few more macroblocks in high quality has marginal impact on the delay-accuracy tradeoff, because the intra-frame encoding commonly used in video codecs makes the compressed video size to grow only sublinearly with more high-quality regions. Appendix A.4 provides the empirical evidence.

Fast training of *AccModel*: Training *AccModel* naively can be prohibitively expensive, because it requires running numerous forward/backward propagations on the final DNN to calculate losses and obtain gradients. Fortunately, *AccGrad* can be directly derived from the final DNN (Equation 3.1) using only two forward propagations and one backward propagation. As we will see in §5, this allows us to separate the compute-intensive final DNN from *AccModel* training. This can save 10× overhead, as AccMPEG trains *AccGrad* for 15 epochs (see §5 for details), each requiring only three propagations through the final DNN.

CHAPTER 4

ONLINE ENCODING

We now describe AccMPEG's online encoding process, including the architecture of *AccModel* and how AccMPEG assigns encoding quality to each macroblock.

Architecture of *AccModel*: We leverage the pretrained MobileNet-SSD feature extractor Howard et al. (2017), a widely-used feature extractor for cheap edge devices, as the feature extractor of *AccModel*. We resize these features so that one macroblock corresponds to one feature vector, and append three convolution layers to classify which macroblock should be in high quality.

Compute cost of *AccModel*: Our model is much more compact than other commonly-used feature extractors. To put it into perspective, our *AccModel* uses 12 GFLOPs, about 3× less than a typical cheap convolutional model such as ResNet18 which uses 33 GFLOPs. In addition, since the architecture consists only of convolutional layers (except for batch normalization and activation), its computational overhead is proportional to the size of the input frame (*e.g.,* 4× faster when the frame size halved in both dimensions).

***AccGrad*-based quality assignment:** Given macroblock-level *AccGrad* of a frame, AccMPEG then uses a threshold α to determine which macroblocks should be in high quality—all blocks B with $AccGrad_B \geq \alpha$ will be encoded in high quality. After a set of blocks are selected, AccMPEG then expands these selected blocks to each direction by γ (by default 5) blocks (if they are not already selected). Intuitively, a lower α increases the accuracy at the expense of encoding more macroblocks with high *AccGrad*.

Frame sampling for cheap inference: We further reduce *AccModel*'s compute overhead by running it once every k frames and using its output to encode the next k frames (by default, $k = 10$). Empirically, it significantly reduces the camera-side overhead (Figure 6.3) without much impact on accuracy. The intuition is that although the *AccGrad* of a macroblock fluctuates over time, its value will not shift dramatically across consecutive frames to change its

quality selection. We have also empirically examined this intuition on the quality assignment generated from the dashcam dataset (see §6 for the detail of our dataset) by $AccModel$ pretrained from FasterRCNNRen et al. (2015): the encoding quality selection of at least 84% of macroblocks remains unchanged between one frame and the ninth frame after it.

CHAPTER 5

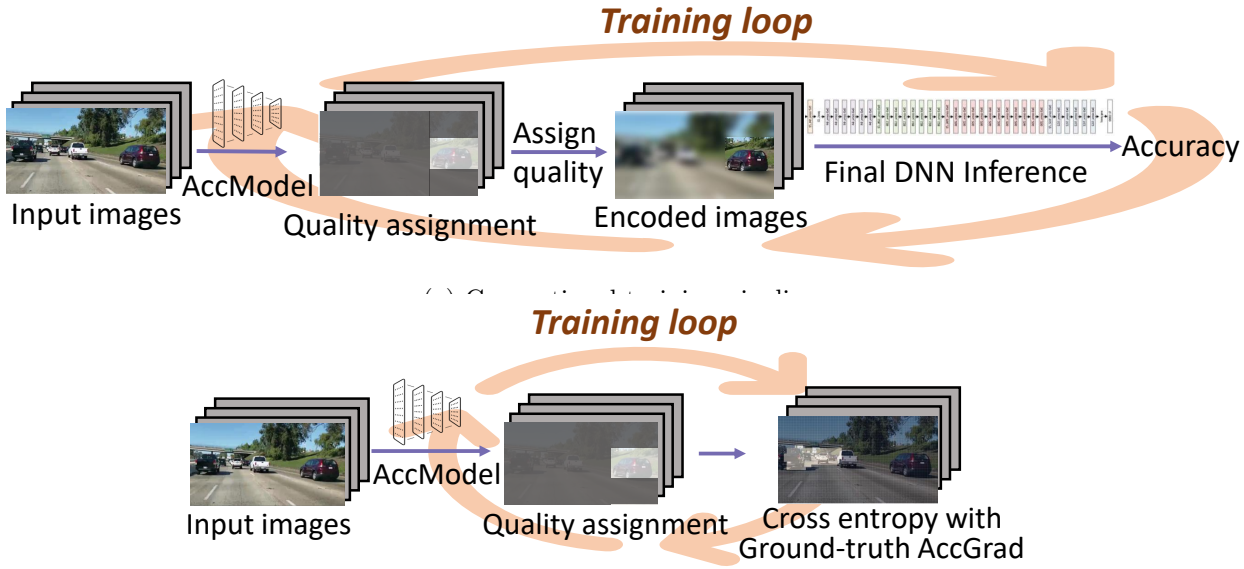
OFFLINE TRAINING

AccModel customizes the video encoding for near-optimal accuracy-delay tradeoffs of a given final DNN model. So a natural question is how to quickly create the *AccModel* for a new DNN. Here, we describe how we speed up the training of *AccModel* by separating the final DNN from the training process, and then explain why reusing *AccModel* may also lead to decent performance.

Conventional training process: Before describing how AccMPEG trains *AccModel*, we first explain the straightforward approach to training *AccModel* (depicted in Figure 5.1a) which sets up the entire pipeline of encoding and inference and minimizes the end-to-end loss (we will define the training loss soon). For each input image, it first feeds the high-quality version $\mathbf{H}$ through the *AccModel* to get the *AccGrad* matrix $\mathbf{M} = AccModel(\mathbf{H})$, creates the encoded image $\mathbf{X} = \mathbf{M} \times \mathbf{H} + (\mathbf{1} - \mathbf{M}) \times \mathbf{L}$ by linearly combining the low-quality version $\mathbf{L}$ and $\mathbf{H}$ using $\mathbf{M}$[1], feeds it through the final DNN to get result $D(\mathbf{X})$, and finally calculates the accuracy (loss) of $D(\mathbf{X})$ with $D(\mathbf{H})$ as the ground truth using *Acc*. This training process is actually widely used in computer vision (*e.g.,* Goodfellow et al. (2014); Johnson et al. (2016); Ledig et al. (2017)) and video analytic systems Wang et al. (2019b). However, it is prohibitively expensive: each forward or backward propagation of the pipeline must run the expensive D, in addition to *AccModel* and *Acc* each once. Thus, the compute overhead of *AccModel* training is dominated by running the forward/backward propagations on D (Caching *AccModel* results does not help, since the *AccModel* output changes after each update).

Separating final DNNs from training via *AccGrads*: In contrast, AccMPEG calculates the *AccGrads* on each training image first (using Equation 3.1), which requires only one

1. We make the elements in $\mathbf{M}$ to be between 0 and 1 by applying a softmax filter on the output of *AccModel*.



(b) Decoupling the final DNN from training using *AccGrad*

Figure 5.1: Contrasting the conventional approach to *AccModel* training with AccMPEG. We separates the final DNN from training by first generating the ground-truth *AccGrad* from the final DNN, and then training *AccModel* to minimize the loss with the ground-truth *AccGrad*, which significantly speeds up the training.

forward and backward propagation of D on the high-quality version of the image. Once the *AccGrads* of each training image is generated, the training can be reformulated as $\min_{AccModel} CrossEntropy\big(AccModel(\mathbf{H}), \mathbf{M}^*\big)$, where $\mathbf{M}^*$ is the "ground truth" *AccGrad* matrix of size $w \cdot h$ generated by Equation 3.1 (not to be confused with the ground-truth inference output of a final DNN) and the cross-entropy loss (with 4x weight on those blocks that should be in high quality) commonly used in deep learning to measure the discrepancies between two vectors. Training *AccModel* thus requires only one forward and one backward propagation on *AccModel*. Thus, by generating the ground truth first and then training *AccModel*, we can train the *AccModel* within 8 minutes using 8 GPUs (§6.4).

Using pre-trained models: Instead of training *AccModel* from scratch, we initialize *AccModel* with a pretrained MobileNet-SSD backbone and then fine-tune the model. It has the similar benefit of model fine-tuning widely used in industry: the training can converge with fewer training epochs on fewer training images Gao et al. (2021). Specifically, we train *AccModel* on a 10× randomly downsampled training set of the final DNN model (*e.g.,* COCO

dataset) for 15 training epochs and pick the model with lowest loss on cross validation set as our final *AccModel*. The total training time of *AccGrad* is about 8 minutes (§6.4).

Reusing *AccModel*: Ideally, any new server-side final DNN requires a (slightly) different *AccModel*. However, when the new final DNN is trained on the same dataset (same images and same labels) as another final DNN (whose *AccModel* is already trained), it is possible to reuse the *AccModel*. This is because the macroblocks with high *AccGrads* are typically those related to small, partially occluded, or darkly lit objects in the dataset. Thus, training the new *AccModel* based on the *AccGrads* of the old final DNN on the same dataset would likely yield a similar *AccModel*. Since DNN models are sometimes trained on popular datasets (such as the COCO dataset COCO (2017)), *AccModel* can sometimes be re-used among different final DNNs (we will empirically evaluate it in Figure 6.5).

CHAPTER 6

EVALUATION

Finally, our evaluation of AccMPEG shows that:

- AccMPEG achieves better accuracy-delay tradeoffs: 10-43% lower delay while maintaining comparable accuracy as the baselines. The improvement remains similar on three vision tasks and five final DNN models with a variety of architectures and backbones (§6.2).
- AccMPEG has the lowest camera-side overhead compared to all the baselines that deploy customize logic at the camera side and achieve comparable accuracy, and the extra compute overhead due to *AccModel* is less than the popular video codecs (§6.3).
- Given a final DNN, an *AccModel* can be created within 8 minutes using 8 GPUs. Even if a final DNN changes without updating the *AccModel*, AccMPEG still achieves better accuracy-delay tradeoffs if the new vision model is trained on the same dataset as the previous one (§6.4).

6.1 Setup

Dataset: Table 6.1 summarizes the 3 video datasets we used to evaluate AccMPEG: 5 driving videos and 7 dashcam videos for object detection, and 6 surfing videos for keypoint detection and semantic segmentation. All videos are obtained by searching on YouTube. We search keywords (such as "highway dashcam hd") in incognito mode to avoid customization bias. All videos and collection details are available in this anonymous link AccMPEG.

Device setting: We create a 30fps video source where different methods can read raw

Name	Vision task	# Videos	# Frames
Driving	object detection	5	9000
Dashcam	object detection	7	12600
Surf	semantic segmentation keypoint detection	6	6598

Table 6.1: Summary of our datasets.

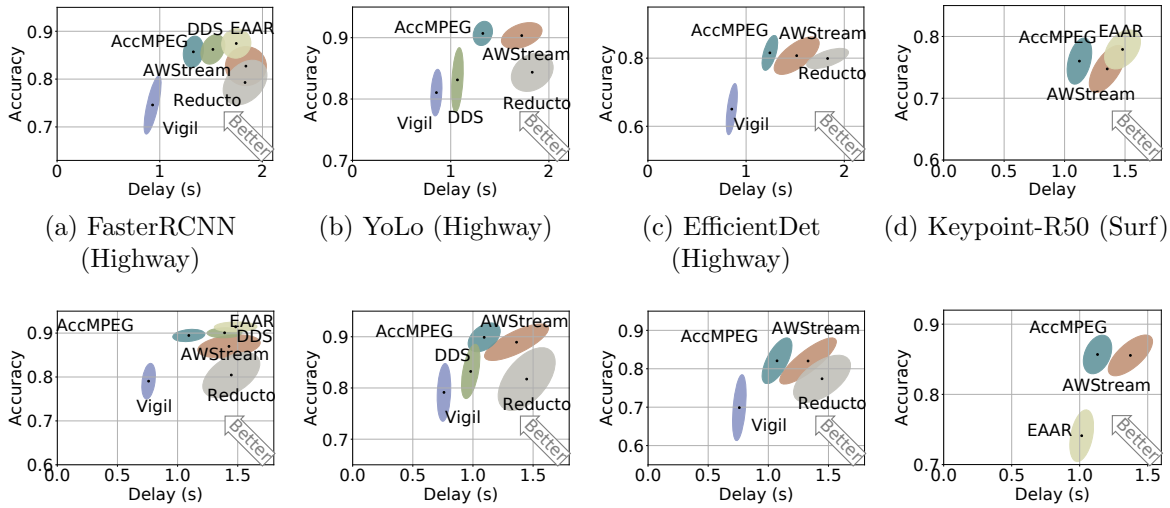
(1280×720) frames one by one. To achieve real-time video streaming, we let the camera stream out the video in the form of short video chunks (this aligns with previous work Du et al. (2020); Zhang et al. (2018)), each consisting of 10 frames. To fairly compare the encoding delay of different methods, we benchmark the encoding delay on one Intel Xeon Silver 4100 CPU and run the encoding of AccMPEG and baselines everytime the camera reads 10 frames for its current video chunk (we also benchmark the performance of AccMPEG on baselines on Jetson Nano, a cheap GPU device (with one 128-core Maxwell GPU, one Quad-core ARM A57 CPU and 4GB memory ChameleonHardware (2021)))[1] provided in the Chameleon testbed Keahey et al. (2020)). We use openVINO to accelerate[2] all camera-side DNNs on CPUs. We also make minor modification to the H.264 codec to enable macroblock-level region-of-interest encoding.

Server: We train *AccModel* offline on the server with 8 GeForce RTX 2080 SUPER GPU. In the online encoding phase, we run the decoding on Intel Xeon Silver 4100 CPU and run the inference on GeForce RTX 2080 SUPER GPU.

Video analytics tasks and DNNs: We test AccMPEG on three tasks: object detection, semantic segmentation and keypoint detection. Here we list the DNNs we use for these tasks (We use *italic* to show the DNN that we use to deliver *AccModel* for that vision task. All DNNs are pretrained from COCO dataset COCO (2017)). We pick three *object detection* models that represent three types of different architectures: *FasterRCNN* Ren et al. (2015) (a two-stage detector with features from different resolutions Lin et al. (2017a)), *YoLov5* Redmon & Farhadi (2017) (a single-stage detector), and *EfficientDet* Tan et al. (2020) (a detector with machine-optimized architecture Zoph & Le (2016)). We *FCN-ResNet50* PyTorch for *semantic segmentation*. We also pick two *keypoint detection* models: *Keypoint-ResNext101* He et al. (2017); Xie et al. (2017) and Keypoint-ResNet50 He et al. (2017),

1. A Jetson nano developer board is only 60$ Amazon.

2. This acceleration will not reduce floating point precision, and thus will not alter the inference result of AccMPEG.



(a) FasterRCNN (Highway) (b) YoLo (Highway) (c) EfficientDet (Highway) (d) Keypoint-R50 (Surf)

(e) FasterRCNN (Driving) (f) YoLo (Driving) (g) EfficientDet (Driving) (h) Keypoint-X101 (Surf)

Figure 6.1: The delay vs. inference accuracy of AccMPEG and several baselines on various video datasets (in parentheses) and different DNN models (the three object detection models use different backbones). AccMPEG achieves high accuracy with 10-43% delay reduction on object detection and 17% on keypoint detection. Ellipses show the 1-σ range of results.

Setting of AccMPEG: For the encoding quality, we use (30, 40) as the QP value for high quality and low quality for object detection and (30, 51) for keypoint detection. By default, we use $\alpha = 0.2$ as the *AccGrad* threshold.

Baselines: We use baselines from five categories:

- *Uniform quality:* AWStream Zhang et al. (2018) tunes the encoding parameters of the underlying codec (resolution, QP, and frame rate), though unlike AccMPEG, they use the same configuration for all frames in each time window (on the timescale of minutes)[3]. (VStore Xu et al. (2019) shares a similar idea.) To show their limitation, we use an "idealized" version where the parameters are set such that the size reduction is maximized while its accuracy is almost same to AccMPEG.
- *Server-driven approach:* DDS Du et al. (2020) and EAAR Liu et al. (2019) belong to this type and they share the idea of encoding different regions with different quality levels. By

3. We assume that AWStream can obtain the accuracy-delay profile without extra cost, which makes AWStream strictly better.

default, we use QP = (40, 30) as the low quality and high quality settings.[4]

- *Frame filtering:* We choose Reducto Li et al. (2020), one of the most recent proposals along this line. We use the implementation from Github (b).
- *Autoencoder*: We pick a pre-trained autoencoder Github (a) (introduced in Theis et al. (2017)).

We do not include CloudSeg Wang et al. (2019b) in our evaluation, because it augments the server-side DNN by a super-resolution model, which is complementary to the camera-side video encoding schemes above.

Metrics: Following the definitions in §2.2, we compare different techniques along three key metrics: *delay*, *inference accuracy*, and *camera-side compute cost* (the cost is measured by camera-side encoding delay and overheads). In particular, we use F1 score as the accuracy metric in object detection, IoU in semantic segmentation, and distance-based accuracy in keypoint detection. These metrics all values in [0,1], with higher values the better. We calculate the camera-side delay on one Intel Xeon Silver 4110 CPU. We assume there are 5 video streams sharing a network link with 2.5mbps bandwidth upload speed (the average upload speed of Sprint LTE connection OpenSignal (2018)) and 100ms latency Wang et al. (2019a). We do not include the server-side inference delay, since AccMPEG does not put extra compute cost on the server side, and the optimization of server-side delay is not our contribution either.

6.2 Better accuracy-delay tradeoffs

Figure 6.1 compares AccMPEG's performance distributions with those of the baselines on the three tasks (on their perspective default full DNNs) and various datasets[5]. We can see

4. Instead of letting EAAR predict the region proposal on new incoming frames through tracking, we directly let EAAR obtain the new region proposal, which makes EAAR strictly better.

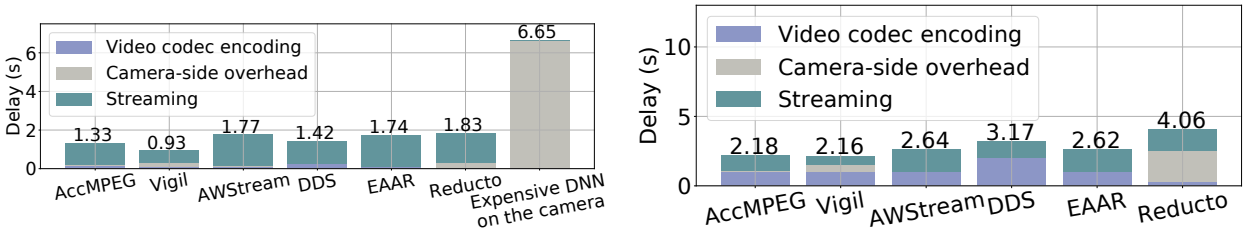
5. We do not evaluate region-proposal-based approach like EAAR and DDS on EfficientDet and Yolo since these DNNs have no region proposal (except that we evaluate DDS on Yolo since DDS develops specific heuristics to handle Yolo).

that AccMPEG outperforms the baselines: in terms of delay, AccMPEG has 10-43% smaller encoding delay than the best baselines with comparable accuracy. Vigil has lower streaming delay than AccMPEG, but it has low accuracy (many small objects are missed). AccMPEG is also 0.5-2% more accurate when compared to the non-server-driven baselines with lower streaming delay. Though some server-driven techniques have higher accuracy than AccMPEG on region-proposal-based DNNs like FasterRCNN, they are not applicable to DNNs that lack explicit region proposals like Yolo and EfficientDet.

We also evaluate AccMPEG's performance on semantic segmentation with FCN PyTorch as the final DNN. We find that AccMPEG has 20% higher accuracy than Reducto with lower streaming delay, or 5% lower streaming delay while maintaining higher accuracy than AWStream. This improvement may seem marginal, but the actual improvement of AccMPEG will be higher, since AWStream streams the highest-quality video whenever bandwidth permit to the server to identify the best video encoding decision, which can incur a high delay.

We also compare AccMPEG to the autoencoder Theis et al. (2017) for object detection on the highway dashcam videos. AccMPEG achieves an accuracy of 85%, but the autoencoder only achieves 62%. Moreover, the encoded frame size of AccMPEG is about 7KB, much less than that of autoencoder (240KB Github (a) per frame). As a result, the streaming delay of autoencoder is over 38 seconds. Thus, AccMPEG has a much better accuracy-delay tradeoff.

While AccMPEG improves performance in most cases, AccMPEG still has marginal or negative improve in some settings. For instance, AccMPEG cuts the delay of running $AccModel$ by running it once every $k = 10$ frames (§4), but when objects are moving very quickly (*e.g.,* monitoring camera feeds from a race car), this optimization can hurt accuracy. Moreover, because $AccModel$ uses the architecture of MobileNet-SSD Sandler et al. (2018) which works well on medium to large sized objects, it can perform poorly with with tiny objects (like the distant vehicles in drone videos).



(a) Delay breakdown on Intel Xeon Silver 4100

(b) Delay breakdown on Jetson Nano

Figure 6.2: Delay breakdowns of AccMPEG and baselines. AccMPEG achieves minimum streaming delay and has marginally higher encoding delay than codec encoding (used in AWStream, EAAR).

6.3 Encoding and streaming delays

Delay breakdown: Figure 6.2 shows the video codec encoding delay, camera-side extra compute delay, and the streaming delay of AccMPEG and those of the baselines based on the settings of Figure 6.1e (other settings have similar delay comparisons). We can see that AccMPEG has the lowest end-to-end delay on both camera-side hardware settings compared to all baselines except Vigi (whose accuracy is much lower than AccMPEG in Figure 6.1).

Camera-side compute cost: We then zoom in on the camera-side compute cost, which consists of encoding delay and the camera-side overhead delay. Figure 6.2 shows that AccMPEG's camera-side *AccModel* is cheaper than H264-based video encoding on the CPU, and is 20x cheaper than encoding on Jetson Nano. Moreover, AccMPEG's camera-side compute cost is lower than existing camera-side heuristics, such as Vigil and Reducto. Compared to Vigil, AccMPEG has lower compute cost, because it only runs the camera-side *AccModel* inference once every 10 frames, whereas Vigil performs camera-side inference on every frame. Compared to Reducto, AccMPEG does have higher encoding delay (since Reducto discards some frames and only encodes the remaining ones), but Reducto runs expensive camera-side logic on every frame[6] and thus has a much higher camera-side overhead than AccMPEG.

As a reference point, we also test the camera-side overhead of running the expensive DNN

6. Reducto performs Harris feature extraction, which contains several convolution filters and per-pixel eigen value decomposition and contributes 70% of the camera-side ovehead.

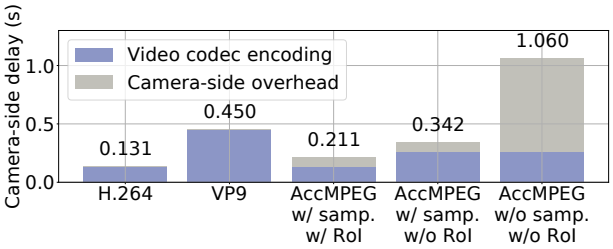


Figure 6.3: Breakdowns of camera-side delay: the delay of running *AccModel* is marginal (more so after the frame sampling optimization), compared to encoding delay of H.264 and VP9.

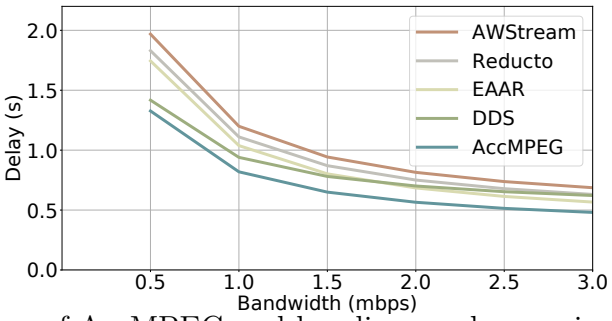


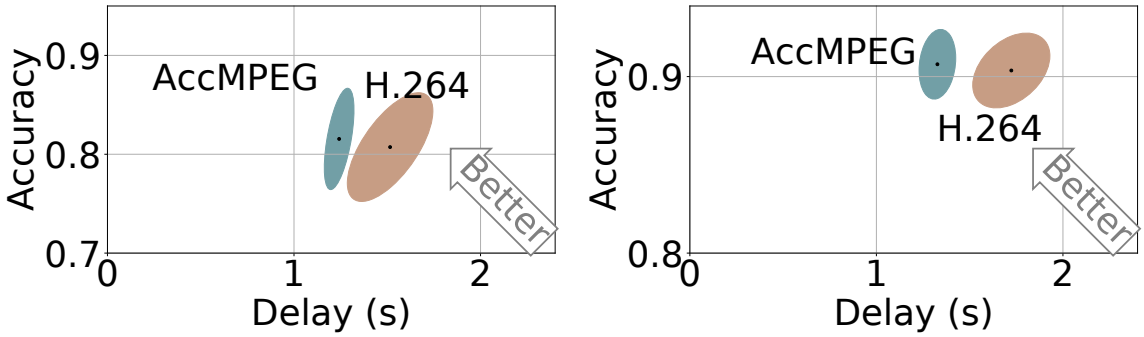
Figure 6.4: The delays of AccMPEG and baselines under varying network bandwidth.

on the camera: the camera-side delay is almost 7 seconds on CPU, and the expensive DNN cannot fit into the GPU memory of Jetson Nano.

Delay vs. bandwidth: Next, we benchmark the impact of network bandwidth on the video-analytics delay, we calculate the delay of AccMPEG and various baselines (except for Vigil, which has accuracy lower than 80% for most of the cases) under increasing network bandwidth. From Figure 6.4, we see that AccMPEG consistently achieves the lowest delay under different network bandwidth, though with more gains under low bandwidth.

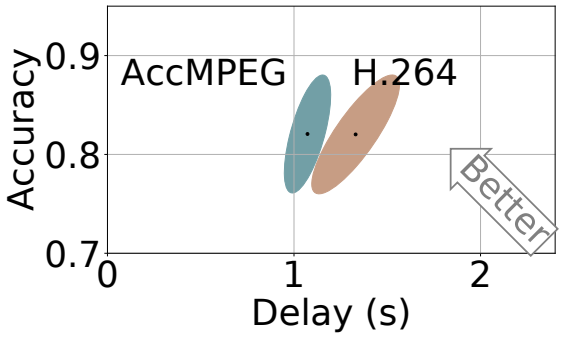
Delay optimizations of AccMPEG: AccMPEG uses two techniques to speed up its *AccGrad*-based encoding: (1) using region-of-interest encoding to encode the video (rather than encoding video twice as in DDS Du et al. (2020)), and (2) running the *AccModel* model once per 10 frames. Figure 6.3 shows their incremental reductions on AccMPEG's camera-side delay. The figure breaks down the encoding delay of AccMPEG into *AccGrad* prediction (*AccModel*) and the actual codec encoding, and as a reference point, it also shows the encoding delay of H.264, DDS and VP9[7]. As AccMPEG uses the *AccModel* (a shallow

7. We use the real-time encoding option of VP9.

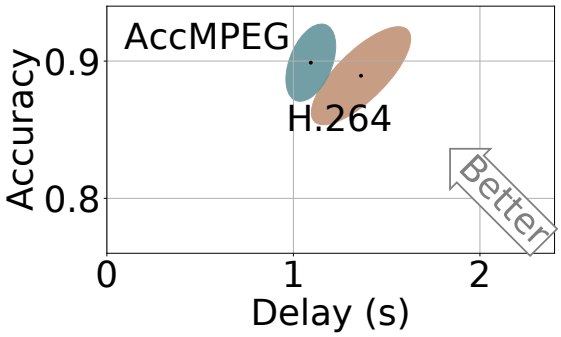


(a) FasterRCNN→EfficientDet (Highway)

(b) FasterRCNN→YoLo (Highway)



(c) FasterRCNN→EfficientDet (Driving)



(d) FasterRCNN→YoLo (Driving)

Figure 6.5: Even if we reuse the *AccModel* trained for a different final DNN, AccMPEG still offers decent performance gain over the best H.264 encoding scheme. DNN $A \rightarrow$ DNN B means the *AccModel* trained for A is reused to encode videos for B.

DNN) for its accuracy gradient model, the delay of accuracy gradient prediction is much smaller than prior work such as DDS which needs to actually run the final DNN. That said, it is sizable compared with the encoding delay. AccMPEG further reduces the delay by running *AccModel* on one frame every 10 frames, which allows AccMPEG to encode frames at 30fps on one Intel Xeon Silver 4100 CPU.

6.4 Fast *AccModel* training and reusing

Efficacy of reusing *AccModel*: From Figure 6.5, we see that in object detection, the *AccModel* trained on FasterRCNN also provides performance benefit on YoLo and EfficientDet across two different datasets. Similarly, in keypoint detection, the *AccModel* trained on KeypointRCNN-ResNet101 also generalizes to KeypointRCNN-ResNet50 on the surfing

Basic training pipeline (Figure 5.1(a))	453 minutes
After decoupling final DNN from training (Figure 5.1(b))	74.0 minutes
After 10× training data downsampling	7.40 minutes

Table 6.2: The training time of AccMPEG on 8 GPUs.

dataset. This demonstrates that AccMPEG can generalize to different vision models and provide better accuracy-delay trade-off, as long as the models are trained on the same dataset (as explained in §5).

Fast training: To benchmark the training speed, we train *AccModel* for FasterRCNN Ren et al. (2015) on 10x-downsampled COCO dataset COCO (2017). The training takes less than 8 minutes in total on 8 RTX 2080 Super GPU. From Table 6.2, we see that downsampling and the *AccGrad* abstraction reduce the overall training time by 60x.

6.5 Related work

Video analytics pipelines: There are many proposals to balance video analytics accuracy and its costs, including computing cost (*e.g.,* Zhang et al. (2017); Keahey et al. (2019); Xu et al. (2019); Canel et al. (2019); Xu et al. (2018)) as well as compression efficiency (*e.g.,* Du et al. (2020); Zhang et al. (2018); Liu et al. (2019); Zhang et al. (2015)). Besides those elaborated elsewhere in the paper, other techniques also try to discard unimportant frames Shen et al. (2017); Chen et al. (2015); Apicharttrisorn et al. (2019); Canel et al. (2019); Hsieh et al. (2018) or downsize the quality/framerate of an entire video segment Xu et al. (2019); Keahey et al. (2019); Zhang et al. (2017); Haris et al. (2018), offload inference of RoI bounding boxes Zhang et al. (2021) to remote servers, and raise bitrate in regions found by feeding DNN through the final DNN Galteri et al. (2018); Choi & Bajic (2018). Again, AccMPEG differs in that it introduces a cheap DNN-aware module to perform macroblock-level (rather than object-based) quality optimization and can quickly customize for any given final DNN.

Vision feature encoding: Other video encoders extract vision feature maps from the video and then compress the features (*e.g.,* Duan et al. (2020); Xia et al. (2020); Emmons

et al. (2019); Kang et al. (2017b); Matsubara et al. (2019)), with some efforts to standardize this approach Gao et al. (2021); VCM; CDV. Some also optimize for both vision accuracies and human visual quality (*e.g.,* Hu et al. (2020)). These video codecs explore a different design point than AccMPEG: (1) they assume that all video analytics DNNs share the same feature extractor (instead, AccMPEG treats each final DNN as just a blackbox); (2) they redesign both the encoder and the decoder (instead, AccMPEG run on any standard video codec); and (3) Their target vision tasks (*e.g.,* classification or action recognition) have more error tolerance when compressing feature maps (instead, AccMPEG handles more expensive tasks, like object detection, where any distortion on the feature maps matters).

Deep learning-based video compression: Some parallel efforts also replace the video codec by autoencoders (*e.g.,* Lu et al. (2019); Habibian et al. (2019); Agustsson et al. (2020); Rippel et al. (2019); Wu et al. (2018)). In a similar spirit, recent work trains differentiable video encoders to improve inference accuracy on the decompressed videos (*e.g.,* Chamain et al. (2021)). These DNN-based autoencoders do not directly apply, since these autoencoders are orders of magnitude more expensive than the standard video codes (used in AccMPEG): the fastest autoencoder runs at similar speed on GPU as H264 on CPU Rippel et al. (2019).

Adapting spatial scales in computer vision: The computer vision community also uses adaptive image sizing or partitioning to improve inference accuracy; *e.g.,* feature pyramid networks (FPN) Lin et al. (2017b) and BiFPN Tan et al. (2020) extract feature maps from multiple resolutions to detect small objects. Others use attention mechanisms to focus computation on regions with potential objects Wang et al. (2017); Ozge Unel et al. (2019); Ržička & Franchetti (2018); Fan et al. (2019). While AccMPEG shares similar insights, it optimizes the video compression efficiency, rather than computation complexity.

CHAPTER 7

CONCLUSION

In this work, we present AccMPEG, a new video codec for video analytics that improves the tradeoffs between inference accuracy and compression efficiency for a variety of computer vision tasks. It does so by treating any vision DNN as a *differentiable black box* and infers the *accuracy gradients* to identify where in the frame the DNN's inference result is highly sensitive to the encoding quality level and thus needs to be encoded with high quality. Our evaluation of AccMPEG over three vision tasks shows that compared with the state-of-the-art baselines, AccMPEG reduces upto 43% of the delay while increasing accuracy by upto 3% at the same time. Moreover, AccMPEG's camera-side overhead is almost the same as those of the traditional codecs.

BIBLIOGRAPHY

Compact descriptors for video analysis (the moving picture experts group). https://mpeg.chiariglione.org/tags/cdva.

Video coding for machines (the moving picture experts group). https://mpeg.chiariglione.org/standards/exploration/video-coding-machines.

AccMPEG. Accmpeg dataset link. https://docs.google.com/spreadsheets/d/15sJ1yt86OuNqVx-WjFmi

Agustsson, E., Minnen, D., Johnston, N., Balle, J., Hwang, S. J., and Toderici, G. Scale-space flow for end-to-end optimized video compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8503–8512, 2020.

Amazon. Amazon.com: Nvidia jetson nano 2gb developer kit (945-13541-0000-000) : Electronics. https://www.amazon.com/NVIDIA-Jetson-Nano-Developer-945-13541-0000-000/dp/B08J157LH $hyprod - 20linkCode = df0hvadid = 475750632217hvpos = hvnetw = ghvrand = 872618593832062796 4hvpone = hvptwo = hvqmt = hvdev = chvdvcmdl = hvlocint = hvlocphy = 9021740hvtargid = pla - 1010773195372psc = 1.(Accessedon10/12/2021)$.

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

Canel, C., Kim, T., Zhou, G., Li, C., Lim, H., Andersen, D. G., Kaminsky, M., and Dulloor, S. R. Scaling video analytics on constrained edge nodes. *arXiv preprint arXiv:1905.13536*, 2019.

Chamain, L. D., Racapé, F., Bégaint, J., Pushparaja, A., and Feltman, S. End-to-end optimized image compression for machines, a study. In *2021 Data Compression Conference (DCC)*, pp. 163–172. IEEE, 2021.

ChameleonHardware. Hardware info — chameleon. https://chameleoncloud.org/experiment/chiedge/hardware-info/, 2021. (Accessed on 10/15/2021).

Chen, T. Y.-H., Ravindranath, L., Deng, S., Bahl, P., and Balakrishnan, H. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pp. 155–168. ACM, 2015.

Choi, H. and Bajic, I. V. High efficiency compression for object detection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1792–1796. IEEE, 2018.

COCO. Coco dataset. https://cocodataset.org/home, 2017.

Coding, H. E. V. and Rec, I. H. 265 and iso, 2013.

Dai, X., Kong, X., Guo, T., and Huang, Y. Cinet: Redesigning deep neural networks for efficient mobile-cloud collaborative inference. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 459–467. SIAM, 2021.

Detectron2. Detectron2 model zoo. https://github.com/facebookresearch/detectron2.

Du, K., Pervaiz, A., Yuan, X., Chowdhery, A., Zhang, Q., Hoffmann, H., and Jiang, J. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 557–570, 2020.

Duan, L., Liu, J., Yang, W., Huang, T., and Gao, W. Video coding for machines: A paradigm of collaborative compression and intelligent analytics. *IEEE Transactions on Image Processing*, 29:8680–8695, 2020.

Emmons, J., Fouladi, S., Ananthanarayanan, G., Venkataraman, S., Savarese, S., and Winstein, K. 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*, pp. 27–32, 2019.

Fan, D.-P., Wang, W., Cheng, M.-M., and Shen, J. Shifting more attention to video salient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8554–8564, 2019.

Galteri, L., Bertini, M., Seidenari, L., and Del Bimbo, A. Video compression for object detection algorithms. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 3007–3012. IEEE, 2018.

Gao, W., Liu, S., Xu, X., Rafie, M., Zhang, Y., and Curcio, I. Recent standard development activities on video coding for machines. *arXiv preprint arXiv:2105.12653*, 2021.

Github. Lossy image compression with compressive autoencoders, source code. `https://github.com/alexandru-dinu/cae`, a.

Github. Reducto code base. `https://github.com/reducto-sigcomm-2020/reducto`, b.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

GoodVision. Goodvision: Smart traffic data analytics. `https://goodvisionlive.com/`, 2021.

GrandViewResearch. Video analytics market size worth $9.4 billion by 2025 — cagr: 22.8%. `https://www.grandviewresearch.com/press-release/global-video-analytics-market`, 2018. Accessed: 2019-2-13.

Habibian, A., Rozendaal, T. v., Tomczak, J. M., and Cohen, T. S. Video compression with rate-distortion autoencoders. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7033–7042, 2019.

Haris, M., Shakhnarovich, G., and Ukita, N. Task-driven super resolution: Object detection in low-resolution images. *arXiv preprint arXiv:1803.11316*, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

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

Hsieh, K., Ananthanarayanan, G., Bodik, P., Venkataraman, S., Bahl, P., Philipose, M., Gibbons, P. B., and Mutlu, O. Focus: Querying large video datasets with low latency and low cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 269–286, 2018.

Hu, Y., Yang, S., Yang, W., Duan, L.-Y., and Liu, J. Towards coding for human and machine vision: A scalable image coding approach. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6. IEEE, 2020.

intuVision. intuvision va traffic use case. `https://www.intuvisiontech.com/intuvisionVA`$_s$*olutions/intu*

Johnson, J., Alahi, A., and Fei-Fei, L. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pp. 694–711. Springer, 2016.

Kang, D., Emmons, J., Abuzaid, F., Bailis, P., and Zaharia, M. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11): 1586–1597, 2017a.

Kang, D., Bailis, P., and Zaharia, M. Blazeit: optimizing declarative aggregation and limit queries for neural network-based video analytics. *arXiv preprint arXiv:1805.01046*, 2018.

Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *Acm Sigplan Notices*, 52(4):615–629, 2017b.

Keahey, K., Riteau, P., Stanzione, D., Cockerill, T., Mambretti, J., Rad, P., and Ruth, P. *Chameleon: A Scalable Production Testbed for Computer Science Research*, pp. 123–148. 05 2019. ISBN 9781351036863. doi: 10.1201/9781351036863-5.

Keahey, K., Anderson, J., Zhen, Z., Riteau, P., Ruth, P., Stanzione, D., Cevik, M., Colleran, J., Gunawi, H. S., Hammock, C., Mambretti, J., Barnes, A., Halbach, F., Rocha, A., and Stubbs, J. Lessons learned from the chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

Li, Y., Padmanabhan, A., Zhao, P., Wang, Y., Xu, G. H., and Netravali, R. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the*

applications, technologies, architectures, and protocols for computer communication, pp. 359–376, 2020.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017a.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017b.

Liu, H., Shen, H., Huang, L., Lu, M., Chen, T., and Ma, Z. Learned video compression via joint spatial-temporal correlation exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 11580–11587, 2020.

Liu, L., Li, H., and Gruteser, M. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, pp. 1–16, 2019.

Lu, G., Ouyang, W., Xu, D., Zhang, X., Cai, C., and Gao, Z. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11006–11015, 2019.

Matsubara, Y., Baidya, S., Callegaro, D., Levorato, M., and Singh, S. Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pp. 21–26, 2019.

Microsoft. Traffic video analytics – case study report. `https://www.microsoft.com/en-us/research/publication/traffic-video-analytics-case-stud` 2019.

Mnih, V., Heess, N., Graves, A., et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pp. 2204–2212, 2014.

Mullapudi, R. T., Chen, S., Zhang, K., Ramanan, D., and Fatahalian, K. Online model distillation for efficient video inference. In *Proc. of ICCV*, 2019.

OpenSignal. Explaining the huge gap between fastest and slowest 4g upload speeds in the us — opensignal. `https://www.opensignal.com/2018/06/20/explaining-the-huge-gap-between-fastest-and-slow` 2018. (Accessed on 10/05/2021).

Ozge Unel, F., Ozkalayci, B. O., and Cigla, C. The power of tiling for small object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

PyTorch. Fcn-resnet101 — pytorch. `https://pytorch.org/hub/pytorch`$_{v}ision_{f}cn_{r}esnet101/.(Accessedo$r

Redmon, J. and Farhadi, A. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Rippel, O., Nair, S., Lew, C., Branson, S., Anderson, A. G., and Bourdev, L. Learned video compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3454–3463, 2019.

Ržička, V. and Franchetti, F. Fast and accurate object detection in high resolution 4k and 8k video using gpus. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–7. IEEE, 2018.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

SecurityInfoWatch. Market for small IP camera installations expected to surge. http://www.securityinfowatch.com/article/10731727/, 2012.

SecurityInfoWatch. Data generated by new surveillance cameras to increase exponentially in the coming years. http://www.securityinfowatch.com/news/12160483/, 2016.

Shen, H., Han, S., Philipose, M., and Krishnamurthy, A. Fast video classification via adaptive cascading of deep models. *arXiv preprint*, 2017.

Slate. Humans can't watch all the surveillance cameras out there, so computers are. https://slate.com/technology/2019/06/video-surveillance-analytics-software-artificial-
2019. Accessed: 2019-3-4.

StreamingMedia. Video compression for machines: The next frontier. https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=133860, 2019.

Tan, M., Pang, R., and Le, Q. V. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

Theis, L., Shi, W., Cunningham, A., and Huszár, F. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

TrafficTechnologyToday. Ai traffic video analytics platform being developed. https://www.trafficktechnologytoday.com/news/traffic-management/ai-traffic-video-analyt
2019.

TrafficVision. Trafficvision: Traffic intelligence from video. http://www.trafficvision.com/, 2021.

VisionZero. The vision zero initiative. http://www.visionzeroinitiative.com/.

Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., and Tang, X. Residual attention network for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3156–3164, 2017.

Wang, J., Zheng, Y., Ni, Y., Xu, C., Qian, F., Li, W., Jiang, W., Cheng, Y., Cheng, Z., Li, Y., et al. An active-passive measurement study of tcp performance over lte on high-speed rails. In *The 25th Annual International Conference on Mobile Computing and Networking*, pp. 1–16, 2019a.

Wang, Y., Wang, W., Zhang, J., Jiang, J., and Chen, K. Bridging the edge-cloud barrier for real-time advanced vision analytics. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019b.

Wiegand, T., Sullivan, G. J., Bjontegaard, G., and Luthra, A. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003. doi: 10.1109/TCSVT.2003.815165.

Wu, C.-Y., Singhal, N., and Krahenbuhl, P. Video compression through image interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 416–431, 2018.

Xia, S., Liang, K., Yang, W., Duan, L.-Y., and Liu, J. An emerging coding paradigm vcm: A scalable coding approach beyond feature and signal. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6. IEEE, 2020.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations

for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Xu, M., Zhu, M., Liu, Y., Lin, F. X., and Liu, X. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 129–144. ACM, 2018.

Xu, T., Botelho, L. M., and Lin, F. X. Vstore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pp. 1–17, 2019.

Zhang, B., Jin, X., Ratnasamy, S., Wawrzynek, J., and Lee, E. A. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 236–252. ACM, 2018.

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

Zhang, T., Chowdhery, A., Bahl, P. V., Jamieson, K., and Banerjee, S. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 426–438. ACM, 2015.

Zhang, W., He, Z., Liu, L., Jia, Z., Liu, Y., Gruteser, M., Raychaudhuri, D., and Zhang, Y. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pp. 201–214, 2021.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

APPENDIX A

APPENDIX

A.1 Optimal quality assignment analysis

We formalize the spatial quality assignments in this section and derive the near-optimal solution through *AccGrad*.

A.2 Formalize quality assignment

To make the discussion more concrete, we split each $W \cdot H$ frame into $w \cdot h$ grids of 16x16 blocks ($W = 16w, H = 16h$) and assign each block either a high quality or a low quality. We now consider this problem: what is the best quality assignment for these 16x16 blocks that maximizes the accuracy subject to no more than c blocks encoded in high quality. Formally, it searches for a binary mask $\mathbf{M}$ of size $w \cdot h$ ($\mathbf{M}_{x,y} = 1$ means block x, y is in high quality), such that

$$\max \quad Acc\left(D(\mathbf{M} \times \mathbf{H} + (\mathbf{1} - \mathbf{M}) \times \mathbf{L}), D(\mathbf{H})\right) \tag{A.1}$$

$$\text{s.t.} \quad \|\mathbf{M}\| \leq c \tag{A.2}$$

where $\mathbf{H}$ and $\mathbf{L}$ are the high-quality encoding and the low-quality encoding of each frame[1], $D : \mathbb{I} \mapsto \mathbb{O}$ returns the DNN inference result ($\mathbb{I}$ and $\mathbb{O}$ are the spaces of input frames and DNN output), and $Acc : \mathbb{O} \times \mathbb{O} \mapsto \mathbb{R}$ returns the accuracy of D's output on a compressed frame by comparing its similarity with D's output on the high quality image $D(\mathbf{H})$.

This formulation involves two simplifying assumptions: the 16x16 blocks may be suboptimal boundaries between quality levels, and it restricts the encoding to only two quality levels. That being said, we believe that analyzing this formulation is still valuable for two

1. The dimension of a frame is the same for different QP values, so $\mathbf{H}$ and $\mathbf{L}$ have the same dimension.

reasons. First, the block granularity of 16x16 is on par with the block sizes employed in H.264 and H.265, which means more fine-grained blocks will not have much impact on the encoded video size. Second, the use of two quality levels does subsume many recent solutions (*e.g.,* Zhang et al. (2015); Du et al. (2020); Zhang et al. (2018); Chen et al. (2015); Li et al. (2020)) which use two or fewer quality levels.

A.3 Deriving near-optimal solution through *AccGrad*

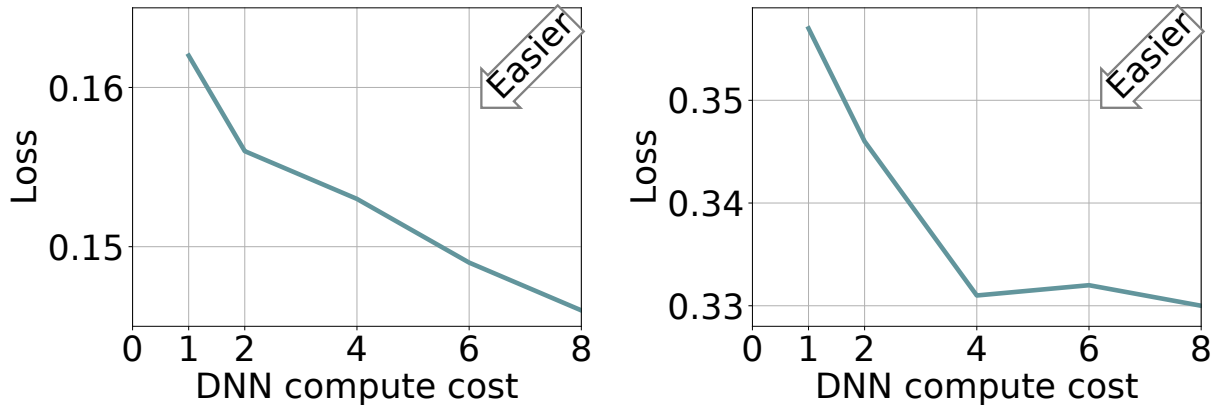
In this section, we "derive" the quality assignment from the pixel values of a frame. To this end, we first rewrite the objective of the idealized algorithm in Eq (A.1) in a differentiable form as follows. We notice that max $Acc(D(\mathbf{X}), D(\mathbf{H}))$ (where $\mathbf{X} = \mathbf{M} \times \mathbf{H} + (\mathbf{1} - \mathbf{M}) \times \mathbf{L}$) is equivalent to the following (note that the first term of Eq A.3 a constant to $\mathbf{M}$):

$$\begin{aligned}
\min \quad & Acc(D(\mathbf{1} \times \mathbf{H}), D(\mathbf{H})) - Acc(D(\mathbf{X}), D(\mathbf{H})) && \text{(A.3)}\\
=& \left\langle \frac{\partial Acc(D(\mathbf{X}'), D(\mathbf{H}))}{\partial \mathbf{X}'}, (\mathbf{1} \times \mathbf{H} - \mathbf{X}) \right\rangle_{\mathrm{F}} && \text{(A.4)}\\
\approx& \left\langle \frac{\partial Acc(D(\mathbf{L}), D(\mathbf{H}))}{\partial \mathbf{L}}, (\mathbf{1} \times \mathbf{H} - \mathbf{X}) \right\rangle_{\mathrm{F}} && \text{(A.5)}\\
=& \left\langle \frac{\partial Acc(D(\mathbf{L}), D(\mathbf{H}))}{\partial \mathbf{L}}, (\mathbf{1} - \mathbf{M}) \times (\mathbf{H} - \mathbf{L}) \right\rangle_{\mathrm{F}}, && \text{(A.6)}
\end{aligned}$$

where $\langle \cdot, \cdot \rangle_{\mathrm{F}}$ is the Frobenius inner product. Eq (A.4) uses Lagrange's Mean Value Theorem, where $\mathbf{X}'$ lies between $\mathbf{H}$ and $\mathbf{X}$. In Equation (A.6), since the value of $\mathbf{M}$ is identical inside each block, each block (x, y) contributes the following value to Eq (A.6):

$$\begin{aligned}
& (1 - \mathbf{M}_B) \sum_{i \in B} \left. \frac{\partial Acc(D(\mathbf{L}), D(\mathbf{H}))}{\partial \mathbf{L}} \right|_i \cdot (\mathbf{H}_i - _i)\\
\leq & \delta \cdot (1 - \mathbf{M}_B) \cdot \sum_{i \in B} \left\| \left. \frac{\partial Acc(D(\mathbf{L}), D(\mathbf{H}))}{\partial \mathbf{L}} \right|_i \right\|_2, && \text{(A.7)}
\end{aligned}$$

where δ denotes the maximum element in $\mathbf{H}_{i,j} - \mathbf{L}_{i,j}$. This upper bound holds when the gradient of $Acc(D(\mathbf{X}), D(\mathbf{H}))$ at $\mathbf{X}$ is non-negative at all pixels in the block, which is a reasonable assumption since changing pixel values closer to the high quality encoding will



(a) Traditional loss (y-axis) used to train segmentation models.

(b) The loss function (y-axis) used to train *AccModel*.

Figure A.1: Comparing the training curve of a traditional segmentation model and that of *AccModel*: with more DNN layers (higher DNN compute cost), the traditional segmentation loss drops slowly, whereas the loss function of *AccModel* (*i.e.,* low-dimensional binary segmentation with a higher tolerance to false positives) drops very quickly, which indirectly suggests that a cheap model might suffice to train an accurate enough *AccModel*.

likely make the inference result more similar to $D(\mathbf{X})$.

A.4 Empirical evidence on how false-positive-tolerance reduces the cost of *AccModel*

To empirically support that false positive tolerance can reduce the compute demand of segmentation task, we train a series of DNNs with compute power [1,2,4,6,8]x on the same dataset with two different losses: traditional segmentation loss and false-positive-tolerant segmentation loss that applies only 1/4 penalty to false positives than to false negatives. From Figure A.1, we see that the loss of the 4x DNN is much higher than the loss of the 8x one under traditional segmentation loss (see Figure A.1a), but the 4x DNN can achieve similar loss as the 8x DNN under false-positive-tolerant loss (see Figure A.1b). This indicates that false-positive-tolerant loss can efficiently reduce the compute demand of the segmentation task so that even a cheap DNN can achieve similar performance as an expensive DNN.