AccDecoder: Accelerated Decoding for Neural-enhanced Video Analytics

Tingting Yuan[†§], Liang Mi[‡§], Weijun Wang[†‡], Haipeng Dai[‡*], Xiaoming Fu[†]
[†]University of Göttingen, Germany; [‡]Nanjing University, China
{tingting.yuan,weijun.wang,fu}@cs.uni-goettingen.de, liangmi@smail.nju.edu.cn, haipengdai@nju.edu.cn

***Abstract*—The quality of the video stream is key to neural network-based video analytics. However, low-quality video is inevitably collected by existing surveillance systems because of poor quality cameras or over-compressed/pruned video streaming protocols, *e.g.*, as a result of upstream bandwidth limit. To address this issue, existing studies use quality enhancers (*e.g.*, neural super-resolution) to improve the quality of videos (*e.g.*, resolution) and eventually ensure inference accuracy. Nevertheless, directly applying quality enhancers does not work in practice because it will introduce unacceptable latency. In this paper, we present AccDecoder, a novel accelerated decoder for real-time and neural-enhanced video analytics. AccDecoder can select a few frames adaptively via Deep Reinforcement Learning (DRL) to enhance the quality by neural super-resolution and then up-scale the unselected frames that reference them, which leads to 6-21% accuracy improvement. AccDecoder provides efficient inference capability via filtering important frames using DRL for DNN-based inference and reusing the results for the other frames via extracting the reference relationship among frames and blocks, which results in a latency reduction of 20-80% than baselines.**

***Index Terms*—Video analytics, super-resolution, deep reinforcement learning**

I. Introduction

Advances in computer vision offer tremendous opportunities for autonomous analytics of videos generated by pervasive video cameras. Deep Neural Networks (DNNs) [1]–[4] have been developed to dramatically improve the accuracy for various vision tasks, while introducing stringent demands on computational resources. Due to the compute-resource shortage of commercial cameras, videos need to be streamed to powerful servers for inference, which is called distributed video analytics pipeline (VAP) [5], [6].

Nevertheless, providing highly accurate video analytics remains challenging for the state-of-the-art distributed VAPs. Since most methods for video analytics currently rely on high-resolution videos, it is difficult to analyze low-quality videos, such as object detection at low resolutions. For example, the accuracy of Faster R-CNN [3], a modern DNN-based inference method, can only achieve around 56% accuracy for videos in 360p and 61% accuracy for videos in 540p which are collected by [7]. However, low-quality videos are inevitably collected by existing surveillance systems. One of the reasons is that existing low-quality collectors can only capture low-resolution frames. For example, New York city's department of transportation [8] has made videos from all 752 traffic cameras to the public; however, the videos are transmitted at an extremely low resolution (240p) due to the default configuration of cameras [9]. Another reason is that current video streaming protocols over-compress/prune videos due to upstream bandwidth limitations. For example, AWStream [10] aggressively reduces the resolution of the video from 540p to 360p and the frame rate from 1 to 0.83. It eventually brings about a 66% saving of bandwidth with a reduced accuracy from 61% to 54%.

To address this challenge, some VAPs [11], [12] try to utilize image enhancement models like Super Resolution (SR) [2], [13] and Generative Adversarial Network (GAN) [14] to enhance frames in videos before feeding them into the inference model. This idea is inspired by the observation from the computer vision community – running object recognition-related tasks on high-resolution images can largely improve the detection accuracy [13]. However, the video enhancement via DNN-aware image enhancement models introduces extra latency, resulting in around 500 ms end-to-end latency [15] for each frame, which is far from the real-time requirement (*e.g.*, less than 15-30 ms for real-time object recognition [6]).

Although existing DNN-aware video enhancement provides a promising way to improve the inference accuracy [16], there is still much room for improvement. First, prior video enhancement mechanisms are largely agnostic to video contents, treating each received frame equally, but not all the frames need to be enhanced. For example, only the frames containing vehicles are valuable for traffic flow analysis; on the contrary, enhancing frames with empty streets is worthless but only increases system latency. Therefore, content-agnostic enhancement mechanisms cannot avoid being suboptimal. Second, although new DNN frameworks are designed to accurately recognize important frames (*e.g.*, [17], [18]), they are too heavy to achieve low latency. Third, decoding all the frames for analytics is computationally intensive and time-consuming, and video encoding contains plenty of unexploited but handy information to capture the important frames, such as motion vectors (MVs) and residuals. We argue that the codec information, although inaccurate, is valuable to reveal which content is important, thereby speeding up video analytics.

Motivated by the above insights, we present AccDecoder, a novel accelerated video streaming decoder for real-time and neural-enhanced video analytics. AccDecoder is a content-aware DNN-integrated video decoder utilizing codec information to select a few frames for quality enhancement, which is called as *anchor frames* and some frames for DNN-aware

*Corresponding author. §Equal contributors.

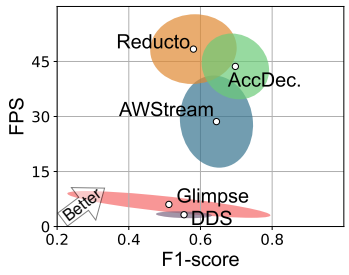


Fig. 1: Example results of AccDecoder vs. baselines.

inference, which is called as *inference frames*. In particular, AccDecoder applies SR to anchor frames and transfers the high quality of frames/blocks to benefit the entire video via extracting the frame and block reference relationships; it further leverages codec information to reuse the results of inference frames for acceleration.

Challenge and solution. AccDecoder needs to adapt to various quality of videos to enable robust and real-time video analytics. Since SR and DNN-based inference is time-consuming, an accuracy-latency tradeoff must be made carefully to keep low latency without drastically compromising the accuracy. Our preliminary study (see more in Section II and III) shows that AccDecoder needs adaptive metrics for anchor and inference frame selection according to factors like video content and quality. In other words, various videos (or even different chunks of the same video) demand different settings for frame selection, but a static setting is not adaptive to the varying contents of videos. To address this issue, we leverage deep reinforcement learning (DRL) [19]–[21], which has been widely used to solve dynamic and sequential problems [22], [23]. Specifically, AccDecode enables adaptive settings via DRL in frame selection to accelerate the video analytics and achieve a good tradeoff between accuracy and latency.

Our contributions are summarized as follows.

- We design a novel content-aware DNN integrated video decoder that is resilient and robust to the quality of videos. For example, for a 540p crossroad video collected by [7], AccDecoder can improve 10-38% accuracy compared with the state-of-the-art VAPs (see Fig. 1).
- We exploit temporal redundancies within a video and codec information to drastically increase the speed of analytics. For example, AccDecoder performs 1.5-8.0 times faster compared with AWStream [10], DDS [5], and Glimpse [24] (see Fig. 1).

The rest of the paper is organized as follows. We first present the background and motivation in Section II. Then we discuss AccDecoder's key design in Section III, followed by our implementation in Section IV. The experimental studies are demonstrated in Section V. In Section VI, we introduce some related work. We conclude this paper in Section VII.

II. Background and Motivation

We introduce distributed VAPs and video codec, followed by performance requirements of VAPs that drive our design (*i.e.*, high accuracy, low end-to-end latency, and low overheads). We then elaborate on why prior solutions struggle to meet the three requirements simultaneously.

A. Background

Distributed Video Analytics. The proliferation of video analytics is facilitated by the advances of deep learning and the low prices of high-resolution network-connected cameras. However, the accuracy improvement from deep learning comes at a high computational cost. Although state-of-the-art smart cameras can support deep learning methods, the current surveillance and traffic cameras show only suboptimal use of resources. For example, DNNCam [25] that ships with a high-end embedded NVIDIA TX2 GPU [26] costs more than $2000 while the price of deployed traffic cameras today ranges $40-$200. These cameras are typically loaded with a single-core CPU, only providing scarce compute resources. Because of this huge gap, typical VAPs follow a distributed architecture. A typical distributed VAP architecture includes a filter and an encoder on the camera side and a decoder and an inference model on the server side, as illustrated in Fig. 2. In live analytics, video frames are continuously encoded and sent to a remote server that runs the inference DNN to analyze the video in an online fashion. For example, a vehicle detection pipeline consists of a front-end traffic camera (which compresses and streams live videos to a compute-powerful edge/cloud GPU server upon wire/wireless networks) and a back-end server (which decodes received video into frames and feeds them into inference models like Faster R-CNN [3] to detect vehicles). Such an architecture brings challenges in bandwidth cost, and thus some schemes rely on aggressively pruning videos (via *e.g.*, reconfiguration, filtering) to meet upstream bandwidth limitations.

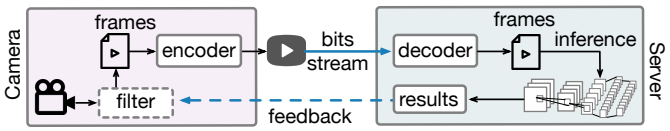


Fig. 2: Distributed video analytics pipeline.

Video codec is the key technology in distributed VAPs. It comprises an encoder and a decoder, a software/hardware program used to compress/decompress video files for easier storage or network delivery. The encoder compresses video data and wraps them into common video formats (*e.g.*, H.264 [27]), while the decoder decompresses the compressed video data into frames before post-processing (*e.g.*, playback or analysis). This compression process is usually lossy, which strikes a balance between the video quality and the compression ratio according to the self-preference of codecs and encoding settings of users (*e.g.*, bitrate, frame rate, and group of pictures). We take H.264, one of the most popular codecs, as an example to explain the compression process. During encoding, each video frame is first divided into non-overlapped macroblocks (16×16 pixels), then to each macroblock, the

encoder searches for the optimal compression method (including the block division types and encoding types of each block) according to the pixel-level similarity and encoding settings. One macroblock may be further divided into non-overlapped blocks (8×8 or 8×16 pixels) encoded with intra- or inter-frame types. The intra-coded block is encoded using the reference block with the most pixel-value similarity, and the offset between these two blocks is encoded into an MV with a residual. With the same procedure, the inter-coded block locates the most pixel-value similar block searched by the reference index and the MV from other frames. An MV indicates the spatial offset between the target block and its reference, while the difference in pixel values of two blocks is encoded as the residual for decoding.

An ideal distributed VAP should meet three goals:

- **High accuracy**. Inspired by the success of image enhancement methods in video streaming for QoE improvement [28]–[31], researchers try to use image enhancement for machine-centric video analytics [15], [31], [32]. For example, [15] leverages SR to enhance image details for robotics applications, while [31] embeds GAN on Google Glasses to generate facial features for face recognition. The experimental results from [15], [31] demonstrate that image enhancement improves inference accuracy.
- **Low latency**. The latency of decoding the video (*i.e.*, decoding latency), inference, and streaming the video to the server (*i.e.*, streaming latency) should be low. Previous works focus on reducing the size of streaming to reduce the streaming latency (*e.g.*, Reducto [3]), learning and filtering key frames for inference, and utilizing MVs to reuse the other frames.
- **Low bandwidth cost**. We define the total size of a video file delivered from the camera to the server of each VAP as its bandwidth cost.

B. Motivation

Limitations of previous work. Although a couple of bandwidth-saving and accuracy-improvement approaches have been developed, three significant limitations remain.

- **Adaptive encoding in cameras**. A camera may leverage light-weight DNN [33] or heuristic methods (*e.g.*, inter-frame pixel-level difference [24]) to distinguish and prune frames/regions without labelled information. However, these cheap methods may cause false positive (*e.g.*, pixel-level distance changes by background may trigger a camera to send many frames) and false negative (*e.g.*, cheap object detection model may miss small appeared objects), thus increasing bandwidth cost or reducing the inference accuracy. Server-side decision-making controls the camera's actions with feedback from the cloud. For example, according to the servers' instruction, the camera in [5] iteratively delivers the region of interest in a higher resolution. The server-side decision-making introduces extra latency, especially due to the information delivery between the camera and the cloud crossing a wide area network.
- **Image enhancement in servers**. Image enhancement contributes to higher accuracy but also causes higher latency. Although recent studies have successfully leveraged image enhancement to increase QoE and inference accuracy, they still suffer from high latency. The root cause is that image enhancement models are much heavier than other computer vision models. For instance, the complexity of the SR model is as high as 1000× heavier than image classification, and object detection models in terms of MultAdds [12], as SR outputs high-resolution (HR) images whereas others output labels or Bounding boxes (Bbox). In other words, naively enhancing each frame is not practical in real-time video analytics applications.

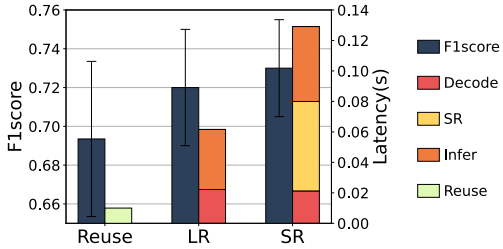


Fig. 3: Accuracy and latency using different schemes.

As shown in Fig. 3, our preliminary study confirms the challenge in the tradeoff between accuracy and latency in video analytics. Due to resource restrictions, to use existing techniques in real-time and provide high accuracy, servers need to adaptively shift multiple pipelines, including inference after SR using HR frames, inference with low-resolution frames (LR), and reuse the last inference results. However, conventional algorithms (*e.g.*, k-nearest neighbor (KNN) used in [6]) are largely suboptimal as they ignore the temporal relationship among frames and the possibility of transferring high-quality frames to the whole video. Therefore, we aim to design an efficient decoder to schedule the multiple pipelines for video analytics adaptively.

III. System Design

In this section, we present the design goals and our solution details of AccDecoder.

Design goals. We take a pragmatic stance to focus on the server-side decoder because most of the installed surveillance or traffic cameras only have cheap CPUs without programmable ability [6]; besides, rich information that may improve VAPs' performance in the decoder has not been excavated. In this context, we propose AccDecoder, a portable tool/decoder which can be plugged into any VAPs for video streaming analytics to achieve high accuracy, limited latency (*e.g.*, 30 ms), and low-resource goals simultaneously. As illustrated in Fig. 4, AccDecoder achieves these goals via the following three mechanisms: 1) Pipeline ❶ leverages *SR* model enhancing a small set of LR *anchor frames* to HR ones to achieve high accuracy. 2) Pipeline ❷ and Pipeline ❸ extract the codec information (*e.g.*, frames reference relationship, MVs, and residuals) from the decoder, then utilize them to *transfer* the gains of enhanced anchor frames and *reuse* DNN inference

results (*e.g.*. Bbox in object detection) onto the entire video respectively. *Transfer* and *reuse* amortize the computational overhead of SR and inference across the entire video and thus achieve low latency. 3) The *scheduler* classifies all frames into three subsets, and each executes one of three pipelines. It can greatly reduce latency and computational cost by exploiting the content features of the key frames (*e.g.*, the intra-coded frame) and the change of codec information (*e.g.*, residuals of continuous frames).

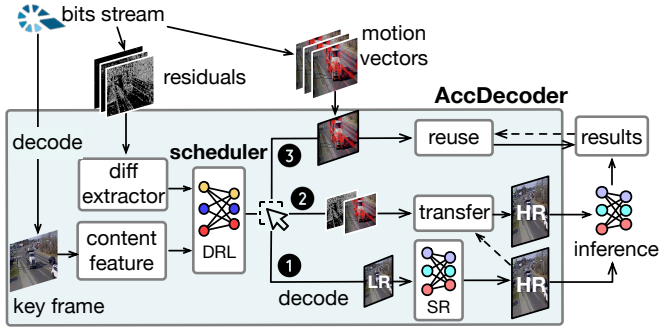


Fig. 4: Architecture of AccDecoder.

Path towards the goal. We seek answers to the following pivotal questions, which lead to our key design choices. Q1 — How to effectively transfer the gains of SR to up-scale non-anchor frames? Q2 — How to reuse the results of inference to the other frames? Q3 — How to assign appropriate decoding pipelines to frames in a fine spatial granularity to achieve a better accuracy-latency tradeoff than baselines?

Q1 – How to transfer gains of SR to non-anchor frames?
Approach: Pipeline ❶ + ❷. To make the best of reuse, AccDecoder enhances *anchor frames* with the SR model and caches the output (see ❶ in Fig. 5); then it transfers the enhancement benefit to non-anchor frames with the reference information and the cached outputs (see ❷ in Fig. 5). This approach follows the same findings in [2] and [30], where most of the latency of SR occurs at the last couple of layers. Namely, caching and reusing the final output (*i.e.*, high-resolution images) is most effective in achieving low latency.

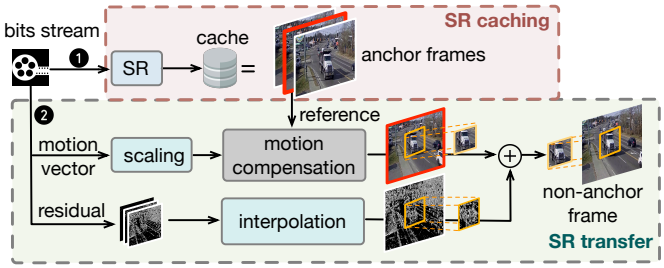


Fig. 5: Process of SR gain transfer inspired by [30] and [27].

Fig. 5 illustrates the process of transferring SR gains to a non-anchored frame for the inter-coded type. Modern video codec encodes/decodes frames on the basis of non-overlapped *inter-* and *intra-coded* blocks (§II-A). AccDecoder uses the reference index, MVs, and the residual in the codec information to decode a target block. The process is the same as normal decoding except for the additional SR, scaling, and interpolation modules (blue boxes in Fig. 5). First, AccDecoder selects the reference blocks among cached anchor frames following the inference index. Next, AccDecoder up-scales the MV with the same amplification factor as SR (*e.g.*, from 270p to 1080 is 4). Following the MV, AccDecoder transfers the SR gain from the reference block in the cached frame to the target one. At last, AccDecoder up-scales the residual by light-weight interpolation (*e.g.*, bilinear or bicubic), accumulates it to the transferred block to output the HR block, and pastes on the non-anchor frame. To the intra-coded blocks without the cached reference anchor frame, AccDecoder directly decodes and up-scales then by interpolation. Fortunately, with our carefully designed scheduler, most intra-coded blocks are assigned to the SR pipeline; the impact of the minority of interpolated intra-coded blocks can be negligible.

Q2 — How to reuse inference results for non-inference frames?
Approach: Pipeline ❸. AccDecoder infers *inference frames* using the inference model and caches the results; then, it uses the MVs and cached results to infer non-inference frames (see ❸ in Fig. 4). MV indicates the offset between the target and reference blocks (§II-A). Here we use the object detection task as an example, which aims to identify objects (*i.e.*, their locations and classes) on each frame in videos. Fig. 6 reveals that the MVs between the last inference frame and the current frame can perfect match the movement of objects' Bboxes (*i.e.*, the results of object detection).

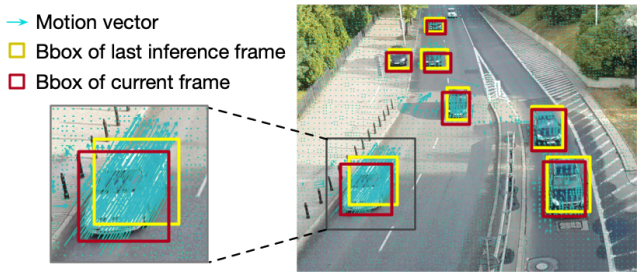


Fig. 6: Relation between MVs and Bboxes.

The *Reuse* module in Pipeline ❸ gets the result of the last inference frame (dotted line in Fig. 4), calculates the mean of all MVs that reside in each Bbox, and uses it to shift each Bbox to the current position. MV, as the block-level offset, is hard to express any semantic meaning (*e.g.*, object moving) [34]. Our preliminary study implies that the accuracy of *reuse* degrades significantly after the MV, which spans multiple reference frames (*e.g.*, over 7-10 frames in [7], [35], [36]).
Optimization. AccDecoder uses two techniques to improve the accuracy of the reuse of inference results. First, it filters the noisy MVs from static backgrounds and outliers. Empirically, AccDecoder filters the MV whose value is equal to zero or greater than the mean plus 0.8 times the standard deviation in the Bbox to which it belongs. Second, to cope with the change in Bbox size due to the object's movement, AccDecoder

expands the MV calculation region to each direction by one macroblock (16 pixels). Note that the reuse module is not necessary to tackle but only remit this erosion because the *scheduler* module in (*Q3*) effectively controls it by judiciously distributing inference frames.

Progress beyond the state of the art. Some prior work (*e.g.*, [37]–[39]) leverages lightweight MV-based methods to reuse analytics results and speed up inference. However, rather than calculating MV between continuous frames in the playback order like them, reuse in AccDecoder works in compressed-video space. That is, the reference blocks used to calculate the target block's MV can be distributed throughout the video (*the target block can even refer to future frames under the playback sequence, which is called backward reference*)[1], but the inference results should output in the playback order. To tackle this mismatch, we maintain a graph to map the coding order to the playback order and accumulate the MVs along the edges. Via statistical experiments on large-scale datasets, we find that the number of forwarding and backward references is less than 4 and 3 frames, respectively, so it is not time-consuming to search and calculate MVs in the graph.

Q3 — How to guarantee accuracy-latency balance?

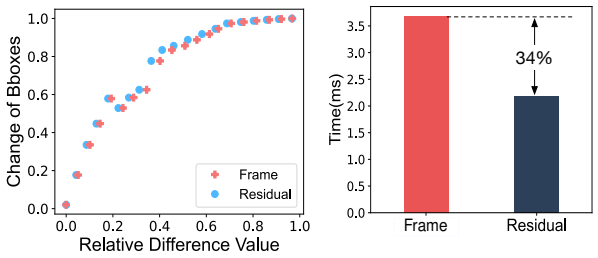
Approach: Scheduler. The key to the accuracy-latency trade-off is how to optimally assign decoding pipelines (*i.e.*, SR, inference, and reuse) to frames in fine spatial granularity. As analyzed in Fig. 3, different pipelines for frame decoding and analytics lead to different levels of accuracy and latency. For each frame in an SR class, selected anchor frames are enhanced by the SR model; after this, the scheduler feeds the up-scaling frame into the inference DNN for inference (*e.g.*, object detection). The frames in the inference class enjoy the benefits from the SR frames following the references in Fig. 5, and then are fed into the inference DNN model for inference. Their accuracy still increases due to the benefits transferred from the SR frames. Note that the transfer is quite fast (the time cost is the same as normal frame decoding) as it only includes additional bicubic interpolation on residual per frame compared to normal frame decoding. For those frames in the reuse class, *e.g.*, object detection, we get the Bbox of each object in the last (playback order) detected frame, calculate the mean of all MVs that reside in the Bbox, and use it to shift the previous position to the current position.

Model for pipeline selection. We formulate the adaptive pipeline selection problem to maximize the accuracy under the latency constraint. Given a video containing the set of frames F, one of the three pipelines is selected for each frame, which can be expressed as follows.

$$\begin{aligned} \max_{\mathbf{x}} \quad & \sum_{f \in F} Acc(x_f) \\ \text{s.t.} \quad & \sum_{f \in F} Latency(x_f) \leq \tau, \end{aligned} \tag{1}$$

[1]Modern video codecs aim at reducing video size; they only consider how to reduce the volume without caring whether the encoding obeys the playback order; thus *coding order* is very different from the playback order.

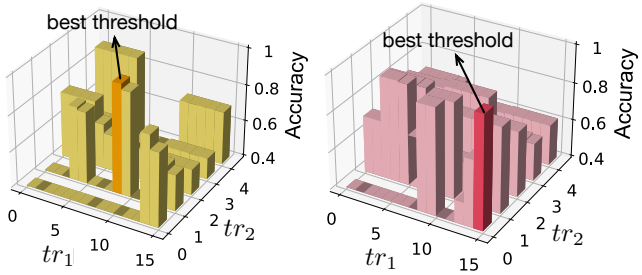
where $\mathbf{x} = \{x_1, ..., x_F\}$ is the selection set and $x_f \in \{1, 2, 3\}$ is for pipeline selection, Acc is the accuracy of a frame, $Latency$ is the latency of a frame given the selected pipeline, and τ is latency tolerant of frames F.



(a) Correlations (b) Time cost

Fig. 7: Frame vs. residual in correlations between the difference values and the changes of Bboxes, and time cost in feature extraction.

Finding the optimal pipeline selection is tricky due to the large searching space $3^{|F|}$. Inspired by Reducto [6] which adaptively filters frame via setting a threshold on frame differencing, we introduce two thresholds tr_1 and tr_2 on frame differencing to cluster frames into tree pipelines. When the frame difference is greater than tr_1, the frame will enter the pipeline ❶, and similarly, tr_2 is the threshold for the pipeline ❷. If the difference of a frame is greater than both of them, it will enter the pipeline❶. The constraint of real-time video analytics (*e.g.*, speed of analytics $\geq$30fps) restricts us from extracting the light-weighted features to categorize frames. Different from [6], we find out the Laplacian (*i.e.*, edge features) on the residual and the frames have a high correlation with the inference accuracy (see Fig. 7(a)). At the same time, executing the Laplacian operator on the residual can save 34% time than that on frame (see Fig. 7(b)). One intuitive reason is that information on residuals is sparse and de-redundant. It preserves differences among frames but is not too dense to process, thus providing a good opportunity to categorize frames efficiently.



(a) Video of crossroad (b) Video of highway

Fig. 8: Best thresholds for pipeline selection vary across videos.

Solution. Categorizing frames into three classes for pipelines is not trivial. Across various videos, their best threshold

combination differs (as shown in Fig. 8). Furthermore, the optimal thresholds for frame feature differences (*e.g.*, pixel and residual differences) among chunks in one video vary greatly. Fig. 9 plots the best thresholds on the videos in [35], which implies we should dynamically adjust the threshold for each chunk. Therefore, the scheduler needs to offer adaptive threshold settings.

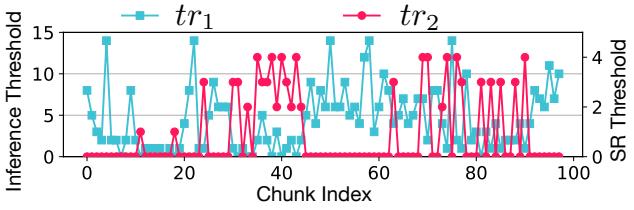


Fig. 9: Best thresholds vary across chunks (even adjacent).

To adaptively set the thresholds, we formulate this problem as a Markov decision process (MDP) where the scheduler makes the threshold setting decision in AccDecoder. The MDP is a discrete-time stochastic process, which can be defined by a quad-tuple $< S, A, R, P >$. In this tuple, S is the set of states, A is the set of actions, R is the set of rewards, and P is the probability of transition from state S to state S' based on action A. While processing frames, the scheduler's goal is to cluster them into three pipelines (*i.e.*, the action A) to maximize its expected long-run reward $\mathbb{E}[R_i]$. We define how the MDP is parameterized as follows.

- **State**: The state consists of two components: the content feature of the key frame and the differencing features including inter-frame differences and difference between the key frame and the last inference frame. First, the features of the key frame (*i.e.*, the first frame of the current chunk) are extracted through the 1x1x1000 fully connected FC-1000 layer of VGG16 [40]. Since the dimension of this feature is too large, we use principal component analysis (PCA) to reduce it to 128 dimensions. Next, we compute the inter-frame differences between every two frames of each chunk, which is the difference of edge features (*i.e.*, apply the Laplacian operator to the residual of each frame) as discussed in Section III. Considering that there is continuity between chunks, it is also necessary to add information about inter-chunk, that is, the difference of edge features between the key frame and the last inference frame in the previous chunk.
- **Action**: The action is to set two thresholds tr_1 and tr_2 for each chunk. The first threshold tr_1 is applied to these frames to select anchor frames for SR and transfer their quality to the others for scale-up. The second one tr_2 is to select inference frames that need to be analyzed by inference DNN. Then, the rest of the frames reuses the inference results by exploring frame reference. It is challenge to make accurate decisions for a large space of actions [41]. Therefore, we discretize the action space to reduce the action space, *i.e.*, $tr_1 \in \{0.05, 0.10, 0.15, ..., 0.75\}$ and $tr_2 \in \{0.5, 1.0, ..., 2.5\}$.
- **Reward**: Given that AccDecoder aims to maximize the inference accuracy within a tolerable latency. Therefore, the reward is designed to include two aspects, namely, the average accuracy of the chunk and the latency required to obtain the inference results for the chunk. Our goal is to achieve real-time inference, so the chunk needs to be analyzed before the arrival of the next chunk (*e.g.*, within 1s for each chunk); otherwise, there is a penalty for exceeding the specified time. The reward for each chunk t is defined as follows.

$$r_t = \frac{\alpha_1}{|F_t|} \sum_{f \in F_t} Acc_f(tr_{t,1}, tr_{t,2}) - \alpha_2 P_t(tr_{t,1}, tr_{t,2}), \quad (2)$$

where α_1 and α_2 are the weight factors to balance the preference for latency and accuracy. The value of α in reward can be adjusted based on different service preferences and requirements. P_t is a penalty function of chunk t for latency exceeding the tolerance τ.

$$P_t(tr_{t,1}, tr_{t,2}) = \begin{cases} 1 & \text{if } Latency(tr_{t,1}, tr_{t,2}) > \tau, \\ 0 & \text{others.} \end{cases} \quad (3)$$

The process of DRL in frame selection is shown below. At each chunk t, the agent observes the current state s_t and gives an action a_t according to its policy. Then, the environment returns reward r_t as feedback, and moves to the next state s_{t+1} according to the transition probability $P(s_{t+1}|s_t, a_t)$. The goal to find an optimal policy can thus be formulated as the mathematical problem of maximizing the expectation of cumulative discounted return $R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, where $\gamma \in [0, 1]$ is a discount factor for future rewards to dampen the effect of future rewards on the action; r_k is the reward of each step, and T is the number of chunks in the video.

Computational complexity. The searching space of the optimal pipeline selection for a chunk is $\mathcal{O}(3^k)$, where k is the number of frames in the chunk. The searching space of AccDecoder's scheduler is $\mathcal{O}(|a|)$, where $|a|$ is the number of possible actions. As we discretize the action space, therefore, the complexity of AccDecoder is much less than that of optimal pipeline selection.

IV. Implementation

We implement AccDecoder as an intelligent decoder in both simulation and prototype. We first introduce the system settings of the experiment and the setup of the dataset and the baselines. Next, we introduce the training setup of AccDecoder on neural networks of DRL, SR, and inference.

A. System Settings, Dataset, and Baselines

System settings. The server component runs on an Ubuntu 18.04 instance with Intel(R) Xeon(R) Gold 6226R CPU at 2.90GHz and 1 NVIDIA GeForce RTX 3070 GPU. AccDecoder is built on H.264 codec [42], JM 19.0 version open source code [43], with 2470 LoC changes. In practice, there are several video codecs besides H.264, but they have a high degree of similarity. For example, they share the same abstracts (*i.e.*, reference index, MV, and residual), which are essential

information to transfer neural super-resolution outputs to non-anchor frames. Their difference is in low-level compression algorithms (*e.g.*, the number of reference frames and the size of blocks). Thus, while we only use H.264 to validate AccDecoder's design, we believe the design is generic enough to accommodate different codecs.

Dataset. Our video dataset mainly contains public data streams from real-time surveillance cameras deployed worldwide. We collected video clips of different scenes and times from different camera data sources. Thus, video datasets with different properties (*e.g.*, time, illumination, vehicle and pedestrian density, road type, and direction) are obtained. All of our videos are available by searching on YouTube [35], [44], [45] and Yoda [7], [36], [46]. The videos are in 30fps, of which each chunk includes 30 frames (*i.e.*, $k = 30$). In particular, VisDrone dataset [47] is used to train SR model. We use CityScapes [48] as dataset and results of ERFNet [49] to calculate inference loss for training.

Baselines. We compare AccDecoder's performance with the following four baselines: (1) Glimpse [24] filters frames by comparing pixel-level frame differences against a static threshold. (2) AWStream [10] adapts encoding parameters (*i.e.*, quantization parameter (QP), resolution, and frame rate) of the underlying codec to cope with different available bandwidth; (3) DDS [5] applies different quality encodings to different regions via region proposal network (RPN) [3], which can offer trade-off accuracy and latency. (4) Reducto [6] filters unnecessary frames in its encoder via a dynamic setting threshold given by the server to save bandwidth in uploading.

B. DNN Implementation

DRL settings. In the experiments, we set $\alpha_1 = \alpha_2 = 0.5$ and $\tau = 1s$ for each chunk according to our practical experience and requirements of services. We use an Adam optimizer with a learning rate of 0.0001. The discount factor for reward $gamma$ is 0.99. We use a two-layer MLP with 128 units to implement the policy networks in DRL. The neural networks use ReLU as activation functions. The capacity of the replay buffer is 10^5, and we take a minibatch of 256 to update the network parameters.

SR model. For object detection, we train the detection-driven SR model based on EDSR [50] following the analytics aware loss function (*i.e.*, a weighted addition of visual quality loss and object detection inference loss) in [15]. We use VisDrone dataset [47] as the training set to train our model; use testing set from VisDrone, video set from DDS [5] and Reducto [6] to evaluate its performance. The visual quality loss comes from the pair of original and down sampling frames; the object detection inference loss comes from the results of YOLOv4 [1] detecting on reconstructed frames (from the down sampling frames) and the labels. The initial weights of EDSR and YOLOv4 are provided by authoritative implementations [1], [51]. The weights of EDSR are updated during our training, but the one of YOLOv4 keeps static.

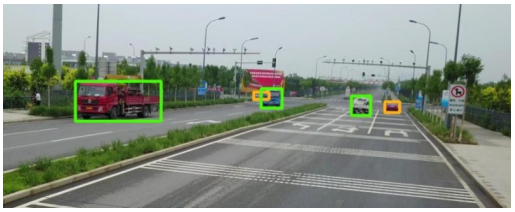
Inference DNN settings. We mainly evaluate AccDecoder's performance on object detection. Here, we list the different DNNs used by AccDecoder for object detection. We choose two object detection models with different architectures including Faster R-CNN [3], YoLov5 [52]. We pre-trained both models using the COCO dataset [53].

V. Evaluation

To demonstrate AccDecoder delivers significant quality improvement, we compare AccDecoder with baselines in terms of inference accuracy and latency.

A. Overall performance of AccDecoder

Video quality improvement. We illustrate the performance of SR via Fig. 10, which shows the visual object detection results. The results vary in the original 1080p images with ground truth labels, inference results of low resolution (in 270p), and inference results of up-scaled images by SR from 270p. From the results, we can see SR delivers more detailed information to the DNN inference model, thus bounding more small objects and improving accuracy in object detection.



(a) Ground truth (1080p)

(b) Low resolution (270p)

(c) Super resolution

Fig. 10: SR can improve inference accuracy by enhancing resolution of frames in videos.

Accuracy and latency. We demonstrate that AccDecoder can effectively improve accuracy-latency trade-off via adaptive pipeline assignment. Fig. 11 shows the per-chunk of pipeline assignment, accuracy (*i.e.*, f1-score as a measure of accuracy), and speed of analytics, respectively. AccDecoder clusters frames into three types: 1) anchor frames (around 6%) are selected for pipeline ❶ with SR; 2) inference frames (11%) are analyzed by inference DNNs in pipeline ❶ and ❷, in which

5% frames are for pipeline ❷; 3) non-inference and non-anchor frames (around 89%) are selected for pipeline ❸. Furthermore, AccDecoder and Reducto can achieve a higher frame rate than the basic requirement (*i.e.*, 30fps), whereas other baselines offer a lower rate. AccDecoder can also achieve higher and more stable accuracy than the baselines. It is worth noting that from the 35th chunk, the density and velocity of vehicles increase significantly (*i.e.*, 3x and 2.7x, respectively) so that the inferred rate is adjusted to a higher level, which ultimately maintains a good accuracy and frame rate.

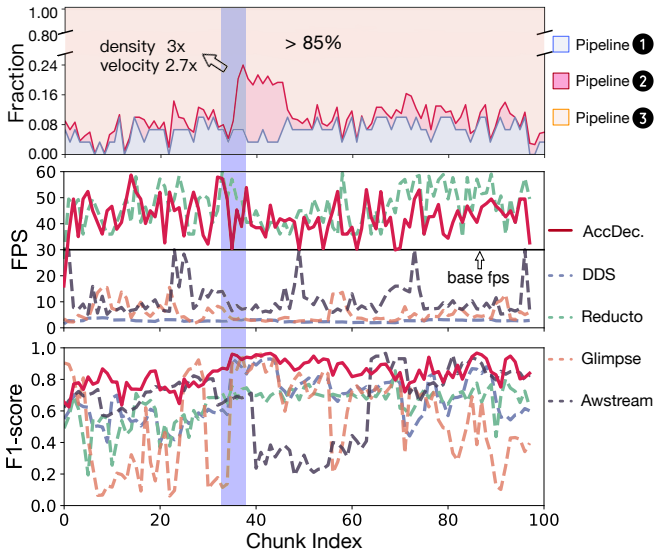
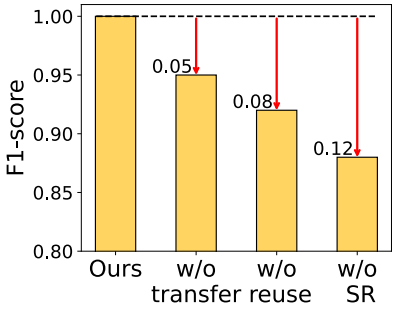


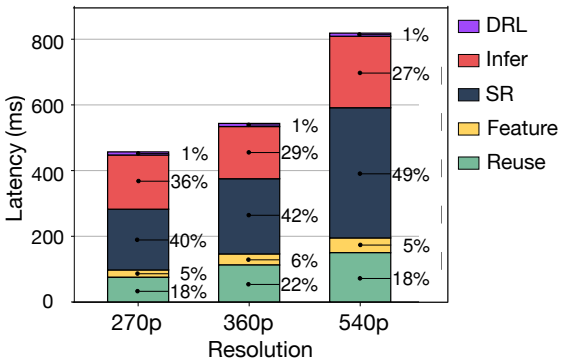
Fig. 11: Pipeline assignment ratio, analytics speed, and inference accuracy of chunks.

B. Component-wise Analysis

Performance breakdown. We break down the accuracy and the latency to show the impact of each pipeline illustrated in Fig. 12. Firstly, Fig. 12(a) breakdowns the accuracy in transfer of anchor frames, reuse of the inference frames, and SR. The impact of reuse and SR on accuracy is obvious, while the impact of transfer is relatively small. The main reason is that the transfer has less improvement on small objects due to blur caused by interpolation, which will be one of our future works. Secondly, Fig. 12(b) shows the latency breakdown in processing each chunk, where we use videos in 270p, 360p, and 540p, respectively. AccDecoder needs more time when processing and analyzing high-resolution videos, *i.e.*, the overall latency in ms. Specifically, SR (∼40%), inference (∼30%), and reuse (∼20%) take up most of the latency, while DRL-based scheduling and feature extraction only occupy around 5%, which can be omitted. SR is time-consuming, although only 6% of frames are assigned to pipeline ❶. Besides, inferring pipeline ❶ and ❷ for around 11% of frames also takes a latency that cannot be ignored. Although the time cost of reuse per frame is small, the time cost of reuse requires 20% latency in total due to more than 85% frames belonging to pipeline ❸.



(a) Accuracy breakdown



(b) Latency breakdown

Fig. 12: Performance breakdown on accuracy and latency.

AccDecoder schedulers. We evaluate the scheduler's performance in AccDecoder, including DRL-based schemes and KNN, as illustrated in Fig. 13. We choose three well-known DRL schemes for training: Asynchronous Advantage Actor-Critic (A3C) [54], Soft Actor-Critic (SAC) [55], and Proximal Policy Optimization (PPO) [56]. Through effective exploration, we find A3C can receive a satisfactory and stable cumulative reward, which was about 11.42% higher than the cumulative reward received by KNN and PPO. From the final cumulative reward value in the figure, the cumulative rewards of A3C and SAC are higher. Considering the advantages of A3C, we choose A3C for the training of the scheduler.

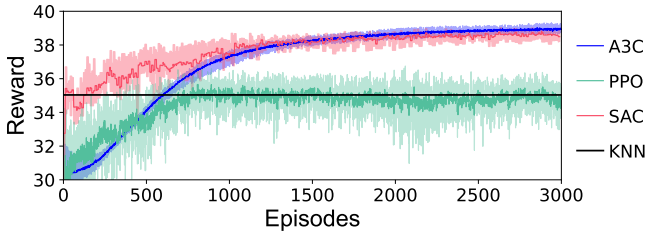
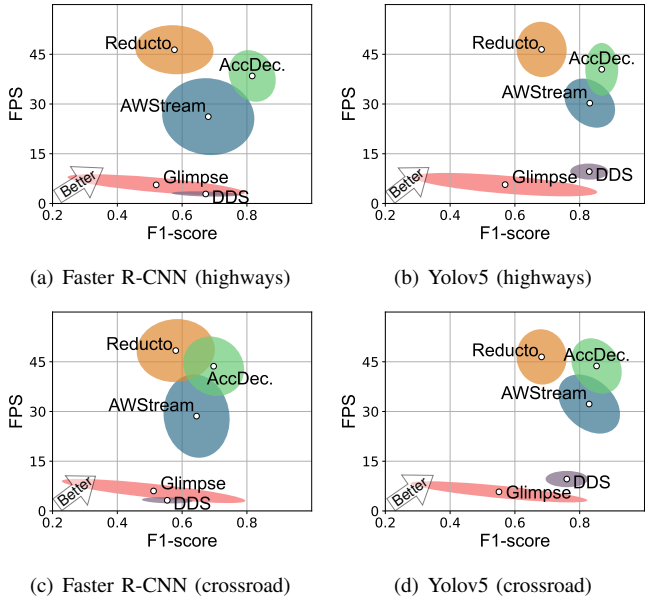


Fig. 13: Comparison of different schemes for the scheduler in AccDecoder. DRL is better than KNN, and A3C achieves better learning performance than the others.

C. AccDecoder vs. Existing VAPs

AccDecoder vs. baselines. We show the advantage of AccDecoder in accuracy and speed of analytics compared with the

baselines. Fig. 14 compares the performance distribution of AccDecoder with the baseline over different DNNs (YOLOv5 and Faster R-CNN with Resnet50) and various types of videos (*i.e.*, highways and crossroads). It can be seen that AccDecoder is better than the baseline in terms of speed and accuracy of analytics. In terms of speed, AccDecoder's analytics speed is around 35fps, which meets the penalty threshold we set for DRL training. In terms of accuracy, AccDecoder can keep relatively higher accuracy compared with the baselines.



(a) Faster R-CNN (highways)

(b) Yolov5 (highways)

(c) Faster R-CNN (crossroad)

(d) Yolov5 (crossroad)

Fig. 14: AccDecoder vs. baselines in terms of the speed of analytics and inference accuracy on various video datasets (in parentheses) and different DNN models (*i.e.*, Faster R-CNN and Yolov5). AccDecoder achieves 6-38% higher inference accuracy than the baselines and 20-80% lower latency than the baselines except for Reducto.

VI. Related Works

Video analytics pipeline. Many computer vision tasks are considered in VAPs, such as traffic control [57], surveillance and security [11]. We consider the following task as a running example — object detection. Object detection aims to identify objects of interest (*i.e.*, their locations and classes) in each frame in the video. Selecting this task has two major reasons: first, it plays a core role in the computer vision community because a wide range of high-level tasks (*e.g.*, autonomous driving) is built on it; second, we seek to keep consistent with prior video analytics work [5], [58]–[60] to allow a straightforward performance comparison.

Deep reinforcement learning. DRL is well suited to tackle problems requiring longer-term planning using high-dimensional observations, which is the case of dynamic pipeline selection. There is a variety of DRL-based algorithms. Value-based algorithms, *e.g.*, Deep Q-Networks (DQN) [61], use a deep neural network to learn the action-value function. However, they do not support continuous action space like the one in our problem. Policy-based algorithms, *e.g.*, Policy Gradient (PG) [62], explicitly build a representation of a policy. However, evaluating a policy without action-value estimation is typically inefficient and causes high variance. Actor-critic algorithms learn the value function (critic) in addition to the policy (actor) since knowing the value function can assist policy updates, for example, by reducing variance in policy gradients. Many existing approaches are based on actor-critic, for example, PPO [56], A3C [54], and SAC [55]. A3C, an asynchronous algorithm, can enable multiple worker agents to train in parallel, allowing faster training.

VII. Conclusion and Discussion

In this paper, we propose AccDecoder to eliminate the dependence of existing VAPs on video quality. AccDecoder is a new universal video stream decoder that uses a super-resolution deep neural network to enhance video quality for video analytics. To accelerate analytics, AccDecoder applies DRL for adaptive frame selection for quality enhancement or/and DNN-based inference. It is a new way to address the key challenge of accuracy-latency tradeoff in distributed VAPs. We show that AccDecoder can substantially improve state-of-the-art VAPs by speeding up analytics (3-7x) and achieving accuracy improvements (6-21%).

In future work, we plan to explore DRL for macroblock selection to offer finer-grained scheduling and joint adaptation encoding and decoding to further improve the accuracy and speed of analytics.

Acknowledgment

This work has been partly funded by EU H2020 COSAFE (Grant 824019) and Horizon CODECO projects (Grant 101092696), the Alexander von Humboldt Foundation, and the National Natural Science Foundation of China under Grant 61872178, 62272223, and Grant 61832005.

References

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," in *arXiv:2004.10934*, 2020.

[2] N. Ahn, B. Kang, and K.-A. Sohn, "Fast, accurate, and lightweight super-resolution with cascading residual network," in *Proc. of ECCV*, 2018, pp. 252–268.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[4] H. Wang, X. Jiang, H. Ren, Y. Hu, and S. Bai, "Swiftnet: Real-time video object segmentation," in *Proc. of CVPR*, 2021.

[5] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proc. of SIGCOMM*, 2020, pp. 557–570.

[6] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proc. of ACM SIGCOMM*, 2020, pp. 359–376.

[7] Yoda, "crossroad video," https://yoda.cs.uchicago.edu/videos/crossroad.mp4.

[8] New York City Department of Transportation, "Real time traffic information," https://webcams.nyctmc.org/, Mar. 2020.

[9] P. Wei, H. Shi, J. Yang, J. Qian, Y. Ji, and X. Jiang, "City-scale vehicle tracking and traffic flow estimation using low frame-rate traffic cameras," in *Proc. of UbiComp/ISWC*, 2019, pp. 602–610.

[10] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics." in *Proc. of SIGCOMM*, 2018, pp. 236–252.

[11] J. Yi, S. Choi, and Y. Lee, "Eagleeye: Wearable camera-based person identification in crowded urban spaces," in *Proc. of MobiCom*, 2020.

[12] J. Yi, S. Kim, J. Kim, and S. Choi, "Supremo: Cloud-assisted low-latency super-resolution in mobile devices," *IEEE Transactions on Mobile Computing*, 2020.

[13] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proc. of CVPR*, 2017.

[14] Y. Chen, Y. Tai, X. Liu, C. Shen, and J. Yang, "Fsrnet: End-to-end learning face super-resolution with facial priors," in *Proc. of CVPR*, 2018.

[15] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," in *Proc. of IEEE INFOCOM*, 2022, pp. 1–10.

[16] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv:1905.05055*, 2019.

[17] A. Ghodrati, B. E. Bejnordi, and A. Habibian, "Frameexit: Conditional early exiting for efficient video recognition," in *Proc. of CVPR*, 2021.

[18] A. Habibian, D. Abati, T. S. Cohen, and B. E. Bejnordi, "Skip-convolutions for efficient video processing," in *Proc. of CVPR*, 2021.

[19] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[20] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *IEEE Signal Processing Magazine*, vol. 34, no. 6, 2017.

[21] T. Yuan, H.-M. Chung, J. Yuan, and X. Fu, "DACOM: Learning delay-aware communication for multi-agent reinforcement learning," in *Proc. of AAAI*, 2023.

[22] T. Yuan, C. E. Rothenberg, K. Obraczka, C. Barakat, and T. Turletti, "Harnessing uavs for fair 5g bandwidth allocation in vehicular communication via deep reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4063–4074, 2021.

[23] T. Yuan, W. da Rocha Neto, C. E. Rothenberg, K. Obraczka, C. Barakat, and T. Turletti, "Dynamic controller assignment in software defined internet of vehicles through multi-agent deep reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 585–596, 2020.

[24] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. of SenSys*, 2015, pp. 155–168.

[25] Boulder AI, "Dncam ai camera," https://groupgets.com/campaigns/429-dnncam-ai-camera.

[26] NVIDIA, "Jetson tx2," https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-tx2.

[27] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[28] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das, "Streaming 360-degree videos using super-resolution," in *Proc. of IEEE INFOCOM*, 2020, pp. 1977–1986.

[29] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, "Neural-enhanced live streaming: Improving live video ingest via online learning," in *Proc. of ACM SIGCOMM*, 2020, pp. 107–125.

[30] H. Yeo, C. J. Chong, Y. Jung, J. Ye, and D. Han, "Nemo: enabling neural-enhanced video streaming on commodity mobile devices," in *Proc. of MobiCom*, 2020, pp. 1–14.

[31] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware internet video delivery," in *Proc. of OSDI 18*, 2018, pp. 645–661.

[32] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the edge-cloud barrier for real-time advanced vision analytics," in *Proc. of HotCloud*, 2019.

[33] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proc. of MobiCom*, 2015, pp. 426–438.

[34] H. Yeo, S. Do, and D. Han, "How will deep learning change internet video delivery?" in *Proc. of ACM HotNets*, 2017, pp. 57–64.

[35] Youtube, "Gebhardt insurance traffic cam round trip bike shop," https://www.youtube.com/watch?v=_XBMMTQVj68.

[36] Yoda, "highway video," https://yoda.cs.uchicago.edu/videos/highway.mp4.

[37] J. Zhang, D. Zhang, X. Xu, F. Jia, Y. Liu, X. Liu, J. Ren, and Y. Zhang, "Mobipose: Real-time multi-person pose estimation on mobile devices," in *Proc. of SenSys*, 2020, pp. 136–149.

[38] R. Xu, C.-l. Zhang, P. Wang, J. Lee, S. Mitra, S. Chaterji, Y. Li, and S. Bagchi, "Approxdet: content and contention-aware approximate object detection for mobiles," in *Proc. of SenSys*, 2020, pp. 449–462.

[39] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. of MobiCom*, 2019, pp. 1–16.

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. of ICLR*, 2015, pp. 602–610.

[41] S. Yan, X. Wang, X. Zheng, Y. Xia, D. Liu, and W. Deng, "Acc: Automatic ecn tuning for high-speed datacenter networks," in *Proc. of ACM SIGCOMM*, 2021, pp. 384–397.

[42] Google, "libvpx official github repository," https://github.com/webmproject/ libvpx/.

[43] Open Source, "Jm 19.0," https://iphome.hhi.de/suehring/.

[44] Youtube, "Jackson hole wyoming usa town square live cam," https://www.youtube.com/watch?v=1EiC9bvVGnk.

[45] Youtube, "City of auburn toomer's corner webcam," https://www.youtube.com/watch?v=hMYIc5ZPJL4.

[46] Yoda, "motorway video," https://yoda.cs.uchicago.edu/videos/motorway.mp4.

[47] H. Fan, D. Du, L. Wen, P. Zhu, Q. Hu, H. Ling, M. Shah, J. Pan, A. Schumann, B. Dong *et al.*, "Visdrone-mot2020: The vision meets drone multiple object tracking challenge results," in *Proc. of ECCV*, 2020, pp. 713–727.

[48] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of CVPR*, 2016.

[49] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2017.

[50] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. of CVPR*, 2017.

[51] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. of CVPR*, 2017.

[52] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger." in *Proc. of CVPR*, 2017.

[53] C. dataset, "highway video," https://cocodataset.org/.

[54] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. of ICML*, 2016.

[55] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. of ICML*, 2018, pp. 1861–1870.

[56] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[57] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *IEEE Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[58] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proc. of ACM SIGCOMM*, 2018, pp. 253–266.

[59] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, V. Bahl, and J. Gonzalez, "Spatula: Efficient cross-camera video analytics on large camera networks," in *ACM/IEEE Symposium on Edge Computing (SEC 2020)*, 2020.

[60] Y. Yuan, W. Wang, Y. Wang, S. S. Adhatarao, B. Ren, K. Zheng, and X. Fu, "Vsim: Improving qoe fairness for video streaming in mobile environments," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1309–1318.

[61] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[62] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour *et al.*, "Policy gradient methods for reinforcement learning with function approximation." in *Proc. of NIPS*, vol. 99, 1999, pp. 1057–1063.