Spliceosome: On-Camera Video Thinning and Tuning for Timely and Accurate Analytics

Ning Chen, Sheng Zhang, *Senior Member, IEEE*, Jie Wu, *Fellow, IEEE*, He Huang, *Senior Member, IEEE*, and Sanglu Lu, *Member, IEEE*

***Abstract*— Running deep neural networks (DNNs) on large-scale videos from widely distributed cameras presents two significant challenges. Firstly, video quality for analytical purposes is severely impacted by the camera deployment environment, which is termed *Pixel Recession* in this paper. Secondly, low-latency video streaming from the source camera to edge servers is greatly hindered by the rapid expansion of video traffic. Despite numerous efforts such as enhancing the video structure, uneven encoding, and filtering frames captured on camera, these methods have proven insufficient to address the challenges at hand. We propose Spliceosome, a novel video analytics system that effectively overcomes the pixel recession and streaming bottlenecks. In brief, Spliceosome 1) recovers from *pixel recession* by adaptive video knobs (i.e., *brightness* and *contrast*) *tuning* in ARP (*anchor region proposal*) granularity, and 2) lowers the transmission volume by video *thinning*, which uses only single-channel information for video encoding. We implemented Spliceosome using only commercial off-the-shelf hardware. Our experimental results demonstrate that Spliceosome outperforms other alternative designs by 4.71-14.47%, 40.94-58.71%, and 14.28% in detection accuracy, end-to-end delay, and efficiency of DNNs inference, respectively.**

***Index Terms*— Video analytics, video quality for analytical purpose, anchor region proposal, single-channel codec.**

I. Introduction

EMPOWERED by the emerging computer vision, video analytics applications ran on mobile (edge) devices are widely used in business (industrial logistics, home assistance, retail, etc) and public (urban planning, traffic management, etc) fields [1], [2], [3], [4], [5], [6]. In real-world deployment, most video applications apply the developed deep neural networks (DNNs) to analyze the massive videos from widely distributed edge video sensors (cameras), leading to an explosive growth of video data [7], [8] for analytical purposes rather than human users entertainment [8], [9], [10] by watching.

In general to acquire the analytics result of a captured video chunk, it undergoes three key procedures [13], [41], including on-camera processing (e.g., video compressing and encoding), network bitrate transmission, and on-server DNNs inference. An ideal video analytics system should meet three requirements that are vital to analytics applications: 1) low resource (i.e., storage and computation) overhead on mobile camera, 2) low end-to-end delay resulting from video encoding and data transmission from camera to centralized edge server, and 3) high inference accuracy through DNNs inference. It is, however, non-trivial to satisfy them simultaneously, since it often requires efficient management of many factors, such as the video quality for analytical purpose, network status for end-to-end data transmission, and balance between analytical performance and overhead. In particularly, we summarize two critical challenges below:

First of all, the deployment environment for mobile cameras in the real world is inherently dynamic and uncertain, which results in that the quality of video captured for analytical purposes (as opposed to human-perceived visual quality) varies over time, thus probably giving rise to poor analytics performance. We take *object detection* application as an example to illustrate this problem (called *pixel recession* in Section II-B). For these videos captured in a blurry environment due to factors such as day-night alternating or camera deployment (e.g., commonly under a huge tree without light), it probably fails to accurately detect the target objects. In this dilemma, optimizing the compression and encoding procedures [14], [37] or improving human-perceived visual quality [15], [61] is ineffective to improve the accuracy. One could employ a specific method to enhance the blurry video. However, performing pixel-level video enhancement is challenging due to the lack of a solid theoretical basis.

Second, it is essential to address the conflict between streaming a large amount of video and providing time-varying network bandwidth. While video compression techniques partially reduce data volume, the expanding scope and scale of camera deployments in cities and organizations generate video traffic for analytical uses, which occupies a substantial portion [7], [8], resulting in significant end-to-end streaming delays. This challenge is exacerbated when there is increased competition for limited bandwidth among multiple mobile cameras in unreliable network connections. Additionally, cameras with limited-capacity batteries make it unfeasible to continuously transmit high-volume video data.

Unfortunately, the current research is insufficient in addressing the aforementioned challenges. Mainstream video enhancement methods primarily concentrate on enhancing structural quality (measured by SSIM or PSNR [16]) through encoding optimization [14], [37] and super-resolution

Received 15 July 2024; revised 12 November 2024; accepted 2 January 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor L. Fu. This work was supported in part by the Nanjing Key S&T Special Projects under Grant 202309006 and in part by NSFC under Grant 62332013. *(Corresponding author: Sheng Zhang.)*

Ning Chen and He Huang are with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China (e-mail: ningc@smail.nju.edu.cn; huangh@suda.edu.cn).

Sheng Zhang and Sanglu Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China (e-mail: sheng@nju.edu.cn; sanglu@nju.edu.cn).

Jie Wu is with the Center for Networked Computing, Temple University, Philadelphia, PA 19122 USA (e-mail: jiewu@temple.edu).

Digital Object Identifier 10.1109/TON.2025.3526218

techniques [15], [18], [19], [20], [61]. However, none of the previous studies have addressed the critical concern of improving the quality of analytics-based video by addressing *pixel recession*. Although various research efforts have attempted to reduce the volume by utilizing camera-side heuristics to filter out redundant pixels, the camera with limited resources cannot efficiently identify the superfluous pixels. Several subsequent proposals with low camera-side overhead and high analytical accuracy have emerged [13], [17], [37]. These proposals require sending content to a centralized server for DNN inference and receiving the results, which optimize the camera's RoI (Region of Interest) encoding operation but result in increased end-to-end delay.

To overcome the above two challenges, this paper introduces **Spliceosome**,[1] a novel video analytics system that satisfies the three requirements previously mentioned. The development of Spliceosome was motivated by two observations (discussed in Section II): 1) the poor analytical accuracy caused by *pixel recession* is primarily due to inadequate video *brightness* and *contrast*; therefore, adaptively adjusting these parameters shows potential to improve performance, and 2) instead of feeding information from all channels into DNNs for inference, employing pixels from a single channel can attain high accuracy, significantly reducing end-to-end transmission volume and server-side inference computation. Spliceosome consists of three essential components, namely the *Anchor Region Proposal Builder*, the *F1_AccGrad Scheduler*, and the *Single-Channel Codec*.

Given the limited resources available on mobile cameras, Spliceosome proposes performing region-level optimization rather than computationally expensive frame-level operations, which aligns with the fact that only partial regions of the frame contain the target objects. The *Anchor Region Proposal Builder* is responsible for producing these target regions. Unlike other approaches [13], [37] that use server-driven feedback to identify regions, Spliceosome leverages the local *motion vector* that indicates the offsets between frames. It first applies a *three-step clustering algorithm* to the extracted motion vectors of the current frame to generate the initial *anchor region proposals* (ARPs). To address the potential issue of incomplete target object coverage, it then designs a *padding* mechanism to expand the initial ARPs. Through these two stages, the final ARPs are generated.

To address the issue of *pixel recession*, Spliceosome proposes an *F1_AccGrad Scheduler* to automatically adjust the *brightness* and *contrast* for each ARP. Preliminary results show that when fixing *brightness*, the detection accuracy increases monotonically with an increase in *contrast*, but the marginal improvement diminishes. Conversely, when fixing *contrast*, accuracy significantly improves at first but gradually decreases if the *brightness* exceeds a specific threshold. The concavity of these two relationships allows for the application of *alternate gradient ascent* to obtain the global optimum [26]. As there is no closed-form equation for the accuracy and its derivative, Spliceosome uses a numerical method to derive the approximate gradient.

Having finished the above procedures, Spliceosome uses *Single-Channel Encoder* to encode the video. It is based on the open-source x265 HEVC Encoder [27], with three modifications. Firstly, for the *YUV* format (i.e., *YCbCr*), it sets *Cb* and *Cr* to 0 directly, which significantly accelerates video encoding. Secondly, in the stage of *chroma subsampling*, it only maintains luma information (i.e., *Y*), further reducing the video volume. Finally, it introduces an uneven *QP* (Quantization Parameter) value assignment for ARPs and non-ARPs, which matches the variation of network bandwidth.

By seamlessly integrating the above three modules, Spliceosome not only significantly reduces end-to-end delay and server-side overhead but also effectively mitigates the accuracy degradation caused by *pixel recession*. Moreover, with no strict hardware requirements, Spliceosome can be ported to any operating system. We implemented Spliceosome using commercial-off-the-shelf hardware and used two types of videos, BridgeCam LIVE [21] and Rotary Traffic Live [22] from YouTube [23], to evaluate Spliceosome's performance. The experimental results demonstrate that Spliceosome enhances detection accuracy by 4.71-14.47%, reduces the average response time by 40.94-58.71%, and speeds up inference by 14.28% compared to state-of-the-art designs. In summary, our work makes the following key contributions:

- *On-camera ARP generation.* By utilizing camera-side resources to run lightweight clustering and padding on the extracted motion vector, Spliceosome is able to accurately build the potential ARPs.
- *Analytics-oriented pixel-level video enhancement.* By adaptively tuning video knobs (i.e., *brightness* and *contrast*), Spliceosome significantly improves the detection accuracy though facing *pixel recession*.
- *Efficient single-channel encoding.* By discarding the chroma information in two channels, Spliceosome largely accelerates the camera-side encoding and server-side inference.

II. Motivations and Challenges

We start with the performance metrics for mainstream video analytics applications, then we point out several potential limitations of prior system designs, and finally we present our critical ideas to improve them.

A. Video Analytics Evaluation Metrics

Target Applications. In this paper, we focus on those applications that use state-of-the-art convolutional neural networks (CNNs) to run a variety of complicated vision tasks. Examples include object detection, face recognition and semantic segmentation. Typically, these applications involve feeding a video frame into a well-trained CNN network, which produces detailed results such as bounding boxes with corresponding classified object types or facial key points. These fundamental applications are widely used in real-world scenarios such as vehicle collision detection, security authentication systems, and human-computer interaction.

[1]The spliceosome is responsible for removing introns, which are non-coding regions of the pre-mRNA transcript, and joining together the exons, which are the coding regions of the pre-mRNA transcript, to produce a mature mRNA molecule that can be translated into protein.

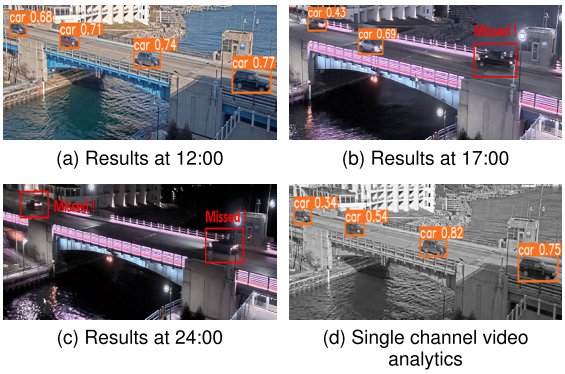
Performance Metrics. An ideal end-to-end video analytics system should prioritize two key metrics, i.e., analytical accuracy and response time.

- *Analytical accuracy:* For each frame that is streamed to a server, we define its accuracy as the similarity between the DNN output of this frame and the output of its corresponding highest-quality (or optimal-configuration) frame. Rather than human-annotated labels, we use the DNN output on the highest-quality frame as the "ground truth". This method is consistent with recent studies [13], [40], [49], [52]. We evaluate the accuracy by *F1 score* (i.e., the harmonic mean of recall and precision for the detected objects' locations and class labels) in object detection application and by *IoU* (i.e., the intersection over union to measure the degree of overlap between the predicted segmentation map and the ground truth segmentation map) in semantic segmentation.
- *Response delay:* it consists of two key components. The first is the end-to-end delay, which results from on-camera processing such as video encoding and streaming from the camera to an edge server. The end-to-end delay largely depends on the video compression ratio and network connection status. The second component is the inference delay, which is caused by DNN forward propagation and is subject to the model's complexity and input frame size. If the system enables these components to be pipelined and executed in parallel, we take the *average response delay* as the measurement metric.

B. Limitations of Previous Work

Case for Improving Detection Accuracy. Current studies that aim to optimize analytical performance can be broadly categorized into two approaches. The first approach, an encoding-driven method [14], [37], assigns a higher bitrate to encode the potential regions that contain the target object. The second approach, a neural super-resolution-based method [18], [19], [20], upscales the low-resolution video, which significantly improves detection accuracy, especially for small objects. However, the former lacks effective methods to identify the target regions, while the latter causes extra inference overhead due to compute-intensive neural super-resolution, which largely affects real-time video analytics. Although these designs focus on enhancing the video's structural quality measured by SSIM or PSNR [16], they prove inefficient when encountering a phenomenon known as *pixel recession*, which occurs when the pixels that make up a target object gradually blend with the background pixels due to changing light conditions or camera deployment, such as when the camera is placed under a large tree without light. This makes the object unrecognizable to support accurate analytics, hence we call the *Pixels* are in *Recession*. In this case, preserving high structural quality is probably ineffective.

Specifically, we take Fig. 1 as an example to further understand *pixel recession*. We first record three video frames of BridgeCam LIVE [21] from YouTube [23] at three time point (i.e., 12:00 pm, 17:00 pm and 24:00 pm), and then we run the state-of-the-art detection algorithm model yolov5 [29] on them. It's clear that in Fig. 1a, all the objects are accurately



(a) Results at 12:00 (b) Results at 17:00

(c) Results at 24:00 (d) Single channel video analytics

Fig. 1. Video analytics at different times of a day.

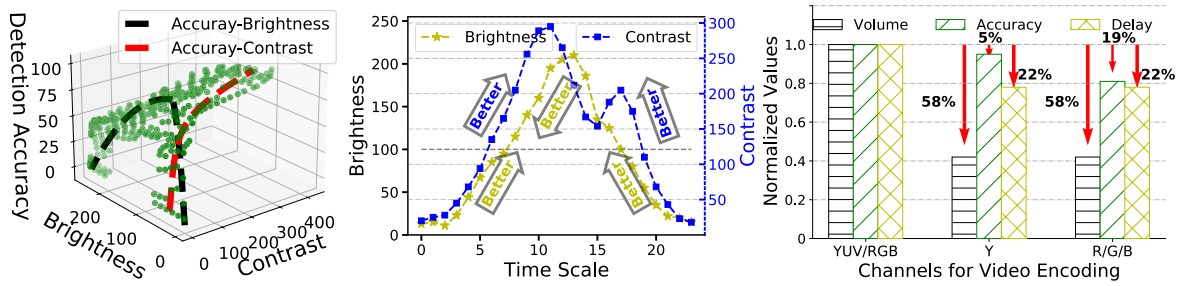
detected, but in Fig. 1b at 17:00 pm, it misses a car and outputs classes with low confidences. What's worse in Fig. 1c, it unexpectedly misses all the target objects. Investigate its reasons, the car pixels are extremely similar to pixels of background (e.g., bridge). Though in Fig. 1b and Fig. 1c several lamps are deployed for illumination, it is still hard to distinguish the cars from the dark. For another instance in real scenarios, criminals prefer drive the car in a dark environment to avoid tracking, and animals disguise their appearance as similar as surroundings for hunting or escaping.

Case for Optimizing Average Processing Time. To reduce the end-to-end delay by decreasing the streaming volume, a widely used approach is to implement camera-side logic [11], [12] to filter out frames that are irrelevant or redundant to the vision tasks. This method is effective when the video content is relatively stationary, such as in wildlife camera feeds, where the background is static and animals are rare, resulting in the filtering out of massive frames. However, for frames that cannot be discarded, the camera encodes the entire frames with equal quality. This approach is suboptimal since the objects of interest are sparsely distributed in each frame. Some works [24], [41] use local heuristics to lower the quality of the background, and send quality-enhanced object-related areas (e.g., region-of-interest encoding [37]) to the server for DNN inference. Nevertheless, these on-camera heuristics are limited by local compute resources, leading to significant identification time and false negatives (i.e., object-related regions being regarded as background and sent in low quality, causing the server to miss the target objects).

To address this issue, a follow-up approach [13], [37] streams low-resolution videos to a powerful server, which leverages its sufficient resources to output and return the initial results. Based on these results, the camera resends the regions that are not well-detected. Although this mechanism effectively lowers the transmission volume while maintaining high accuracy, it may suffer from an expensive response delay. For example, in the work by DDS [13], when faced with a terrible network status, the camera takes at least two network round-trip times (RRTs) to receive server-driven feedback before executing video encoding, resulting in high response time on each frame.

C. Potential Room for Optimization

Recovery from *Pixel Recession*. As previously stated, the analytical performance of vision application is significantly



(a) Accuracy under diverse knobs (b) Knob counting of a day (c) Performance under diverse channels

Fig. 2. Motivations. (a) the detection accuracy under different knobs, (b) the statistical knobs at different time points, and (3) the achieved performances under diverse encoding schemes.

influenced by the deployment of the camera and the variations in the surroundings. In order to identify the underlying reason for this phenomenon, we have introduced two critical knobs, namely video *brightness* and *contrast*.[2] To investigate the correlation between these two parameters and the detection accuracy (i.e., *F1 score*), we have extracted 350 video frames from BridgeCam LIVE [21] at different time intervals and conducted the analysis using the yolov5 [29] model with the NVIDIA RTX 2080 Ti GPU. As depicted in Fig. 2a, the detection accuracy improves significantly with an increase in video brightness initially, but gradually decreases beyond a specific threshold (e.g., around 100). For videos with a low initial contrast, even a slight increase in contrast leads to remarkable improvement in detection *F1 score*, but the benefit eventually becomes marginal. Therefore, a video is considered to be in a *pixel recession* state when its *brightness* and *contrast* parameters are not adequate to support precise video analytics.

Hence to resolve *pixel recession*, an intuitive approach is to continuously adjust the video brightness and contrast until achieving optimal detection accuracy. As Fig. 2b shows, we count the *brightness* and *contrast* for each hour. Based on the observation in Fig. 2a, it is suggested to decrease brightness from 8:00 to 16:00 and increase it at other times. Similarly, we propose to increase contrast significantly before 10:00 and after 17:00. By following this approach, we can ensure that the video analytics system operates optimally, resulting in accurate and reliable detections.

Optimization of Video Encoding. As described before, the server-side DNN-driven method for conducting quality-uneven encoding is often hindered by unreliable network connections. Recent years have witnessed the rapid growth in compute capacity of mobile device (e.g., NVIDIA JetSon Nano, TX2, and Xavier NX), which allows for relatively complex algorithms to be run locally on the device. Despite this, running camera-side DNN-based algorithms to identify regions of interest remains a challenge, particularly when the camera is powered by a battery with limited capacity. Fortunately, the *Motion Vector* (MV) commonly used in existing video codecs (e.g., H.264 and H.265) to indicate pixel offsets among frames provides a promising method for locating *Anchor Region Proposals* (ARPs) that contain moving objects.

[2]For a video frame f with pixel size $M \times N$, we define its brightness $B_f = \sum_i^M \sum_j^N \frac{p_{i,j}}{M \times N}$, and contrast $C_f = \sum_i^M \sum_j^N \sum_k^{\delta_{i,j}} |p_{i,j} - k|^2/(S_f)$, where $p_{i,j}$ indicates the pixel value of point (i, j), $\delta_{i,j}$ represents the neighbor points set of (i, j) and S_f is the number of adjacent pixel pair ($4M \times N$ by default in this paper).

In a typical video analytics system, three-channel videos (i.e., *RGB* or *YUV*) go through multiple stages including video splitting, intra prediction, inter prediction, DCT transformation, quantization, entropy encoding, and bitstream generation to reach the remote server for DNN inference, which incurs significant end-to-end delays in three-channel granularity. Can we reduce this delay by using only one channel of information? To investigate the feasibility, we implemented some optimizations (detailed in Section VI) based on the open-source x265 HEVC Encoder [27], and re-encoded live videos [21] using one channel and three channels separately. We then fed these two types of videos into DNNs for inference. Note that the HEVC Encoder first transforms the video format from *RGB* to *YUV* based on the ITU-R standard [28], and we use the information of the *Y* channel (i.e., luma information) or any one channel from *R/G/B* for single-channel encoding. As Fig. 1d shows, it achieves the similar detection result to that in Fig. 1a, which uses three-channel information for encoding. Fig. 1c illustrates the average performance in three metrics. Single-channel encoding outperforms the traditional method by 58% and 22% in terms of data volume and inference delay, respectively. Moreover, compared to a 19% accuracy drop when using any one channel of *RGB*, the *Y*-channel-based method incurs only a 5% accuracy loss, primarily because *Y* is derived from *RGB* and contains the key information of *RGB*.

D. Design Challenges

There are several key challenges to fulfill these designs:

- *Online brightness and contrast adjustment.* Given the significant variation of video knobs in different camera surroundings, continuous adjustments are crucial to adapt to such changes. However, the absence of a solid theoretical foundation makes it difficult to conduct such adjustments. Furthermore, calculating frame-level contrast on the camera-side is expensive, which can affect the real-time capabilities of video analytics.
- *MV-based ARP prediction.* When the object pixels closely resemble the surrounding pixels (i.e., *pixel recession*) or when the object is situated far from the camera, the extracted MVs may not sufficiently cover the entire object.
- *Single-channel video codec.* Deciding which channel information to use for video encoding is a non-trivial task. Since most video codecs are oriented towards three-channel video, it is essential to design an efficient

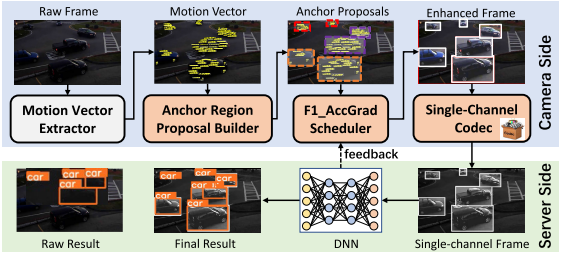


Fig. 3. Spliceosome system architecture. We present the system using an object detection example.

mechanism for transplanting the single-channel encoding approach into existing codecs.

III. System Overview

In this section, we introduce Spliceosome, an innovative solution specifically designed to tackle both challenges 1 and 2 concurrently. Unlike traditional methods that rely on optimizing frame-grained parameters such as *brightness* and *contrast* across the entire video, Spliceosome takes a novel approach by lowering the adjustment granularity and focusing on region-oriented modifications. This strategic shift allows Spliceosome to better target specific areas within each frame that require enhancement, thereby efficiently addressing the issue of *pixel recession*, which can significantly impair video analysis tasks. By employing this method, Spliceosome not only aims to substantially lower the end-to-end delay associated with video encoding and streaming but also seeks to minimize the inference overhead, all while enhancing the analytical accuracy crucial for precise DNN inference operations. Figure 3 illustrates the streamlined workflow of Spliceosome, providing a visual representation of its components and processes.

Camera Side Execution. The execution on the camera side is orchestrated through three interlinked components: the *Anchor Region Proposal Builder*, the *F1_AccGrad Scheduler*, and the *Single-Channel Codec*. These components collaborate seamlessly to optimize video processing by leveraging the available computing resources within the camera. Initially, Spliceosome analyzes the raw captured video frames to extract motion vectors, which serve as the foundation for the *Anchor Region Proposal Builder*. This builder plays a crucial role in pinpointing specific *anchor region proposals* that denote areas of interest within frames, ensuring that subsequent processing tasks are targeted and efficient. The *F1_AccGrad Scheduler* then steps in to dynamically adjust the *brightness* and *contrast* of these identified anchor regions, refining the image quality to enhance visibility and analytical precision. Lastly, the optimized video segments are encoded using the *Single-Channel Codec*, which efficiently compresses the data before transmitting it to the edge server for further DNN-based inference. Detailed explanations of the algorithms used for ARP identification, knobs adjustment, and video encoding are provided in Sections IV, V, and VI, respectively.

Server Side Execution. Upon receiving the processed video from the mobile camera, the edge server undertakes a systematic approach, beginning with the extraction of single-channel frames. These frames are then fed into the pre-trained deep neural networks, purpose-built to analyze and interpret the incoming video data with high precision. The feedback loop integral to Spliceosome's operation is anchored in the inference outcomes of these DNNs. The accuracy metrics from these results are relayed back to the *F1_AccGrad Scheduler*, enabling it to make informed adjustments in future video processing cycles. This adaptive learning mechanism ensures that Spliceosome persistently enhances its performance, even under the challenging conditions imposed by *pixel recession*.

IV. Anchor Region Proposal Builder

When a new video frame arrives, Spliceosome builds the potential anchor region proposals (ARPs) of this frame based on the extracted motion vectors. In this section, we present two key designs that are critical to achieve effective ARP prediction: a MV-based cluster to generate the initial ARPs and a fine-tune *padding* method to generate the final ARPs.

A. Motion Vector Cluster

As we know, motion vectors, which indicate the pixel offset between frames, are widely used to improve compression ratio in existing advanced codecs. Generally, for videos captured by a fixed camera, MVs are likely to result from moving objects, which Spliceosome aims to detect in this paper. Commodity cameras are usually equipped with specific hardware to accelerate video encoding and calculate the motion vectors. Typically, a longer interval between the reference frame and the current frame may result in excessive MVs that may cover the whole frame; hence, we regard the previous frame as the reference frame. Assuming that the MV set of current frame f is $\mathcal{M}(f)$, the goal of this module is to effectively building the initial ARPs from $\mathcal{M}(f)$.

Intuitively, we can cluster them into several classes, and each class represents a target ARP. The traditional *K-Means Clustering* method achieves $O(|\mathcal{M}(f)|KT)$ time complexity, where K and T are the number of classes and iteration rounds, respectively. However, this method needs to specify the value of K in advance. Without prior feedback information about the amount of objects from server-side, it is challenging to make assignment for K. In such dilemma, we consider a *hierarchical iterative algorithm* to find the optimal K. We summarize three key steps as follows:

- *Step 1*: For the extracted MVs $\mathcal{M}(f)$, it first sets a large value to K, denoted by K_{max}. Then, it runs the general K-means clustering algorithm and returns the initial class set $\mathcal{M}_1(f)$, where $|\mathcal{M}_1(f)|$ equals K_{max};
- *Step 2*: For $\forall m_i \in \mathcal{M}_1(f)$, it finds $m_j = argmin_m E(m_i, m)$, where $m \in \mathcal{M}_1(f) - m_i$. Then, if $E(m_i, m_j) \leq \delta$, it merges m_i and m_j and updates their union central point, where $E(\cdot)$ indicates the euclidean distance between two central points and δ is the given threshold; otherwise, it does nothing for m_i. Note that the newly merged set will not participate in the secondary merge in one iteration. We denote the output set by $\mathcal{M}_2(f)$.
- *Step 3*: For $\mathcal{M}_2(f)$ and following sets (if exist), it repeats the similar operations in step 2 until the distance between any two subsets in the previous set is larger than

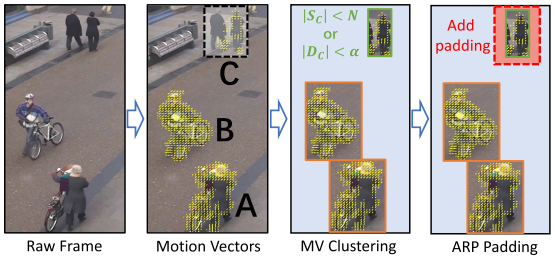


Fig. 4. Workflow of Anchor Region Proposal Builder.

δ. We denote the final set by $\mathcal{M}_{min}(f)$ and regard $|\mathcal{M}_{min}(f)|$ as K to cluster the MVs.

Generally, the server receives live videos in chunks, and each chunk consists of several groups of pictures (GoP) [25]. An independent GoP contains two types of frames: 1) a key frame (I-frame) that contains complete information, and 2) P and B frames that are encoded based on the I-frame. Considering that consecutive frames in one or several video chunks probably contain the same number of objects, we only run the *hierarchical iterative algorithm* in the first I-frame to acquire the number of clusters K, which is then used directly in the subsequent frames for general K-*Means Clustering*.

B. Anchor Region Proposal Padding

Through simple clustering of motion vectors (MVs), we obtain initial ARPs. However, using these ARPs directly in follow-up operations (e.g., encoding and inference) can lead to a degradation in analytical performance, as an initial ARP is often not large enough to cover the corresponding target objects for three reasons. Firstly, for objects that contain a large number of similar pixels (e.g., in Fig. 4, the left person in region C is wearing a large black coat, and within the coat region, the pixels are almost identical), it extracts less useful motion vectors whose lengths are higher than the minimum value (e.g., 2), resulting in incomplete coverage of the target objects. Despite movement between frames, this results in fewer pixel offsets and therefore fewer motion vectors. Secondly, based on the law that *the object is big when near and small when far*, the visual moving intensity of objects far away from the camera is lower than that of objects close to the camera, resulting in smaller and shorter motion vectors for the former objects. Thirdly, when the object pixels closely resemble the surrounding pixels (i.e., *pixel recession*), the quantity of MVs is also insufficient to cover the objects.

To address this problem, we propose a padding-based method to expand the initial ARPs. Specifically, for an initial ARP $\mathcal{A}$ with shape (R, C), we define its MV-density $D_{\mathcal{A}}$ as $\frac{|\mathcal{S}_{\mathcal{A}}|}{R \times C}$, where $\mathcal{S}_{\mathcal{A}}$ is the MV set of ARP $\mathcal{A}$, and then we design a strategy to mitigate the effect caused by the aforementioned two reasons. If we detect that 1) $D_{\mathcal{A}}$ is lower than α, which is probably caused by the first reason, or 2) $|\mathcal{S}_{\mathcal{A}}|$ is smaller than the N, which results from the second reason, then we expand ARP $\mathcal{A}$ by γ times, where parameters α, N, and γ jointly control the level of expanding. It is clear that a larger α or N leads to more ARPs to perform expanding operation, and a bigger γ grants more confidence to completely cover the objects. In view of managing larger ARPs inevitably causes extra overhead in the follow-up procedures, the values of these parameters should be well-assigned.

It's worth noting that Spliceosome's approach with ARP prediction differs from prior works [13], [14], [37] in that it does not focus solely on predicting ARP for encoding optimization. Rather, the main goal is to reduce the overhead of adjusting various knobs. To accomplish this, the proposal is to replace frame-level tuning with ARP-level tuning in Section V.

V. F1_AccGrad Scheduler

This section details the methodology of *F1_AccGrad Scheduler*, including its objective and approach.

A. Problem Definition and Objective

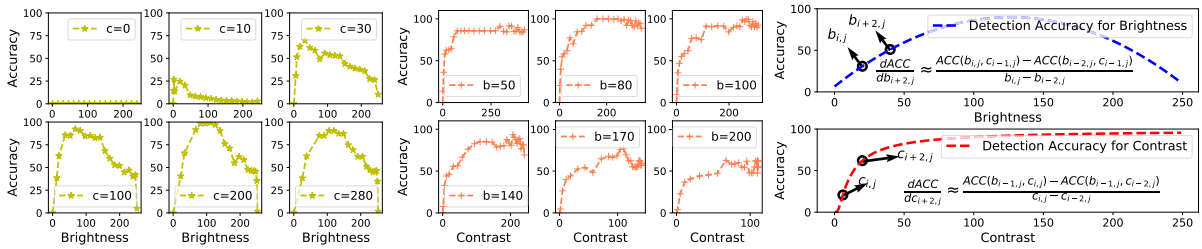
In general, video analytical performance is affected by the encoding quality (i.e., bitrates), brightness and contrast (described in section II). Assuming that we encode each ARP with equal quality (i.e., same QP), hence we only consider the later two factors. Given a video chunk C with F frames, *Anchor Region Proposal Builder* generates K ARPs, the goal of this module is to find the optimal brightness b and contrast c for each ARP, that maximize the overall detection accuracy. Thus, we express our optimization goal as the average accuracy of each frame in chunk C:

$$\mathcal{P}: \max_{\{b_{i,j}, c_{i,j} | i \le F, j \le K\}} \frac{1}{F} \sum_{i}^{F} \sum_{j}^{K} ACC(b_{i,j}, c_{i,j})$$
$$\text{s.t, } \forall i, \forall j, b_{i,j} \in [0, 255], c_{i,j} \in [0, 400] \quad (1)$$

where $b_{i,j}$ and $c_{i,j}$ indicate brightness and contrast decisions for ARP j in frame i respectively, $ACC(\cdot, \cdot)$ represents the achieved detection accuracy given $b_{i,j}$ and $c_{i,j}$. Considering the intra-frame similarity, it is improbable for the frame contrast to be excessively high. Therefore, we have set a maximum limit of 400 for its value.

B. Alternate Gradient Ascent Algorithm

We now focus on efficiently solving the above problem $\mathcal{P}$. A naive solution is to run all potential combinations of $\{(b_{i,j}, c_{i,j})\}$ and select the optimal one. However, resource-constrained cameras hardly support expensive DNN inference. In addition, the large search overhead significantly affects the real-time of video analytics. In such dilemma, we turn to exploit the property of function $ACC(\cdot, \cdot)$. Without prior works that reveal concrete function-form among *brightness*, *contrast* and *accuracy*, we conduct a trace-driven experiment to acquire their approximate correlations. On the basis that tuning both brightness and contrast simultaneously complicates the $ACC(\cdot, \cdot)$ profiling, we choose to fix one factor, and investigate the relationship between another factor and detection accuracy. Specifically, we first extract a total of 100 video frames from BridgeCam LIVE [21] and Rotary Traffic Live [22], all of these frames have similar initial brightness and contrast. Then, we define a tuning epoch, in which only one knob is adjusted. Finally, we run the state-of-the-art yolov5 [29] for each epoch.



(a) Accuracy vs. brightness (b) Accuracy vs. contrast (c) Accuracy gradient computing

Fig. 5. Motivation of alternative gradient ascent. (a) Detection accuracy when tuning brightness but fixing contrast; (b) Detection accuracy when tuning contrast but fixing brightness; (3) Accuracy gradient computing.

Fig. 5a and Fig. 5b illustrate the measurement results of multiple epochs, from which we derive two observations: 1) $ACC(b, \#)$ (i.e., fixing contrast) is concave because accuracy is significantly improved at first but gradually decreased if the brightness exceeds a specific threshold, and 2) $ACC(\#, c)$ (i.e., fixing brightness) is also concave as accuracy monotonically increases with respect to increase in the contrast, whereas the marginal improvement diminishes. The concavity of above two functions allows us to apply *alternate gradient ascent* to obtain the global optimum [26]. Suppose that the knob decision for ARP j of frame i in chunk C is $(b_{i,j}, c_{i,j})$, the update rule for frame $i+1$ (i.e., adjusting b) is represented as

$$(b_{i+1,j}, c_{i+1,j}) = \left(\alpha_1 \cdot \frac{dACC(b_{i,j}, c_{i,j})}{db_{i,j}} + b_{i,j}, c_{i,j}\right), \quad (2)$$

and for frame $i+2$ (i.e., adjusting c)

$$(b_{i+2,j}, c_{i+2,j}) = \left(b_{i+1,j}, \alpha_2 \cdot \frac{dACC(b_{i+1,j}, c_{i+1,j})}{dc_{i+1,j}} + c_{i+1,j}\right), \quad (3)$$

where α_1 and α_2 are the step sizes for brightness and contrast updating respectively. It is well-known that a larger step size can lead to significant updates in the knob, resulting in either a near-optimal value or an unfortunate miss. To address this, we have implemented an adaptive method that automatically adjusts the step size based on the gradient magnitude. If the gradient is large, the step size is increased; otherwise, it is decreased. For frames within the same chunk, we utilize an alternate method to update the knobs until we no longer observe appreciable accuracy improvements. If the knobs (i.e., brightness and contrast) converge in one frame, then all subsequent frames in this chunk use the latest knob settings.

Due to the complexity of profiling an analytical closed-form equation for detection accuracy and its derivative, as depicted in Fig. 5c, we rely on a numerical method to derive an approximate gradient using live measurement values. To acquire the partial derivative of $ACC(b_{i+2,j}, c)$, an intuitive method is to use the slop of two recent accuracies (i.e., $ACC(b_{i,j}, c_{i-1,j})$ and $ACC(b_{i,j}, c_{i-1,j})$) to approximate it.

Offline Analyzer. Unfortunately, the resource-constrained camera is not capable of directly computing the accuracy slop, and server-driven approach is also hindered by the lack of full ground-truth references. In such dilemma, we propose a server-side Offline Analyzer. Specifically, for these recent frames from the same camera, the edge server leverages the idle compute-resources to run DNNs inference on them under each potential knobs (i.e., brightness and contrast) assignments. Then, it regards the output that has the maximum bounding boxes and maximum average confidence as the *ground truth*, based on which it is able to calculate the accuracy under each knob setting. Finally, the server encapsulates it into a configuration file and returns it to the camera to guide the knob adjustment. Once camera receives this file, it updates the local configuration file.

Therefore, to obtain the partial derivative of $ACC(b_{i+2,j}, c)$, the camera derives two accuracies under knobs $(b_{i,j}, c_{i-1,j})$ and $(b_{i,j}, c_{i-1,j})$ respectively from the latest configuration file, and uses their slop as the derivative. Similarly, we regard the slop between accuracies under knobs $(b_{i-1,j}, c_{i,j})$ and $(b_{i-1,j}, c_{i-2,j})$ as the partial derivative of $ACC(b, c_{i+2,j})$. Both operations of configuration file lookup and accuracy slop computing are lightweight, incurring negligible overhead.

VI. Single-Channel Codec

In this section, we introduce Single-Channel Encoding that is able to significantly reduce the encoding difficulty, data volume and accelerate the final DNN inference while not decreasing the analytical performance.

Modern video codecs (e.g., VP9, AV1 and HEVC), which runs multiple processes such as DCT transform to encode the three-channel videos, faces two critical challenges. Firstly, it involves multiple format conversions, incurring extra overhead. For example in H.265, it needs to convert the video format from *RGB* to *YCrBr* in advance., and transmits the videos with bitstream form. Secondly, despite efficient mechanisms (e.g., inter-prediction, and quantification) that use video redundancy to reduce volume, it is still challenging when facing terrible network connection. What's more, as described before, compared with three-channel input, single-channel input significantly lowers the DNN inference overhead (e.g., inference time and runtime GPU memory usage) in multiple layers such as convolution and pooling. Hence in this module, we make several optimizations based on the open source x265 HEVC Encoder [27] below.

Efficient Format Transformation. For the input video frame with three channels (i.e., RGB), we first transform it into *YCbCr* format, where *Y* is computed based on the *ITU-R Recommendation BT.601* [28], and *Cb* and *Cr* are automatically set to 0. In this way, we only need to compute *Y* but care less about other information (i.e., *Cb* and *Cr*), which largely improves the format transforming efficiency. Then, in the decoding stage that regenerates RGB frame, based on *BT.601* [28], any channel information of *RGB* is equal to *Y*.

Hence we only extract one channel information for DNN input, which accelerates the final inference procedure.

Lightweight Single-Channel Sampling. It makes critical sense to assign less resolution for chroma information than for luma information for video encoding, taking advantage of the experimental result in Section VIII that CNN expresses acceptable performance despite only offering luma information. Hence, the subsampling scheme, commonly expressed as a three-part ratio, is set to 4:0:0 in this paper. By this way, we achieve single-channel encoding by only maintaining luma information (i.e., *Y*), which further reduces the amount of transmitted video volume.

ARP-oriented Quantization Parameter (QP) adaptation. To some extent, the Quantization Parameter (QP) reflects the level of compression of spatial details. A smaller QP value means that most of the details will be retained, which in turn requires a higher bitrate for encoding. On the other hand, increasing the QP value might lead to missing some details, thereby requiring less bitrate but worsening the frame distortion. When combining Anchor Region Proposal (ARP) and QP, a common approach is to apply a low QP value (e.g., 10) to encode each ARP, while assigning a high QP value (e.g., 40) to encode the areas outside these ARPs. However, the network condition plays a critical role in determining the appropriate QP value. When the bandwidth is sufficient, it is strongly recommended to use a much low QP value for encoding. However, in the case of poor network conditions, this may not be feasible. In such a situation, we prioritize assigning a low QP value to ARPs to ensure the final detection accuracy, while setting the QP value in other regions to the lowest possible value that matches the remaining bandwidth.

VII. System Implementation

We implement a prototype of Spliceosome in Python and C++ for easy integration with video analytics applications.

Hardware Setup. We deploy Spliceosome's edge server on a Dell PowerEdge R740 server, a reliable and flexible server solution designed for modern data processing requirements. This server configuration is enhanced with a NVIDIA GeForce RTX2080 Ti GPU, which provides substantial computational power essential for video analytics tasks. Operating on the Ubuntu 20.04 system, well-known for its stability and wide range of supported applications, our setup ensures compatibility with diverse software environments.

For capturing video inputs, we employ a mobile development board, the Nvidia Jetson TX2, acting as the camera. This board is equipped with a Dual-Core NVIDIA Denver 2 64-Bit CPU and a Quad-Core ARM Cortex-A57 MPCore CPU, complemented by an integrated NVIDIA Pascal GPU with 256 CUDA cores. This potent combination allows the board to handle several light-weight computations efficiently. However, despite its capabilities, the Jetson TX2 does not inherently support real-time deep neural network (DNN) inference for video analytics, necessitating reliance on edge computing resources to ensure quick, responsive video data processing.

Software Implementation. On the server side, Spliceosome is built upon the `Pytorch 1.9.0` framework, a highly flexible and efficient open-source machine learning library. By integrating the NVIDIA `TensorRT SDK`, we capitalize on advanced optimization techniques that expedite DNN inference, essential for real-time analytics during both online detection and offline profiling processes. These techniques ensure that computational resources are utilized efficiently, minimizing latency and improving throughput. The server efficiently communicates profiling results back to the camera, employing the `TCP KeepAlive heartbeat detection` mechanism to maintain a persistent connection, which enhances reliability and reduces transmission delays.

On the camera side, Spliceosome uses a unique approach to extract motion vectors, essential for detailed video analysis. This task is accomplished using the `extract_mvs.c` executable file, a utility provided by the versatile media processing library, `ffmpeg`. The extracted motion vectors, represented as `AVMotionVector`, offer critical insights into frame-by-frame motion patterns, which are pivotal in executing precise video analytics. By leveraging these vectors, Spliceosome can efficiently assess and process video data, augmenting the overall effectiveness of our system. Corresponding to the three key components of Spliceosome, the camera needs to implement three abstract functions:

```
1. def ARP_build
       (mvs: List[vector])->List[ARPs]
2. def Accuracy_opt
       (ARP: List[int])->Tuple[int]
3. def Encode
       (img: numpy.array)->Bitstream
```

Spliceosome employs the first API to build the anchor region proposal, `func ARP_build` consists of clustering and padding operations. Then, based on the generated ARPs, the second API is to perform knobs optimization on each ARP. After that, the final API is designed to improve the format conversion (`func X265EncodeYUV`), chroma subsampling, and QP adaptation (`func AdaptiveQP`) based on the open source `x265 HEVC Encoder`.

VIII. Evaluation

In this section, we evaluate the performance of Spliceosome in terms of end-to-end delay, detection accuracy and inference efficiency. The results demonstrate that our system is able to achieve both low end-to-end delay and high inference accuracy requirement of video analytics applications under diverse network status and *Pixel Recessio*n. Specifically, compared to other alternative designs, Spliceosome improves accuracy by 4.71-14.47%, reduces the average response time by 40.94-58.71%, and speeds up inference by 14.28%.

A. Experiment Setup

CNN Model and Datasets. We focus on object detection application in this paper, and utilize the state-of-the-art yolov5 [29] model. To gather datasets for both real-time video analytics and offline profiling, we select two types of live camera videos from YouTube [23]: BridgeCam LIVE [21] and Rotary Traffic Live [22]. For each category, we record 20 minutes of continuous video every hour (e.g. from 5:00-6:00 pm), resulting in a total length of 480 minutes. Half

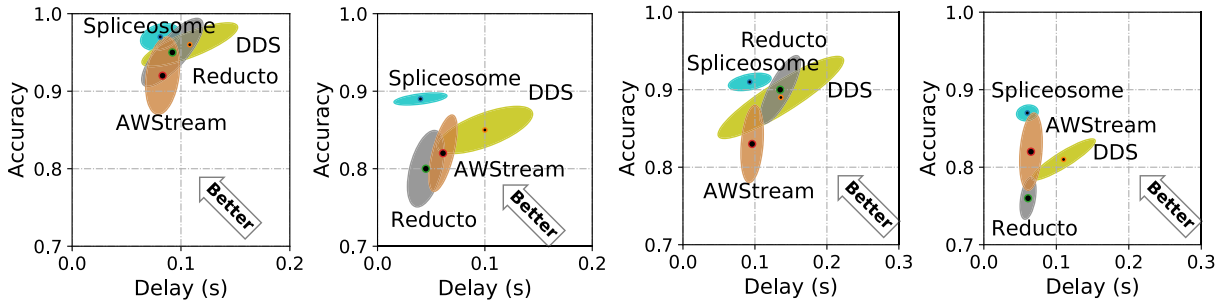


Fig. 6. The total delay vs. inference accuracy of Spliceosome and several baselines on two video datasets (i.e., BridgeCam Live video and Rotary Traffic video) under the condition with or without PR (i.e., *pixel recession*).

of the recorded videos are used for the online stage and the other half are used for offline profiling. It's worth noting that BridgeCam Live video features fewer objects, whereas Rotary Traffic video presents a larger number of objects, some of which are small in size.

Networks. In general, accurately estimating real-time available bandwidth is challenging despite many efforts to do so [30], [31], [32]. Therefore, to emulate real network conditions, we created a corpus of network traces by combining several public datasets and a network emulation tool: a broadband dataset provided by the FCC [33] and the Mahimahi tool [34]. The FCC dataset contains over 1 million throughput traces, each of which records the average value over 2,100 seconds with a granularity of 5 seconds. We selected 200 traces (each with a duration of 500 seconds) from the *Web browsing* category in the February 2016 collection to add to our corpus. The Mahimahi tool generates traces that capture the time-varying throughput of the U.S. cellular network as experienced by mobile users. Similarly, we added 200 Mahimahi traces to our corpus. In total, we collected 400 traces for evaluation. Instead of randomly picking the traces, we sampled them based on the throughput distribution over the entire timescale, excluding traces with an average bandwidth exceeding 10 Mbps to emulate a bandwidth-constrained environment.

Baselines. We use baselines from three categories: 1) Server-driven method *DDS* [13], which first sends low-quality video to server, and re-sends the high-quality region of interest based on the feedback, while Spliceosome does not require server feedback and multiple transmissions; 2) Frame filtering method *Reducto* [11], which uses local resources to filter out the redundant frames, while Spliceosome does not filter out several frames and thus does not degrade the analytics accuracy; and 3) Configuration adaptation method *AWStream* [40], which balances accuracy and processing delay, while Spliceosome also adjusts the brightness and contrast.

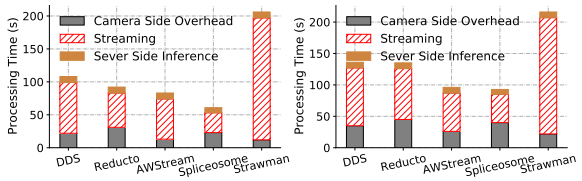
Parameters Selection. 1) In module *Anchor Region Proposal Builder*, we set the initial K_{max} and distance threshold δ to 40 and 0.15 respectively for *hierarchical iterative algorithm*, and set α, N and γ to 0.15, 20, and 0.2 respectively to jointly control the padding level; 2) In module *F1_AccGrad Scheduler*, we use a high α_1 and α_2 (i.e., 0.5) when achieving a high gradient, otherwise applying a low value (i.e., 0.1); and 3) In *Single-Channel Codec*, we set the *QP* for ARP to 15, and assign a higher *QP* for other regions based on the left bandwidth.

B. Improvement of Overall Performance

Fig. 6 illustrates the achieved accuracy and resulting response delay of Spliceosome and other baselines on these two video datasets, with each ellipse representing 80% of the results. We divide each video type into two subsets, one recorded between 6:00am and 18:00pm and the other between 18:00pm and 6:00am, corresponding to cases with and without *pixel recession* (PR), respectively. For the BridgeCam Live video without PR in Figure 6a, all alternative designs achieve high accuracy (0.97, 0.96, 0.95, and 0.92 for Spliceosome, DDS, Reducto, and AWStream, respectively) while exhibiting varying delays (60.8ms, 108.1ms, 92.1ms, and 83.1ms, respectively). DDS improves accuracy through server-driven feedback but incurs additional time, while Reducto filters out redundant frames, reducing transmission volume. On the other hand, AWStream selects the frame encoding configuration purely based on delay and accuracy to achieve a compromise between the two.

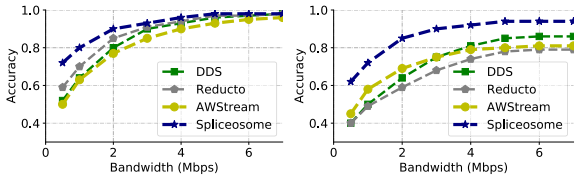
For BridgeCam Live video with PR in Fig. 6b, due to the similarity of objects pixel and surrounding pixel, DDS, Reducto and AWStream present a significant accuracy drop (11%, 15% and 10% respectively). The PR condition causes AWStream to filter out excessive frames, resulting in a decrease in accuracy. Compared to DDS, Reducto and AWStream, Spliceosome achieves 4.71%, 11.25% and 8.54% higher accuracy gains, and 60%, 11.14% and 34.43% delay reduction, respectively. We then verify their performance in Rotary Traffic video, it is clear that in Fig. 6c, they acquire lower accuracy and higher delay (e.g., 92.8, 136.1, 135.1 and 96.1 ms for Spliceosome, DDS, Redcuto and AWStream respectively) compared those in BridgeCam Live video, as this testing videos contain more and small objects. It is aggravated by PR condition in Fig. 6d. Nonetheless, Spliceosome improves the accuracy by 7.4%, 14.47% and 6.12% compared to DDS, Reducto and AWStream respectively.

Delay Distribution. Fig. 7 shows the distribution of each components, including camera side overhead (i.e., local heuristic and encoding), video transmission and server side inference. Note that Strawman directly transfers high-quality video to remote server. Case for BridgeCam Live video in Fig. 7a, Spliceosome accelerates the inference by 14.28% (from 9.1ms to 7.8ms), and outperforms DDS, Reducto and AWStream by 61.04%, 42.31% and 50.82% in terms of streaming delay respectively, which is largely due to the single-channel encoding mechanism. Though ran in Rotary Traffic video,



(a) Delay distribution in BridgeCam (b) Delay distribution in Rotary

Fig. 7. The distribution of resulting delay for Spliceosome and several baselines in two testing videos.



(a) Accuracy-Bandwidth without PR (b) Accuracy-Bandwidth with PR

Fig. 8. Accuracies of Spliceosome and several baselines under diverse bandwidths with or without PR.

Spliceosome is still able to improves the streaming delay by 51.09%, 44.4% and 26.22% respectively. Since Spliceosome leverages the camera side resource to build ARP and run *F1_AccGrad Scheduler*, it causes extra slightly higher overhead but leaves negligible effects to the overall performance. Above all, Spliceosome reduces the end-to-end delay by 46.45% and 36.14% compared to DDS, Reducto and AWStream respectively in BridgeCam Live video, and achieves 33.07% and 32.54% in Rotary Traffic video.

Accuracy under Diverse Bandwidths. Figure 8 illustrates the correlation between detection accuracy and available bandwidth in BridgeCam Live video. In Figure 8a, even with insufficient bandwidth (e.g., 0.5 Mbps), Spliceosome still achieves good results (e.g., 0.72 accuracy) because it uses single-channel information for encoding and assigns more bitrate for the selected ARPs. However, other baselines heavily rely on bandwidth and achieve lower accuracy, less than 0.6. The situation is worse when the video is in PR, as Figure 8b shows. DDS, Reducto, and AWStream only achieve around 0.4 accuracy, while Spliceosome achieves 0.6. Moreover, even given enough bandwidth (e.g., 6 Mbps), other designs fail to achieve the 0.95 accuracy achieved by Spliceosome.

C. Sensitivity to Parameter Settings

We explore the impacts of five parameters on Spliceosome. Without loss of generality, we fix other parameters when tuning a parameter. For parameters $\delta, (\alpha, N)$ and γ, we measure the resulting ratio of the sum of each ARP area to the whole-frame area and achieved accuracy; For QP, we evaluate the accuracy and normalized data volume compared to the performance using the smallest QP.

Impact of δ: We tune the assignment of δ from the candidate set $\{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ and depict the results in Fig. 9a. The accuracy increases with respect to the increase of δ, whereas the marginal improvement diminishes when δ exceeds 0.15. However, the ratio shows exponential growth when δ is set from 0.15 to 0.3. The result in Fig. 9b reveal that a larger δ leads to a larger ARP that needs more bitrate to encode it, thus inevitably increasing the data volume.

Impact of α, N and γ: They are jointly to control the ARP padding level. As Fig. 9c shows, setting α and N to 0.05 and 5 respectively significant lowers the ratio (about 0.12), but obtains only 0.55 accuracy. Selecting a higher α and N (e.g., 0.2 and 40) does improve the detection accuracy (0.95), but the considerable data volume resulting from huge ARPs remarkably aggravates the network bottleneck. Similarly, a small γ 0.05 can hardly enhance the accuracy but greatly lowers the ratio of ARPs (about 0.15); a bigger γ 0.4 leads to large ratio about 0.95 and high accuracy 0.98. Above all, we should well balance the ratio of ARPs area to the whole frame and detection accuracy by making adaptive assignments.

Impact of QP: We evaluate the effect of QP selection from candidate set $\{5, 10, 15, 20\}$ on the data volume and accuracy. As shown in Fig. 9d, it achieves 0.90 and 0.95 in accuracy and data volume respectively when setting QP to 5, whereas they are 0.1 and 0.78 respectively when QP is 20. It indicates that QP assignment severely affects Spliceosome performance in terms of detection accuracy and data volume, thus is proposed to be carefully tuned.

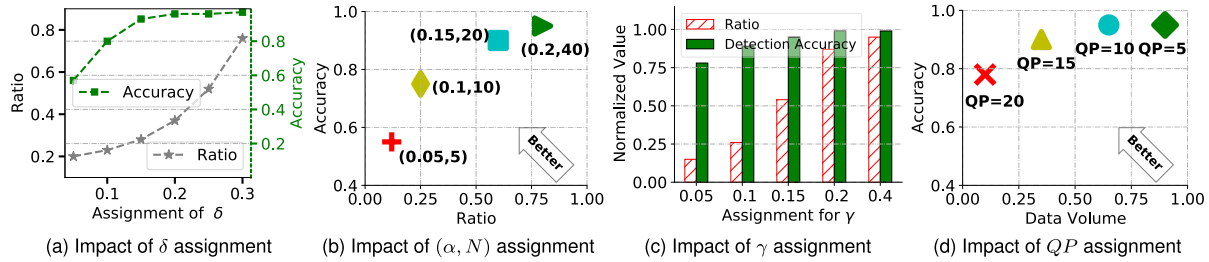
D. Pipeline Execution

Since camera-side operations, video streaming and server-side inference consume different resources (i.e., camera resources, available bandwidth and server GPUs or FPGAs resources respectively) that do not affect each other, hence to make full use the resources, existing general systems such as DDS and AWStream for video analytics enable these three components to be efficiently pipelined and executed in parallel. Fig. 10 compares the performances of different pipeline system, where the pipeline length is set to 10.

Figure 10a shows that DDS spends the majority of its time (around 71.23%) on video transmission, which is largely due to the secondary transmission based on server feedback. On average, it takes 80.11 ms to process each frame. In contrast, Reducto in Figure 10b makes full use of local resources to filter out a large amount of redundant frames and shortens the time spent on camera-side execution and video streaming (i.e., 31 ms and 52 ms). As a result, it achieves an effective pipeline with an average response time of 56.01 ms. AWStream in Figure 10c makes configuration decisions on the server-side, incurring little camera-side overhead (about 13 ms for video encoding). Hence, on average, it takes 63.21 ms to process each frame. Our proposed Spliceosome in Figure 10d utilizes camera-side resources to find potential ARP and uses a single-channel codec to reduce the data volume. This approach achieves an average response time of 33.08 ms. It is evident that both Reducto and Spliceosome obtain effective pipeline execution, as they control the processing time of each step (e.g., camera processing and video streaming) to be consistent. Overall, when arranging each stage to be pipelined and executed in parallel, Spliceosome improves the average response delay by 58.71%, 40.94%, and 47.67% compared to DDS, Reducto, and AWStream, respectively.

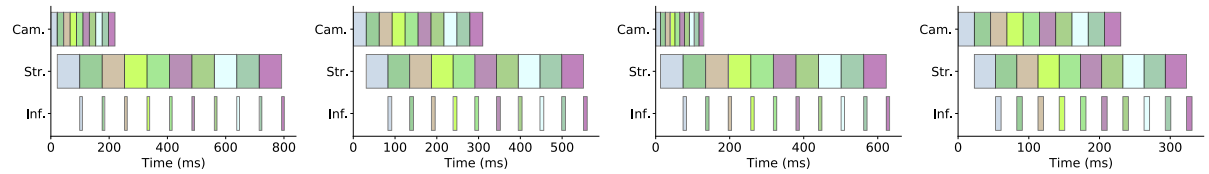
IX. Discussion and Future Work

We notice that the performance of Spliceosome largely relies on the accuracy of ARP generation. Precise ARP predic-



(a) Impact of δ assignment (b) Impact of (α, N) assignment (c) Impact of γ assignment (d) Impact of QP assignment

Fig. 9. The data volume vs. inference accuracy of Spliceosome under diverse parameter assignments.



(a) DDS pipeline execution (b) Reducto pipeline execution (c) AWStream pipeline execution (d) Spliceosome pipeline execution

Fig. 10. Pipeline execution of Spliceosome and several baselines. "Cam.", "Str.", and "Inf." indicate the procedures of camera-side execution, video streaming and server-side inference, respectively.

tion, which benefits not only the knobs tuning to improve the analytical accuracy, but also the video encoding to reduce the video volume, is hindered by the video content. For instance, no MVs are extracted under the extremely severe *pixel recession* (e.g., with almost 0 contrast); whereas excessive MVs are generated for video with more objects and frequent scene changes, and more ARPs incurs larger camera-side overhead.

Hence several designs of Spliceosome warrant further investigation. First, it probably works by enhancing the contrast of referenced frame in MV extracting. Second, reducing the number of updates by setting adaptive step sizes is able to accelerate the knobs tuning. Thirdly, designing a new module to detect the stationary objects is necessary. Lastly, more types of videos are needed to verify the system scalability.

X. Related Work

In this section, we discuss the most closely related work.

Video Analytics System. The growing demand for video analytics has led to numerous research efforts focused on different areas such as camera-edge-cloud collaboration [18], [35], [36], [37], [38], [39], [40], [41], [42], [57], DNN sharing [44], [45], [46], [47], and finding the optimal balance between accuracy and cost [17], [43], [48], [49], [50], [51], [52], [53]. For instance, ELF [17] partitions frames and offloads the slices to multiple edge servers for parallel inference; Neurosurgeon [45], DeepThings [46], and Cracking Open the DNN [47] propose to partition the DNN and distribute the layer slices across edge servers to reduce inference delay; Chameleon [52] VideoStorm [49] and AWStream [40] use adaptive configuration to balance accuracy and overall delay. However, they fall insufficient to solve the issue of *pixel recession.*

Camera-Side Efforts for Video Analytics. FilterForward [12] and Reducto [11] perform on-camera sampling based on video features and filtering thresholds, respectively, significantly reducing transmission volume. Research [54], [55], [56], [57] aims to partition models and schedule them to multiple mobile devices for collaborative inference. AWS Wavelength [58] migrates Amazon Web Services to Verizon's 5G edge computing platform. NVIDIA and Intel enable real-time AI inference at the edge by providing NVIDIA EGX A100 [60] and Intel Xeon Scalable Processor with Intel Deep Learning Boost [59]. However, these methods require powerful on-device compute-resource to run the expensive heuristics.

Anchor Region Proposal Prediction. To reduce transmission volume or inference computation, numerous research works [13], [17], [37], [61] propose emphasizing regions of interest while weakening other regions. For instance, DDS [13] first sends a low-quality frame to the server and then resends high-quality regions based on feedback from the server. ELF [17] designs an attention-based LSTM network to crop the regions of interest; however, this operation is largely constrained by local compute resources. Supremo [61] predicts regions of interest within a frame based on the quantity of detected *edges* and uses neural super-resolution techniques to improve human-perceived visual quality. However, it often mistakes regions in the background as target objects. In contrast, Spliceosome leverages the motion vector among frames to achieve efficient and accurate ARP prediction.

XI. Conclusion

In this paper, we propose Spliceosome, a new video analytics framework that effectively reduces overall delay and enhances detection accuracy in the face of pixel recession. Spliceosome avoids relying on either local neural inference or server-driven feedback and instead employs intermediate motion vectors to generate targeted ARPs. To overcome the decrease in accuracy caused by pixel recession, we propose an accuracy optimizer that automatically adjusts the frame settings for brightness and contrast. In addition, Spliceosome uses the x265 HEVC Encoder to encode frames with one-channel information, which significantly reduces data volume and accelerates DNN inference. We implemented Spliceosome using commercial off-the-shelf hardware, and our experiments demonstrate that it enhances detection accuracy by 4.71-14.47%, reduces average response time by 40.94-58.71%, and speeds up inference by 14.28%.

References

[1] (2021). *Trafficvision: Traffic Intelligence From Video*. [Online]. Available: http://www.trafficvision.com/

[2] VisionZero Official Website. *The Vision Zero Initiative*. Accessed: May 2, 2024. [Online]. Available: http://www.visionzeroinitiative.com/

[3] TrafficTechnologyToday. *AI Traffic Video Analytics Platform Being Developed*. Accessed: May 2, 2024. [Online]. Available: https://www.traffictechnologytoday

[4] GoodVision. (2021). *GoodVision: Smart Traffic Data Analytics*. [Online]. Available: https://goodvisionlive.com/

[5] intuVision. (2021). *Intuvision Va Traffic Use Case*. Accessed: May 2, 2024. [Online]. Available: https://www.intuvisiontech.com/intuvisionVA_solutions/intuvisionVA_traffic

[6] Microsoft. (2019). *Traffic Video Analytics Case Study Report*. [Online]. Available: https://www.microsoft.com/en-us/research/publication/traffic-video-analytics-case-study-report/

[7] SecurityInfoWatch. (2012). *Market for Small IP Camera Installations Expected To Surge*. [Online]. Available: http://www.securityinfowatch.com/article/10731727/

[8] SecurityInfoWatch. (2016). *Data Generated By New Surveillance Cameras To Increase Exponentially in the Coming Years*. [Online]. Available: http://www.securityinfowatch.com/news/12160483/

[9] Slate. (2019). *Humans Cant Watch All the Surveillance Cameras Out There, So Computers Are*. [Online]. Available: https://slate.com/technology/2019/06/video-surveillance-analytics-software-artificial-html

[10] GrandViewResearch. (2018). *Video Analytics Market Size Worth $9.4 Billion By 2025 Cagr: 22.8%*. [Online]. Available: https://www.grandviewresearch.com/press-release/global-video-analytics-market

[11] 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. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, Jul. 2020, pp. 359–376.

[12] C. Canel et al., "Scaling video analytics on constrained edge nodes," in *Proc. Mach. Learn. Syst.*, 2019, pp. 406–417.

[13] K. Du et al., "Server-driven video streaming for deep learning inference," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, Jul. 2020, pp. 557–570.

[14] K. Du, Q. Zhang, A. Arapin, H. Wang, Z. Xia, and J. Jiang, "AccMPEG: Optimizing video encoding for accurate video analytics," in *Proc. Mach. Learn. Syst.*, 2022, pp. 450–466.

[15] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, "Neural-enhanced live streaming: Improving live video ingest via online learning," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, Jul. 2020, pp. 107–125.

[16] A. Horé and D. Ziou, "Image quality metrics: PSNR vs. SSIM," in *Proc. 20th Int. Conf. Pattern Recognit.*, Aug. 2010, pp. 2366–2369.

[17] W. Zhang et al., "Elf: Accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, Sep. 2021, pp. 201–214.

[18] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the edge-cloud barrier for real-time advanced vision analytics," in *Proc. 11th Workshop Hot Topics Cloud Comput.*, 2019, pp. 1–7.

[19] H. Zhang, D. Liu, and Z. Xiong, "Two-stream action recognition-oriented video super-resolution," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 8798–8807.

[20] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1500–1513, Apr. 2023.

[21] *BridgeCam LIVE*. Accessed: Feb. 10, 2024. [Online]. Available: https://www.youtube.com/watch?v=rr-Ehp4lhmg

[22] *Sharx Security Demo Live Cam: Rotary Traffic Circle Derry NH USA*. Accessed: Feb. 10, 2024. [Online]. Available: https://www.youtube.com/watch?v=fuuBpBQElv4

[23] *YouTube Live Streaming*. Accessed: Feb. 10, 2024. [Online]. Available: https://www.youtube.com/live

[24] X. Dai, X. Kong, T. Guo, and Y. Huang, "CiNet: Redesigning deep neural networks for efficient mobile-cloud collaborative inference," in *Proc. SIAM Int. Conf. Data Mining*, 2021, pp. 459–467.

[25] J. R. Ding and J. F. Yang, "Adaptive group-of-pictures and scene change detection methods based on existing H. 264 advanced video coding information," *J. IET Image Process.*, vol. 2, no. 2, pp. 85–94 2008.

[26] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, pp. 107–194, 2011.

[27] *X265 HEVC Encoder*. Accessed: Mar. 22, 2024. [Online]. Available: https://bitbucket.org/multicoreware/x265_git.git

[28] *ITU-R Recommendation BT.601*. Accessed: Mar. 22, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Rec._601

[29] *YOLOv5*. Accessed: Mar. 22, 2024. [Online]. Available: https://github.com/ultralytics/yolov5

[30] S. S. Chaudhari and R. C. Biradar, "Survey of bandwidth estimation techniques in communication networks," *Wireless Pers. Commun.*, vol. 83, no. 2, pp. 1425–1476, Jul. 2015.

[31] J. Fang et al., "Reinforcement learning for bandwidth estimation and congestion control in real-time communications," 2019, *arXiv:1912.02222*.

[32] M. Lin, X. Zhang, Y. Tian, and Y. Huang, "Multi-signal detection framework: A deep learning based carrier frequency and bandwidth estimation," *Sensors*, vol. 22, no. 10, p. 3909, May 2022.

[33] *FCC Broadband Bandwidth Measurement*. Accessed: Mar. 22, 2024. [Online]. Available: https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-eighth

[34] R. Netravali et al., "Mahimahi: Accurate record-and-replay for HTTP," in *Proc. USENIX Annu. Tech. Conf. (USENIX)*, 2015, pp. 417–429.

[35] T. Y. H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. 13th ACM Conf. Embedded Networked Sensor Syst.*, 2015, pp. 155–168.

[36] S. P. Chinchali, E. Cidon, E. Pergament, T. Chu, and S. Katti, "Neural networks meet physical networks: Distributed inference between edge devices and the cloud," in *Proc. 17th ACM Workshop Hot Topics Netw.*, Nov. 2018, pp. 50–56.

[37] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, Aug. 2019, p. 116.

[38] C. Pakha, A. Chowdhery, and J. Jiang, "Reinventing video streaming for distributed vision analytics," in *Proc. 10th USENIX Workshop Hot Topics Cloud Comput.*, 2018, pp. 1–7.

[39] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 328–339.

[40] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "AWStream: Adaptive wide-area streaming analytics," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 236–252.

[41] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, Sep. 2015, pp. 426–438.

[42] K.-J. Hsu, K. Bhardwaj, and A. Gavrilovska, "COUPER: DNN model slicing for visual analytics containers at the edge," in *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, Nov. 2019, pp. 179–194.

[43] C.-C. Hung et al., "VideoEdge: Processing camera streams using hierarchical clusters," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 115–131.

[44] A. H. Jiang et al., "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 29–42.

[45] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Architectural Support Programming Languages Operating Syst.*, 2017, pp. 615–629.

[46] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.

[47] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the DNN black-box: Video analytics with DNNs across the camera-cloud boundary," in *Proc. Workshop Hot Topics Video Analytics Intell. Edges*, Oct. 2019, pp. 27–32.

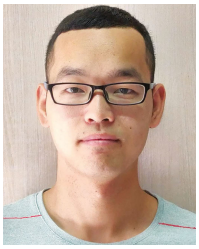
[48] R. LiKamWa and L. Zhong, "Starfish: Efficient concurrency support for computer vision applications," in *Proc. 13th Annu. Int. Conf. Mobile Syst., Appl., Services*, 2015, pp. 213–226.

[49] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. 14th USENIX Symp. Networked Syst. Design Implement*, 2017, pp. 377–392.

[50] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "DeepCache: Principled cache for mobile deep vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2018, pp. 129–144.

[51] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang, "Pano: Optimizing 360° video streaming with a better understanding of quality perception," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 394–407.

[52] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 253–266.
[53] N. Chen et al., "Cuttlefish: Neural configuration adaptation for video analysis in live augmented reality," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 4, pp. 830–841, Apr. 2021.
[54] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9241–9254, Oct. 2020.
[55] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "DeepSlicing: Collaborative and adaptive CNN inference with low latency," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2175–2187, Sep. 2021.
[56] J. Mao et al., "MeDNN: A distributed mobile system with enhanced partition and deployment for large-scale DNNs," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 751–756.
[57] D. Hu and B. Krishnamachari, "Fast and accurate streaming CNN inference via communication compression on the edge," in *Proc. IEEE/ACM 5th Int. Conf. Internet-Things Design Implement. (IoTDI)*, Apr. 2020, pp. 157–163.
[58] *AWS Wavelength: Bring AWS Services To the Edge of the Verizon 5G Network*. Accessed: Apr. 15, 2024. [Online]. Available: https://enterprise.verizon.com/business/learn/edge-computing/
[59] *Intel Xeon Scalable Processors*. Accessed: Apr. 15, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/xeon/scalable.html
[60] *Nvidia Egx A100: Delivering Real-Time AI Processing and Enhanced Security At the Edge*. Accessed: Apr. 15, 2024. [Online]. Available: https://www.nvidia.com/en-us/data-center/products/egx-a100/
[61] J. Yi, S. Kim, J. Kim, and S. Choi, "Supremo: Cloud-assisted low-latency super-resolution in mobile devices," *IEEE Trans. Mobile Comput.*, vol. 21, no. 5, pp. 1847–1860, May 2022.



Ning Chen received the Ph.D. degree from Nanjing University. He is currently a Lecturer with the School of Computer Science and Technology, Soochow University. He has published over 20 papers, including those appeared in INFOCOM, ICDE, IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, SECON, *Computer Networks*, and ICPADS. His research interests include edge computing, deep reinforcement learning, and video streaming.

Sheng Zhang (Senior Member, IEEE) received the B.S. and Ph.D. degrees from Nanjing University in 2008 and 2014, respectively. He is currently an Associate Professor with the Department of Computer Science and Technology, Nanjing University. He is also a member with the State Key Laboratory for Novel Software Technology. He has published more than 80 articles, including those appeared in IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, MobiHoc, ICDCS, and INFOCOM. His research interests include cloud computing and edge computing. He received the Best Paper Award of IEEE ICCCN 2020 and the Best Paper Runner-Up Award of IEEE MASS 2012. He was a recipient of the 2020 ACM Nanjing Rising Star Award and the 2015 ACM China Doctoral Dissertation Nomination Award. He is a Senior Member of CCF and a member of ACM.

Jie Wu (Fellow, IEEE) is currently the Director of the Center for Networked Computing and a Laura H. Carnell Professor with Temple University. He is also the Director of International Affairs with the College of Science and Technology. He was the Chair of the Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and the Associate Vice Provost of the International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was the Program Director of the National Science Foundation and was a Distinguished Professor with Florida Atlantic University. He regularly publishes in scholarly journals, conference proceedings, and books. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He was a recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He serves on several editorial boards, including IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, *Journal of Parallel and Distributed Computing*, and *Journal of Computer Science and Technology*. He was the General Co-Chair of IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, and the Program Co-Chair of IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, an ACM Distinguished Speaker, and the Chair of the IEEE Technical Committee on Distributed Processing (TCDP). He is a CCF Distinguished Speaker.

He Huang (Senior Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC), China, in 2011. He is currently a Professor with the School of Computer Science and Technology, Soochow University, China. From 2019 to 2020, he was a Visiting Research Scholar with Florida University, Gainesville. He has authored more than 100 papers in related international conference proceedings and journals. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a member of the Association for Computing Machinery (ACM). He received the Best Paper Award from Bigcom 2016, IEEE MSN 2018, and Bigcom 2018. He served as the Technical Program Committee Member for several conferences, including IEEE INFOCOM, IEEE MASS, IEEE ICC, and IEEE Globecom.

Sanglu Lu (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a Professor with the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published over 80 papers in referred journals and conferences in the above areas.