Edge-Assisted Real-Time Video Analytics With Spatial–Temporal Redundancy Suppression

Ziyi Wang, Xiaoyu He, Zhizhen Zhang, Yishuo Zhang, Zhen Cao, Wei Cheng, Wendong Wang, *Member, IEEE*, and Yong Cui, *Member, IEEE*

***Abstract*—Driven by plummeting camera prices and advances of video inference algorithms, video cameras are deployed ubiquitously and organizations usually rely on live video analytics to retrieve key information, such as the locations and identities of target objects. However, analyzing real-time video poses severe challenges to today's network and computation systems. To balance accuracy, bandwidth usage, and latency, we present EVA, an edge-assisted real-time video analytics framework, which coordinates computationally weak cameras with more powerful edge servers to enable video analytics under the accuracy and latency requirements of applications. EVA treats the region where a target object is located as a fine-grained transmission unit and exploits the redundancies in both spatial and temporal domains to reduce the bandwidth usage. Based on the framework, we design an adaptive offloading algorithm, which coordinates the recognition process between the camera and the server. To adapt to complex environments, we then design a threshold adjustment algorithm to tune the confidence threshold dynamically. Experiments on real-world video feeds show that compared to several recent baselines on multiple video genres, EVA maintains high accuracy while reducing bandwidth usage by up to 90%.**

***Index Terms*—Deep neural network (DNN), edge computing, spatial–temporal redundancy, video analytics.**

I. Introduction

THE PRIMARY goal for the development of the Internet of Things (IoT) is to create ubiquitous connection and access. Particularly, video cameras are gaining popularity in today's society, and there has been consistent growth in the scale and scope of their deployments. Millions of high-definition and network-connected cameras are employed at traffic intersections, retail stores, and remote industrial sites, for the purpose of traffic condition analytics and safety anomaly detection [1], [2], [3], [4], [5], [6], [7]. For example, counting volumes of cars, pedestrians, or bikes feeds into the traffic light controller to appropriately control the durations and manage traffic. These applications require latency below a second [2]. One key enabler for fast and accurate video inference is the rapid development of deep neural network (DNN), especially convolutional neural network (CNN) [8], [9], [10].

In a typical real-time video analytics pipeline, a camera streams live video to the edge server, which immediately runs an object recognition model to get analytic results about that video. In addition to obtaining the identity of the target object and the corresponding confidence score, the edge server can also obtain further information, such as the location and trajectory coordinates of the target object. Edge servers are usually purchased in advance and their network/computing resources need to be paid according to usage [11], [12], [13]. Therefore, reducing network and compute costs for edge servers is critical for organizations deploying video analytics services [14]. In addition, such pipelines aim to deliver the results with high accuracy, low latency, and low bandwidth usage. However, it is challenging to satisfy these three metrics at the same time because of their conflicting relationships. For example, transmitting a video at a higher bitrate from the camera to the edge server can improve the accuracy, but it can also result in the increase of transmission latency and network bandwidth usage.

Numerous studies have been presented to improve the efficiency of video analytics pipelines, which can be categorized into three groups.

1) Some researchers exploited the spatial redundancies and saved bandwidth usage by encoding each frame with a spatially uneven quality distribution (e.g., Region-of-Interest (RoI) encoding) [14], [15], [16], [17], [18]. However, they do not consider that the same target object might be unnecessarily transmitted multiple times in the temporal domain.
2) Some researchers exploited the temporal redundancies and filtered out frames that did not contain relevant information for the inference results [19], [20], [21], [22]. However, they do not consider that each frame contains some content (e.g., background) that does not affect the result in the spatial domain.
3) Some other researchers adapted configurations (e.g., frame rate, frame resolution, and deep learning model) to optimize a given goal [11], [12], [13], [23], [24],

Manuscript received 6 July 2022; revised 6 October 2022; accepted 21 November 2022. Date of publication 2 December 2022; date of current version 24 March 2023. This work was supported in part by NSFC Project under Grant 62072048, Grant 62072047, and Grant 62132009; and in part by China Unicom (Guangdong) Industrial Internet Company Ltd., Cooperation Project under Grant S2022041. *(Corresponding author: Yong Cui.)*

Ziyi Wang, Xiaoyu He, Zhizhen Zhang, and Yong Cui are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100089, China (e-mail: wangziyi0821@gmail.com; hexy21@mails.tsinghua.edu.cn; zzhizhenzzz@gmail.com; cuiyong@tsinghua.edu.cn).

Yishuo Zhang and Wendong Wang are with the School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: ZhangYishuo1996@outlook.com; wdwang@bupt.edu.cn).

Zhen Cao is with the Computer Network and Protocol Lab, Huawei Technologies Company Ltd., Beijing 100033, China (e-mail: zhen.cao@huawei.com).

Wei Cheng is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100089, China, and also with Unicom Industrial Internet Company Ltd. (Guangdong), Guangzhou 510320, China (e-mail: chengwei53@chinaunicom.cn).

Digital Object Identifier 10.1109/JIOT.2022.3224750

[25], [26]. However, they perform coarse-grained frame-level rather than fine-grained region-level configuration adjustment to make a tradeoff between accuracy, latency, and bandwidth.

In a word, these studies have optimized the video analytics pipelines from their own perspectives, which leads to suboptimal performance.

In this article, we propose EVA, an edge-assisted real-time video analytics framework, which coordinates computationally weak cameras with more powerful edge servers. The edge server can be any device that supplies the requisite computation capability. Unlike traditional video streaming that optimizes user-perceived visual quality, this new type of machine-centric video analytics permits aggressive spatial–temporal compression of irrelevant pixels to save bandwidth usage. For the spatial domain, we use the region where the target object is located instead of the whole frame as the basic transmission element (i.e., blacking out the areas other than the objects of interest). For the temporal domain, we only transmit the target object once when it is in its best position by using temporal correlation in video analytics.

However, it is challenging to make offloading decisions based on this framework. First, although the spatial–temporal redundancy suppression greatly reduces bandwidth consumption, it introduces a new problem to the recognition process: how to select the best region for each object in the temporal domain in order to meet the requirements of accuracy and latency simultaneously. For instance, as an oncoming object is getting closer to the camera, its captured pixels will increase, which results in a higher recognition accuracy. In the meantime, limited by the constraint of the end-to-end latency, we have to transfer the region at a lower bitrate, leading to a decrease in accuracy. Second, both video content and network conditions are complex and changeable, making it a challenge to adaptively adjust the offloading decision. On the one hand, the strategies for identifying objects in well-illuminated and dimly lit environments should be different. On the other hand, available bandwidth and network latency are constantly changing. Therefore, the decision needs to be adaptively adjusted for different conditions.

To properly consider the tradeoff between accuracy and latency, we first estimate their values on the camera side. For accuracy estimation, we fully utilize the preprocessing information (e.g., the locations and confidence scores of objects) from the camera detector to judge how confident it is to recognize a target object. We also estimate the total end-to-end latency according to current network latency, transmission frequency, and waiting latency. Then, according to the accuracy and latency requirements of applications, local recognition results are transmitted to the server as soon as their confidence scores are high enough—in other words, the objects are easy to be detected by the camera detector. For those unconfident objects, on the other hand, the camera sends their best regions (i.e., the regions with the highest confidence score) to the server for recognition when they are about to exceed the latency constraint. To make our system more adaptive to complex and ever-changing environments, the confidence threshold is dynamically tuned. Periodically, we examine the correctness of camera-side detected results using the high-accuracy model on the server side. Then, the feedback from the server is sent to guide the camera detector to tune the threshold accordingly.

The main contributions of this article can be summarized as follows.

1) We develop a fine-grained region-based cooperative video analytics framework which treats the object-located region as the basic transmission element and transmits each target object only once. The framework utilizes the redundancy of both spatial and temporal domains to achieve significant bandwidth saving.
2) We design two key algorithms in the framework. Under the latency and accuracy requirements of applications, the proposed offloading algorithm coordinates the recognition process between camera and server. Meanwhile, the adjustment algorithm dynamically revises the confidence threshold to adapt the system to complicated environmental factors, such as illumination and obstruction.
3) We implement and evaluate a system consisting of the camera side and the server side. Experimental results with real-world video data sets show that compared to several recent baselines, EVA reduces significant bandwidth usage by up to 90%, while consistently meeting the desired accuracy requirement. Its end-to-end latency can also meet the needs of real-time video analytics.

The remainder of this article is structured as follows. Section II presents the cooperative framework. Section III elaborates the details of two algorithms. Section IV describes the implementation details. Experimental results are presented in Section V. Section VI discusses the related work. Finally, we conclude this article in Section VII.

II. Edge-Assisted Video Analytics Framework

In this section, an overview of our framework, namely, EVA, is given first. Then, we will elaborate three key components of the framework, including object recognition (tiny and big model), offloading controller, and threshold adjuster. Finally, we discuss the application of the framework.

A. Framework Overview

As shown in Fig. 1, the framework consists of two parts: the camera side and the server side. After the camera obtains real-time video data, a tiny CNN model is run locally for rudimentary object recognition. Then, the preliminary recognition results are delivered in two paths. The offloading path indicated by solid lines represents our routine recognition workflow, while the adjustment path indicated by lines of dots is executed periodically to make the system more robust and efficient to the impact of the variational external environment. In the offloading path, each object is sent only once. The offloading controller is responsible for deciding to transmit either results of confident objects right away, or regions of unconfident objects at their best recognition location. For the latter, the server runs a more powerful model for further recognition. In the adjustment path, our system periodically feeds the filtered objects back to the threshold adjuster on the server

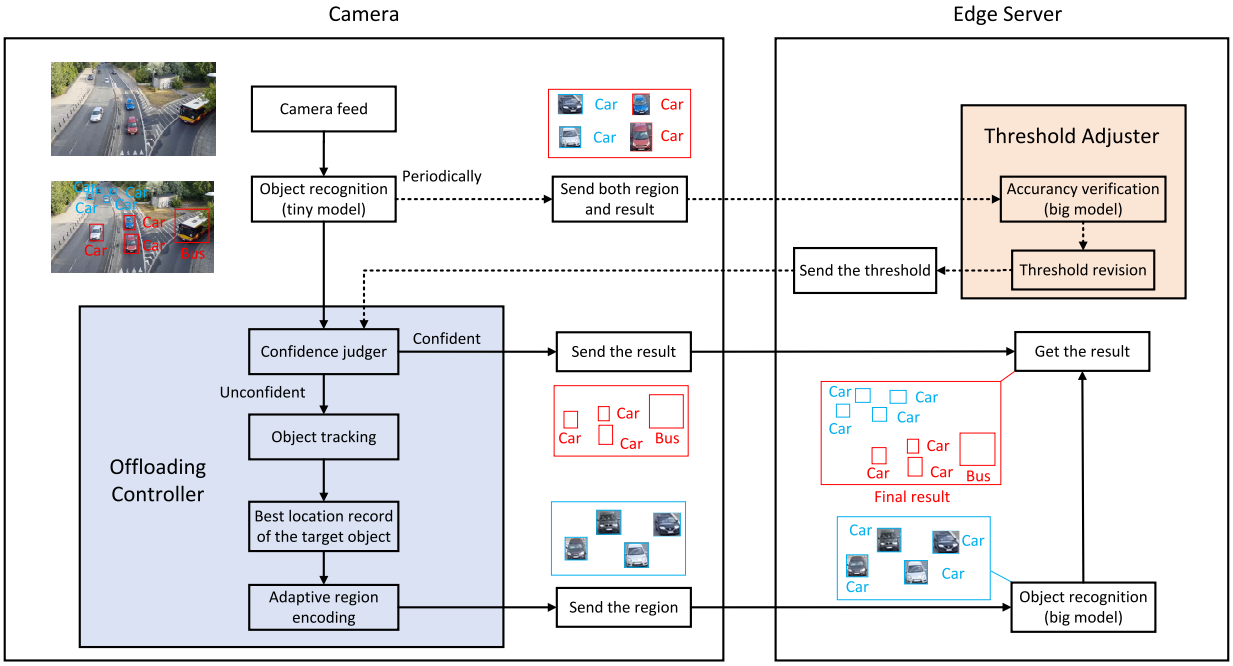

Fig. 1. Edge-assisted real-time video analytics framework with spatial–temporal redundancy suppression.

side. After careful examination and revision, the server then sends the new tuned threshold back to camera-side offloading controller.

The framework makes full use of the computing capacity of both the camera and the server. They cooperate to obtain object recognition results under the constraints of accuracy and latency. In addition, this framework reduces network bandwidth usage. From the perspective of the spatial domain, the framework only transmits the region of the target object, and does not consider the other parts (e.g., background) that have no effect on the recognition accuracy, which greatly reduces the redundant information in the spatial domain. Moreover, those easy-to-identify objects are directly sent in the form of recognition labels, which decreases both end-to-end latency and bandwidth usage. From the perspective of temporal domain, target objects are transmitted only when they are in their best locations, instead of being repeatedly sent in each frame, which greatly reduces the redundant information in the temporal domain without sacrificing the recognition accuracy. Finally, preliminary results generated by the camera detector are tested at regular intervals, which makes our system more intelligent and adaptive to the ever-changing scenarios.

B. Object Recognition

We use a tiny and a big CNN model for object recognition on the camera side and server side, respectively. Because the inference of state-of-the-art CNN is computationally expensive (and slow), it is uneconomical to run a heavyweight high-accuracy model on the camera for video analytics. The compression technique has been developed to reduce the cost of inference. For example, Tiny YOLO [27], a shallower variant of the YOLO object detector, is 5× cheaper than YOLOv2. The tradeoff is that compressed CNN is usually less accurate than the original CNN. Therefore, we design such a collaborative recognition mechanism based on the different computing capabilities of the camera and the server. A lightweight model is run on the camera side for preliminary recognition. If the local detector is quite confident about current detection results, it will directly send the results to the server. Otherwise, the target objects will be transferred for further recognition using a powerful model on the server side.

C. Offloading Controller

The controller decides whether to directly transmit the preliminary recognition results or continue to track objects waiting for the best opportunity to send them for further recognition based on application requirements, such as accuracy and latency. Comprehensive offloading algorithm design is the focus of the next section. The following are the descriptions of the key procedures in the controller.

Object Tracking and Best Location Record: We utilize the preprocessed information of the camera-side CNN model, including detected bounding boxes and confidence scores, to record the best location of an object. For each target object, if the Intersection over Union (IoU) [21] of the two consecutive recognized bounding boxes exceeds a given threshold, it will be considered as a successful tracking. In other words, the target is still in the scene. Then, we will update the record result to the region that has a higher confidence score. If the latency limit set by the application is about to be exceeded, the best region of the target object will be sent immediately. Otherwise, we will continue to wait for a new update.

Adaptive Region Encoding: We improve bandwidth efficiency by sending only a few regions in adaptive quality to achieve the same recognition accuracy as if the whole video is sent in the highest quality. For the target object which the

controller decides to send for further recognition, we dynamically decide the sending bitrate of its best region according to the current network conditions and the latency requirement of the application. These regions are encoded as images. In fact, we are going a step further based on RoI encoding [17], [28], which allows the user to adjust the encoding quality for each macroblock in a frame and does not remove the background. For computer vision tasks, our method uses the region where the target object is located instead of the whole frame as the basic transmission element (i.e., blacking out the areas other than the objects of interest).

D. Threshold Adjuster

The threshold adjuster selects the appropriate confidence threshold for the offloading controller to filter those easy-to-identify objects. This is important, because if the threshold is set too high, it may sacrifice the benefits of local detection (i.e., decrease in bandwidth usage and latency) by unnecessarily sending objects that could have been correctly recognized on the camera side. Meanwhile, a too low threshold will be harmful to recognition accuracy because an actually unconfident object might be mistaken for a confident one. However, such selection can be difficult, for optimal threshold varies rapidly as the scenarios change from time to time, which is hard to be clearly expressed due to the complex relations between background factors and recognition accuracy.

To overcome the difficulty, the threshold adjuster is designed to periodically check the correctness of camera-side recognition results of the filtered objects, whose confidence scores are close to the confidence threshold, by using a heavyweight recognition model on the server side. The confidence threshold is tuned accordingly and transferred to guide the offloading controller to distinguish confident targets from unconfident objects more precisely.

E. Application Discussion

In fact, this framework can be applied to many application scenarios. In addition to obtaining the identity of the target object and the corresponding confidence score, the framework can also obtain further information by replacing different models according to the needs of the application, such as the location and trajectory coordinates of the target object. Furthermore, today's cameras usually retain video for a period of time to their local storage. For some situations that require further detailed analysis, the server can then request the camera to transmit the complete video within a specific period of time.

In industry practice, DNN models are heavily tuned and customized to make them more specialized for the tasks and resource-efficient for the devices. In fact, model optimization and framework optimization are two different dimensions. As the model becomes more efficient, the computational latency will be reduced. Combined with our framework, it yields even greater performance benefits. In addition, state-of-the-art cameras have more resources than the commodity cameras which are widely deployed [19]. However, we do not anticipate a fast update that switches out commodity cameras for state-of-the-art ones because large-scale camera deployments are financially costly to install and maintain. Instead, we expect a more gradual shift and, thus, believe that camera-edge framework will be valuable for a long time to come. Even if all cameras are equipped with powerful AI chips and have powerful computing power in the future, the idea of spatial–temporal redundancy suppression can still help select crucial information for processing, thereby saving computing load. This allows the camera to avoid redundant computations so that other tasks can be performed as well.

Furthermore, our framework can be generalized to other computer vision tasks such as semantic segmentation. The difference is that object detection is based on bounding boxes, while semantic segmentation is based on pixels. Semantic segmentation DNNs assign each pixel a class label and a score for each class (the class with the highest score is the class label). In the beginning, we may give each pixel a score of 1+max'-max, where max is the highest score among the classes of interest and max' is the second highest. Evidently, the higher the score is, the more indecisive the DNN is about which class a pixel belongs to. We could pick the unconfident regions by creating some rectangles that cover as many high-score pixels as possible. Then, we can send these unconfident regions to the server side for further processing. In this way, our framework can be generalized to other computer vision tasks.

III. Algorithm Design

In this section, we first give the problem formulation. Then, we introduce the offloading control algorithm (denoted as Algorithm 1), containing a detailed description of our treatments for three different types of detected objects (distinguished by their occurrence continuity). The confidence threshold adjustment algorithm (denoted as Algorithm 2) is described as a supplement, which is designed to make our system more efficient and robust in the face of complicated and unstable environmental factors, such as illumination and obstruction.

A. Problem Formulation

Although the spatial–temporal redundancy suppression greatly reduces bandwidth consumption, it introduces a new problem to the recognition process: how to select the best region for each object in the temporal domain in order to meet the requirements of accuracy and latency simultaneously. As each target object appears, moves and finally leaves the camera range, it will be captured repeatedly in a series of consecutive frames by the camera. We number the frames in which the target object appears from 1 to W, and use RoI_i (RoI) to represent the bounding box where the target object is located in the ith frame.

When selecting the best region for the target object, we aim at achieving desirable analytics accuracy under the latency constraint. For simplicity of illustration, we use a_i and l_i to denote the accuracy and latency of RoI_i. The natural objective

Algorithm 1 Offloading Control Algorithm

Input: D_n : *the* $n-th$ *detected results*, r : *video bitrate*, R : *maximum video bitrate*, f : *transmission frequency*, L : *network latency*, B : *network bandwidth*, T : *latency threshold*, C : *confidence threshold*

Output: S : *send decision*

1: **for** $n \in 1$ *to* N **do**
2: **for** *object* $\in D_n$ **do**
3: **if** (*object.send* $==$ *false and object.confidence* $\geq C$) **then**
4: *add object.result to S*
5: *object.send* $\leftarrow$ *true*
6: *Count* $D_{n-1} \cap D_n$
7: **for** *object* $\in D_{n-1} \cap D_n$ **do**
8: **if** (*object.send* $==$ *false*) **then**
9: **if** $(\frac{R}{f \cdot B} + L + object.time \geq T)$ **then**
10: $r \leftarrow f \cdot B \cdot (T - L - object.time)$
11: *add* (*object.region*, r) *to S*
12: **else if** ($\Delta object.confidence < 0$) **then**
13: *add* (*object.region*, R) *to S*
14: *update object in* D_n
15: **for** *object* $\in D_{n-1} - D_n$ **do**
16: **if** (*object.send* $==$ *false*) **then**
17: $r \leftarrow \min(R, f \cdot B \cdot (T - L - object.time))$
18: *add* (*object.region*, r) *to S*
19: **return** S

is the maximum of the accuracy, which can be formulated as

$$\max_{\mathrm{RoI}_i} \ a_i \tag{1}$$

$$\text{s.t.} \ \ a_i = \mathrm{RoI}_i.\text{confidence} \tag{2}$$

$$l_i = \frac{\mathrm{RoI}_i.\text{size}}{B} + L + \mathrm{RoI}_i.\text{time} \tag{3}$$

$$l_i \leq T. \tag{4}$$

Equation (2) illustrates that the confidence value fed back by the recognition model can be used to estimate the accuracy of the RoI. We use B and L to denote current network bandwidth and latency. Equation (3) shows the composition of end-to-end latency, including sending latency ($\mathrm{RoI}_i.\text{size}/B$), transmission latency (L) and waiting latency ($\mathrm{RoI}_i.\text{time}$). Equation (4) says that the end-to-end latency cannot exceed the target value T. The goal is to find the RoI that maximizes the accuracy among all RoIs under the latency constraint.

B. Offloading Control Algorithm

The offloading control algorithm is designed to coordinate the recognition process between the camera and the server under the latency and accuracy requirements of applications. The key idea is to make full use of the preliminary recognition on the camera side. We use the confidence score of local recognitions to measure the goodness between two sequential regions. On the one hand, detection results are transmitted immediately to the server when the local detector is confident enough about it. On the other hand, Algorithm 1 sends the best region of unconfident objects to the server for further recognition. To properly consider the tradeoff between latency and accuracy, we calculate the total end-to-end latency of each recognition process, and send the bounding box of an object at the possible maximum bitrate as soon as the latency approaches the threshold set by applications. We will discuss the algorithm in detail in the following parts.

First, for those easy-to-identify objects, they are highly likely to be correctly recognized by the local camera detector. We set a confidence threshold (denoted as C) for Algorithm 1 to decide whether it is positive enough about its detection results. The process corresponds to lines 2–5: for each yet-to-be-sent object (i.e., object.send == false) in the current detected results, if its confidence score is above C, Algorithm 1 will send the object's result (denoted as object.result) to the server right away.

Then Algorithm 1 seeks the highest confidence score in sequence for those objects which continually appear. Thus, each sent decision will be made based on the results of current and last local detection. We use N to denote the total number of local detection. For the nth ($1 \leq n \leq N$) local detection, D_n denotes the set of its results, then D_{n-1} denotes the set of its last detected results, and $D_{n-1} \cap D_n$ indicates the set of objects that continually appear. The intersection can be calculated by using the IoU metric [21], similar to $IoU = (A \cap B / A \cup B)$, where A and B are the bounding boxes of the last detection and the current detection, respectively. After that, Algorithm 1 estimates the end-to-end latency of each unconfident object in $D_{n-1} \cap D_n$ to weigh the risk of a timeout, which includes the transmission latency and the waiting latency. We use L, B, f, and R to denote network latency, network bandwidth, transmission frequency, and maximum video bitrate, respectively. Then, we can calculate the maximum transmission latency as $(R/f \cdot B) + L$. Waiting latency could be easily expressed as the current time minus the first time when the object appeared (i.e., object.time). If the total latency of an object is greater than the latency threshold (denoted as T), Algorithm 1 will transmit the best region at the bitrate of $f \cdot B \cdot (T - L - \text{object.time})$, which is precisely the maximum allowed bitrate under the latency requirement (lines 8–11). Otherwise, the object will be checked to see if its confidence score declines over time. If so, Algorithm 1 will immediately send the object's region to the server at maximum bitrate R (line 12 to 13). This is to identify objects moving farther away from the camera, which would compromise both accuracy and latency if we wait to see whether there will be a better region to be recognized. Then, at line 14, object.time and object.send are updated in the loop.

Finally, Algorithm 1 transmits all the regions of disappearing and yet-to-be-sent objects in D_{n-1}, which can be expressed as $D_{n-1} - D_n$. The transmission bitrate is determined by the following two evaluations. If the total latency of an object approaches T, its sending bitrate is supposed to be $f \cdot B \cdot (T - L - \text{object.time})$ as we discussed above. Otherwise, Algorithm 1 should send the region at R to maximize accuracy. As can be seen in line 17, both evaluation can be unified as one expression $\min(R, f \cdot B \cdot (T - L - \text{object.time}))$.

Algorithm 2 Confidence Threshold Adjustment Algorithm

Input: $C_{current}$: *current confidence threshold*,
D_{above} : *objects whose confidence is above* $C_{current}$,
D_{below} : *objects whose confidence is below* $C_{current}$,
ΔC : *adjustment interval of confidence threshold*,
num : *number of successful detections*,
A : *accuracy target*
Output: C_{new} : *new confidence threshold*

$num \leftarrow 0$
for $object \in D_{above}$ **do**
  $result \leftarrow model.recognize(object.region)$
  if $(object.result == result)$ **then**
    $num++$
if $(\frac{num}{|D_{above}|} \leq A)$ **then**
  return $C_{current} + \lambda \cdot \Delta C$
$num \leftarrow 0$
for $object \in D_{below}$ **do**
  $result \leftarrow model.recognize(object.region)$
  if $(object.result == result)$ **then**
    $num++$
if $(\frac{num}{|D_{below}|} \geq A)$ **then**
  return $C_{current} - \Delta C$
return $C_{current}$

We use N to denote the number of local detection, and M to denote the average number of target objects contained in each detection, then the time complexity of Algorithm 1 is $O(N \times M)$.

C. Threshold Adjustment Algorithm

The threshold adjustment algorithm adaptively adjusts the confidence threshold to guarantee the performance of our system in the face of complex and volatile circumstances. The main idea is to periodically check the correctness of locally detected results whose confidence score is around the current confidence threshold (denoted as C_{current}). Algorithm 2 uses the recognition results from the server as the ground truth and, therefore, requires that the camera detector sends both regions and detected results of certain objects.

To ensure the correctness of confident results detected by the camera-side detector, we first examine the detected objects whose confidence scores are slightly above C_{current} (denoted as D_{above}). For each region, Algorithm 2 runs a CNN model on the server to get the ground truth. Then, a detection will be considered successful if the object's result is consistent with the ground truth. We use num to denote the number of successful detection and $|D_{\text{above}}|$ denotes the total number of confident detection, then the accuracy of D_{above} should be $(\text{num}/|D_{\text{above}}|)$. The accuracy is supposed to be at least equal to the accuracy target (denoted as A) required by applications. If not, the new confidence threshold (denoted as C_{new}) will be revised upward. The adjustment interval (denoted as ΔC) and its coefficient λ can be heuristically obtained.

On the other hand, if the confidence threshold is set too high, considerable objects which could have been recognized locally will be sent to the server, causing the waste of both bandwidth usage and computing resources. To avoid this, we then check the detected objects whose confidence scores are slightly below C_{current} (denoted as D_{below}). Similar to the examination of D_{above}, we can calculate the accuracy of D_{below}. If such accuracy exceeds A, C_{new} will be lowered. It is noteworthy that the adjustment interval of threshold decrease is narrower than that of increase, because achieving the requirement of applications is way more important than saving network bandwidth.

We use K to denote the number of target objects with confidence scores near the current threshold, then the time complexity of Algorithm 2 is $O(K)$.

IV. Implementation

We implement an end-to-end system mostly in Python and it consists of around 2000 lines of code. The system includes a camera side and a server side.

Camera Side: We implement camera-side functions on virtual machine (VM) for emulation and then migrate the system to an Nvidia Jetson intelligent toy car [29] for realistic evaluation. As can be seen in the camera part of our design flow (i.e., Fig. 1), three key modules need to be implemented: 1) object recognition module; 2) tracking module; and 3) data transmission module. For the object recognition module, in order to reduce inference latency, we use Nvidia TensorRT [30] to optimize the deep learning model. For the object tracking module, to ensure that each object is only sent once, we first create a cache to store objects that appear in the current frame, and then implement a `trace()` function to select the objects that appear in both the current frame and the cache according to the IoU of two objects' bounding boxes. For the data transmission module, multimedia streaming protocols (e.g., RTP) aim to be fast instead of reliable. While they can achieve low latency, their accuracy can be poor under congestion. Recent work has moved toward HTTP-based protocols and focused on designing adaptation strategy to improve the performance, as in research (Pensieve [31], CS2P [32]) and industry (HLS [33], DASH [34]). What we transmit is the key information obtained after preprocessing on the camera, such as the preliminary recognition result or the best region of the target object, so we use the reliable protocol HTTP protocol and create a request session to send a POST request to the server side. When the regions of low-confidence objects are ready to be transmitted, we tune the `IMWRITE_PNG_COMPRESSION` parameter in `imwrite()` function of OpenCV to adaptively encode our image data.

Server Side: The implementation of the server side consists of three main functions: 1) object detection; 2) data transmission; and 3) confidence threshold adjustment. The detection function is similar to the camera side, and the transmission function is implemented through the Flask web framework [35]. Both functions can be executed in parallel by our system for the majority of the time. However, when the server needs to check the correctness of the camera detector due to a change in brightness or other factors, we must perform object detection, confidence threshold adjustment, and

data transmission sequentially. In this way, we ensure that the camera side can be regulated in a timely manner in accordance with the new confidence threshold. In our current implementation, periodic threshold adjustments are made whenever 500 target objects with confidence scores near the current threshold are collected. This value can be adjusted according to the actual usage scenario.

V. Evaluation

In this section, we evaluate the performance of our system in comparison with several recent baselines. The server is equipped with an octa-core Intel processor running 3.3 GHz with 16 GB of RAM and an NVIDIA GeForce GTX3060 graphics card with 6 GB of RAM. According to the previous study [19], we use a VM with limited computing power to simulate the camera side. The resource configuration of the simulated VM is 4 GB of RAM and a 1-GHz CPU. We implement the algorithm on the camera side and the server side, respectively. For the CNN model, Tiny YOLO [27] and FasterRCNN-ResNet101 [36] are run on the camera side and server side, respectively. We use two data sets provided in [14]. The traffic data set contains seven videos with a total duration of 2331s. The drone data 3set contains 13 videos with a total duration of 163 s. The videos are fed into the camera sequentially and in real-time. We make the source code publicly accessible.[1]

A. End-to-End Improvements

We start with EVA's overall performance gains over the baselines. The performance metrics we evaluate mainly include the following.

1) *Accuracy:* We measure the accuracy using the $F1$ score which is the harmonic mean of precision and recall for the detected objects' locations and class labels. The ground truth is obtained by running server-side CNN model on the original (highest quality) video. In this way, we can reveal any negative impact of video compression and streaming on CNN inference.
2) *Bandwidth Usage:* We measure the bandwidth usage by the size of the sent data divided by its duration. Different video content will have different bandwidth usage. To avoid impact of video content on bandwidth usage, we report bandwidth usages of our algorithm and the baselines after normalizing them against the bandwidth usage of each original video. In general, the total cost of a video analytics system includes the camera cost, the network cost paid to stream the video, and the cost of the server. Since the camera and the server are purchased upfront, their costs will approach zero in the long run, but the network cost is paid by time. Thus, we focus on reducing the network cost through reducing bandwidth usage.
3) *End-to-End Latency:* We measure the end-to-end latency by counting the time between when an object first appears in the video and when its region is correctly recognized, which includes the processing latency of the camera and the server as well as the transmission latency.

The baseline comparison systems include the following.

1) *DDS [14]:* It reduces bandwidth usage by eliminating spatial domain redundancy. It filters out the parts of the given frame that do not affect the recognition result (e.g., background), but does not consider that the same target object is transmitted multiple times in the temporal domain.
2) *Reducto [19]:* It reduces bandwidth usage by eliminating temporal domain redundancy. It filters out certain frames that do not affect the recognition result, but does not consider that each frame contains some content that does not affect the result in the spatial domain.
3) *Camera-Only:* It only executes the tiny CNN model locally on the camera side for recognition.
4) *Server-Only:* It compresses the videos and offloads all frames to the server side for recognition after the camera captures the video.

Fig. 2 compares the inference accuracy, bandwidth consumption and end-to-end latency of *EVA* with those of the baselines. We normalize the bandwidth usage against the size of the highest-quality videos which we use to get the ground truth. Fig. 2(a)–(c) shows the performance of these systems on the traffic data set. From the figure, we can find that *Server-only* has the highest average accuracy of 98.60%. This is because all videos are transmitted to the server for high-accuracy recognition. As a result, its bandwidth consumption is also the highest, with an average normalized bandwidth of 0.97. Due to the relatively large transmission latency caused by the full video transmission, its average end-to-end latency is also the highest. *DDS* and *Reducto* reduce bandwidth consumption from the perspective of spatial and temporal domain, respectively. It can be found from the figure that these two systems achieve relatively high accuracy (98.05% for *DDS* and 98.27% for *Reducto*). However, they only save bandwidth from a single dimension, which leads to their relatively high bandwidth consumption and end-to-end latency. The average normalized bandwidth of *DDS* is 0.52, while that of *Reducto* is 0.80. In addition, *Camera-only* executes all jobs locally and does not consume any bandwidth resources, resulting in the lowest accuracy and end-to-end latency. Compared with them, our system *EVA* occupies very little transmission bandwidth of 0.08 to ensure a high average accuracy of 97.10%. The reason is that *EVA* removes redundancy from both spatial and temporal domains, which can greatly reduce bandwidth usage without sacrificing much recognition accuracy. Its end-to-end latency is the second lowest, which can well meet the real-time requirement.

Fig. 2(d)–(f) shows the performance of these systems on the drone data set. *Server-only* also has the highest average accuracy of 98.00%. In addition, it has the highest bandwidth consumption and end-to-end latency. *DDS* achieves spatial optimization, as its normalized bandwidth consumption is reduced to 0.45 and its average accuracy is 96.18%. Similarly, *Reducto* achieves temporal optimization. Its normalized bandwidth consumption is 0.72, which is still relatively high, and

[1] https://github.com/STAR-Tsinghua/EVA

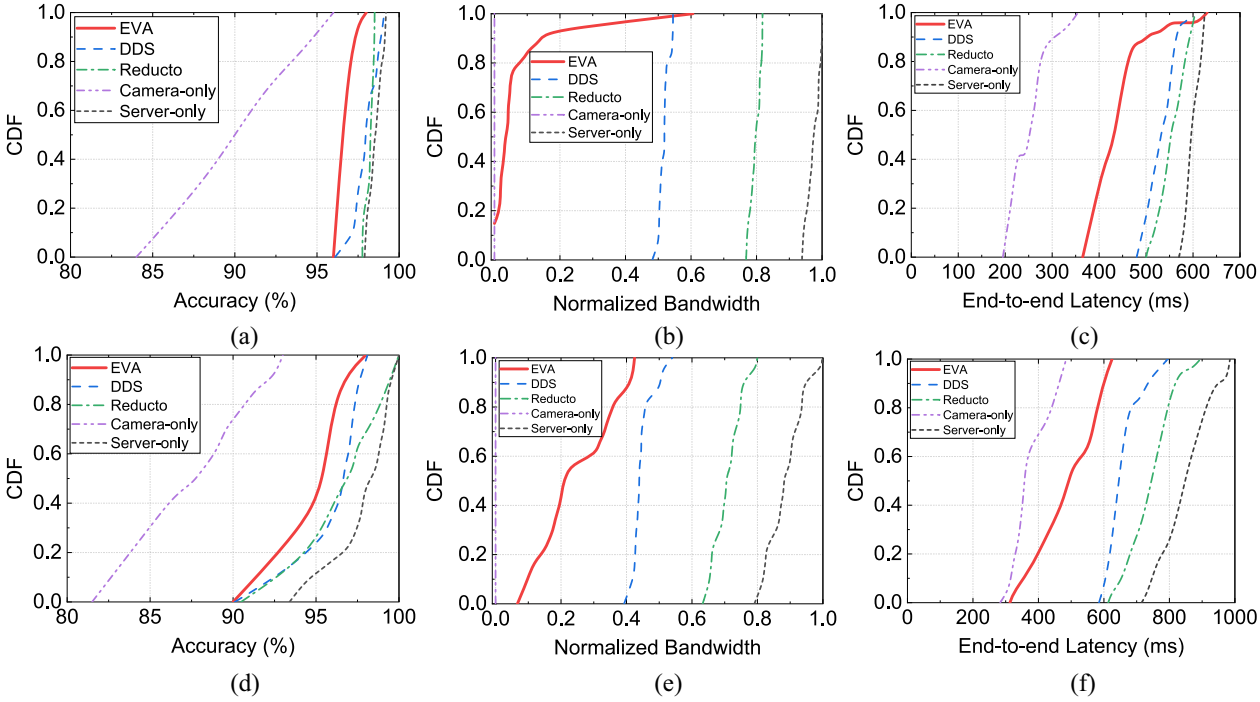

Fig. 2. Inference accuracy, normalized bandwidth consumption and end-to-end latency of EVA and several baselines on traffic and drone data sets. (a) Inference accuracy performance on traffic data set. (b) Normalized bandwidth consumption on the traffic data set. (c) End-to-end latency on the traffic data set. (d) Inference accuracy performance on the drone data set. (e) Normalized bandwidth consumption on the drone data set. (f) End-to-end latency on the drone data set.

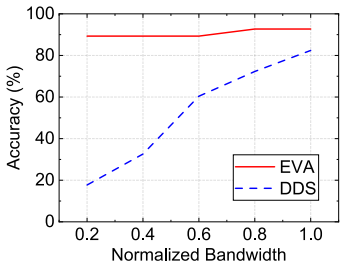

Fig. 3. EVA outperforms DDS in accuracy under various bandwidth consumption budgets.

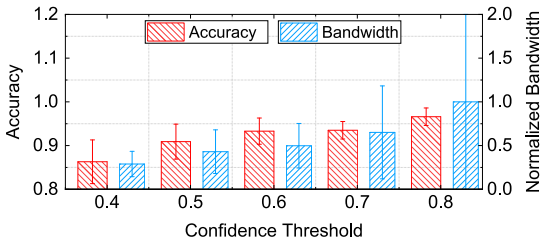

Fig. 4. Impact of confidence threshold on accuracy and bandwidth.

its average accuracy is 96.94%. Although *Camera-only* consumes no bandwidth resources and has the lowest end-to-end latency, its accuracy is the worst, only 87.76%. Our system *EVA* can still maintain an accuracy of 94.76% under the condition that the normalized bandwidth consumption is 0.25. Although it requires co-processing between the camera and the server, it achieves the second lowest end-to-end latency because it greatly reduces the transmission latency.

It can be found that the performances of these systems on the traffic and drone data sets are similar. The results demonstrate that *EVA* outperforms several baseline systems. It maintains high accuracy while reducing bandwidth usage by up to 90%.

B. Impact of Network Settings

We alter the available bandwidth between the camera side and the server side, and compare the accuracy of *EVA* and *DDS*. Fig. 3 shows that *EVA* outperforms *DDS* in accuracy under various bandwidth consumption budgets. This is because *EVA* saves bandwidth from both the spatial and temporal domains, which makes its required bandwidth much smaller than that of *DDS*. Therefore, when we limit the normalized bandwidth to 0.2, *EVA* manages to transmit all data it needs, and its accuracy can reach up to 89.30%. However, the available bandwidth at this time cannot meet the needs of *DDS*, and its accuracy is as low as 17.70%. As we increase the available bandwidth, the accuracy of *EVA* fluctuates around 90%, while the accuracy of *DDS* rises from around 20% to around 80%. It can be seen that when the available bandwidth is limited under a certain threshold, *DDS* cannot guarantee the accuracy while *EVA* still works well by virtue of its advantages of removing redundancy and saving bandwidth.

C. Impact of Confidence Threshold

The confidence threshold is a significant parameter in *EVA*. In this part, we first explore the impact of statically setting different thresholds on accuracy and bandwidth consumption. As shown in Fig. 4, when the confidence threshold is adjusted

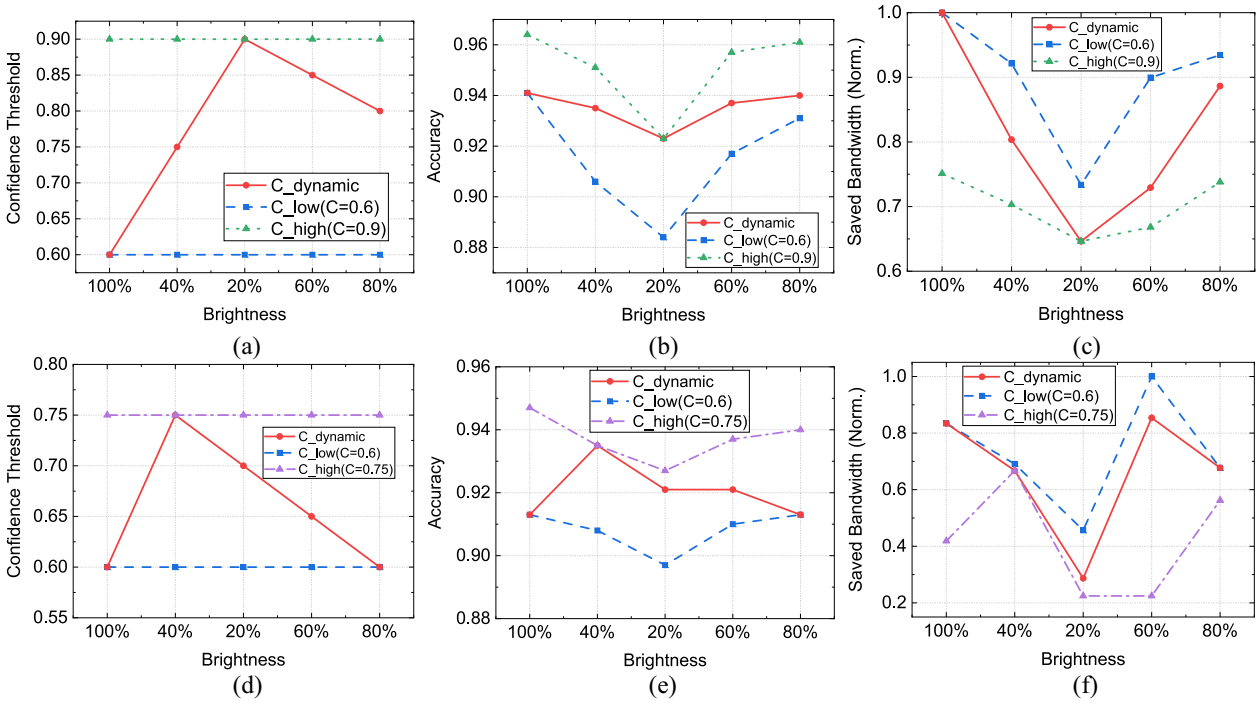

Fig. 5. Confidence threshold, inference accuracy, and saved bandwidth of EVA and two fixed-threshold solutions on traffic and drone data sets. (a) Confidence threshold on the traffic data set. (b) Inference accuracy on the traffic data set. (c) Saved bandwidth on traffic data set. (d) Confidence threshold on the drone data set. (e) Inference accuracy on the drone data set. (f) Saved bandwidth on the drone data set.

from 0.4 to 0.8, the average accuracy of *EVA* rises from 86.30% to 96.60%. At the same time, the average normalized bandwidth consumed also increases from 0.29 to 1. In essence, this is a tradeoff relationship. When the confidence threshold is low, the camera side will consider most of the local recognition results as confident results, and send them directly to the server without further recognition. In other words, more objects are identified preliminarily by the camera detector other than a more powerful detector on the server side, which leads to lower accuracy of the entire system. Meanwhile, it reduces bandwidth usage since fewer regions of unconfident objects are transmitted. On the other hand, when the confidence threshold is tuned higher, the camera is more cautious and more regions of objects, whose confidence is lower than the threshold, will be transmitted to the server for further recognition. Although this improves the accuracy of the system, the corresponding bandwidth consumption also increases.

Since the confidence threshold is a significant parameter in *EVA*, we propose Algorithm 2 to adaptively select the proper value under the accuracy requirement of the application. To better understand the robustness of our algorithm, we now evaluate the performance of the algorithm for dynamic adjustment on two different data sets (i.e., traffic data set and drone data set) in comparison with two fixed solutions whose threshold is the upper and lower bound of ours, respectively. We measure the accuracy and saved bandwidth which is normalized as in previous experiments. The target accuracy set by the application is 90%. To simulate the change of environment in a day, we tune the brightness of tested videos at each time point, whose brightness is 100%, 40%, 20%, 60% and 80% of the original one in sequence.

As shown in Fig. 5(a), two fixed-threshold solutions are indicated by the dotted line (confidence threshold = 0.90, denoted as C_{high}) and dashed line (confidence threshold = 0.60, denoted as C_{low}) severally. When we reduce the brightness by 60% and 20% in succession, it adds difficulty to the preliminary local recognition and, thus, makes the results of the sampling objects unqualified. Therefore, our algorithm dynamically tunes the confidence threshold upward (15% higher at a time) accordingly. In addition, with the increase by 40% and 20% in sequence of the brightness, the threshold is adjusted 5% lower each time. This is consistent with our design idea that the algorithm values accuracy more than bandwidth saving. As shown in Fig. 5(b) and (c), the accuracy and saved bandwidth of our algorithm are the same as C_{low} at first since they share the same threshold. Meanwhile, C_{high} occupies about 25% more bandwidth in exchange for 2% higher accuracy, which is unnecessary because the requirement of accuracy has already been met. Then, when the brightness is reduced gradually, we can see that our algorithm remains high-accuracy similar to C_{high} at the expense of bandwidth usage, while the accuracy of C_{low} falls to about 88%, which violates the accuracy requirement of the application. After that, as the brightness increases, each accuracy continues an upward swing, and the saved bandwidth of our algorithm is approaching the C_{low} line again. This is consistent with the intuition that when target objects are easy to identify, rather than pursuing higher accuracy, our algorithm instead slightly lowers the threshold to save the bandwidth usage.

Fig. 5(d)–(f) shows that our algorithm is able to save up to 60% more bandwidth on the drone data set, while consistently meeting the desired accuracy. Only when the brightness

Fig. 6. We implement our algorithm on Nvidia Jetson intelligent toy car.

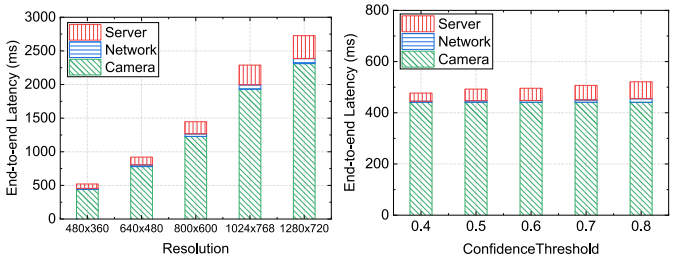

Fig. 7. End-to-end latency performance under different resolutions and confidence thresholds.

is reduced to 20% of the original one, which makes the target objects indiscernible to recognize, does our algorithm decide to occupy more bandwidth in order to meet the accuracy requirement. For comparison, we can see that the solution whose threshold remains 0.6 (indicated by the dashed line) falls short of the demand of accuracy as the brightness turns to 20%. The solution with a fixed threshold of 0.75 (indicated by the dashed–dotted line) is far more bandwidth consuming when it is relatively easy to identify target objects.

It can be seen that our algorithm is well performed on both the traffic data set and drone data set. For videos with different brightness, our algorithm adaptively adjusts the confidence threshold to save bandwidth while meeting the accuracy requirements as expected.

D. Test Bed Evaluation

To evaluate the performance of EVA in a more realistic scenario, we use an Nvidia Jetson intelligent toy car [29] to emulate the client side, as shown in Fig. 6. It has a specific operating system called Linux4Tegra OS, and a Jetpack software development kit (SDK). The SDK contains the general library, application programming interface (API) files, and developer tools. It has a quad-core ARM Cortex-A57 processor and an Nvidia Maxwell architecture with 128 Nvidia CUDA cores. To simulate the limited computing power at the camera side, our implementation runs on Nvidia Jetson's CPU. It works with the server to analyze the video captured by its camera. We measure the end-to-end latency of the system by inserting timestamps in the code, including server-side latency, network transmission latency and client-side latency. We first explore the impact of the input image resolution on end-to-end latency. As shown in Fig. 7, we can find that as the image resolution increases from 480 × 360 to 1280 × 720, these three latency components are all increasing. Due to the limitation of computing resources on the client side, its computing latency increases most obviously. In addition, we limit the resolution to 480 × 360 and explore the impact of the confidence threshold on the end-to-end latency. From the figure, we can find that when the confidence threshold increases from 0.4 to 0.8, the client side has higher requirements for the accuracy of the local recognition results, which will cause more target objects to be transmitted to the server for further recognition. Thus, the network transmission latency and server-side computing latency are increasing. It can be seen from the figure that although the confidence threshold keeps changing, the total end-to-end latency basically fluctuates around 500 ms.

VI. Related Work

Significant studies have been presented to improve the efficiency of video analytics pipelines, including exploiting the spatial redundancies, temporal redundancies, and adaptive configurations.

A. Spatial Redundancies

Du et al. [14] explored a DNN-driven approach to machine-centric video streaming, in which video compression and streaming are driven by the server-side DNN. Their approach improves bandwidth efficiency by sending only a few regions in high quality. Guo et al. [15] established cross-camera region associations to generate optimized RoI masks. Then, they applied the masks to boost real-time analytics performance. However, these studies do not make use of redundant information in the temporal domain to further save transmission bandwidth. Specifically, the size of the target object keeps changing over time. There is no need to repeatedly transmit low-quality and high-quality versions when the target object is relatively small. On the other hand, Liu et al. [16] designed a dynamic RoI encoding technique to adjust the encoding quality on each frame in order to reduce the transmission latency and bandwidth consumption in the AR offloading pipeline. Feng et al. [17] presented RoI-based viewport prediction approach for live VR streaming services. Lai et al. [18] dynamically adjusted visual quality according to the users' field of view to enable high-quality and low-latency VR. However, these studies allow the user to adjust the encoding quality for each macroblock in a frame and do not remove the background. For computer vision tasks, our method uses the region where the target object is located instead of the whole frame as the basic transmission element. In this way, we are going a step further based on RoI encoding.

B. Temporal Redundancies

Li et al. [19] presented a video analytics system that supports efficient real-time querying by performing on-camera frame filtering. Kang et al. [20] employed specialized binary classification models that eliminate frames that do not contain objects of interest. Chen et al. [21] adopted simple pixel-level differences to eliminate frames whose features have not changed substantially and are expected to produce the same results. Hsieh et al. [22] used compressed object recognition models (e.g., Tiny YOLO) that compute lower-confidence results to filter out frames. Zhang et al. [37] uploaded only the frames that best capture the scene when multiple cameras

looking at the same scene to suppress redundancy. However, these efforts do not make use of spatial information to further save bandwidth. That is to say, the transmitted frame contains a large proportion of nontarget object parts, and this information is useless to improve the accuracy of the machine-centric application.

C. Adaptive Configurations

Zhang et al. [23] presented a framework for building adaptive stream processing applications that simultaneously simplifies development and improves application accuracy in the face of limited or varying wide-area bandwidth. Jiang et al. [24] presented a controller that dynamically picks the best configurations for existing NN-based video analytics pipelines. Fang et al. [25] presented a framework that takes the dynamics of runtime resources into account to enable resource-aware multitenant on-device deep learning for mobile vision systems. Shen et al. [26] presented a scalable system for serving DNN-based video analytics. The system enables several optimizations in batching and allows more efficient resource allocation. Ran et al. [11] developed a measurement-driven framework that chooses the type of deep learning model and the location to perform a task based on application requirements, such as accuracy, frame rate, energy, and network data usage. Wang et al. [12] studied joint configuration adaption and bandwidth allocation for the edge-assisted real-time video analytics system and proposed an efficient online algorithm which can select appropriate configurations for multiple video streams. Hanyao et al. [13] maximized the overall accuracy of object detection by deciding the frequency of edge-assisted inference. These studies make configuration decisions either from a systematic or theoretical perspective to optimize a given goal. However, they do not leverage the uneven distribution of important pixels; instead, the videos are encoded by traditional codecs with the same quality level on each frame. In fact, the nontarget object parts are useless to improve the accuracy and impose burdens on the network transmission.

VII. Conclusion

In this article, we propose a framework that supports efficient real-time video analytics by leveraging both spatial and temporal redundancies. We used the region where the target object is located instead of the whole frame as the basic transmission element and transmit each object only once. Based on the framework, we designed an adaptive offloading algorithm, which coordinates the recognition process between the camera and the server according to current network conditions, in conjunction with the application's requirements. We also designed a confidence threshold adjustment algorithm to guarantee the robustness of our system. Results from extensive experiments showed that EVA outperforms several baseline systems: it reduces significant bandwidth usage by up to 90%, while consistently meeting the desired accuracy.

References

[1] 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. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 377–392.

[2] G. Ananthanarayanan, V. Bahl, L. Cox, A. Crown, S. Nogbahi, and Y. Shu, "Video analytics-killer app for edge computing," in *Proc. 17th Annu. Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2019, pp. 695–696.

[3] 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. (HotNets)*, 2018, pp. 50–56.

[4] Q. Zhang, H. Sun, X. Wu, and H. Zhong, "Edge video analytics for public safety: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1675–1696, Aug. 2019.

[5] J. Wang, B. Amos, A. Das, P. Pillai, N. Sadeh, and M. Satyanarayanan, "A scalable and privacy-aware IoT service for live video analytics," in *Proc. 8th ACM Multimedia Syst. Conf. (MMSys)*, 2017, pp. 38–49.

[6] M. Xu, T. Xu, Y. Liu, and F. X. Lin, "Video analytics with zero-streaming cameras," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2021, pp. 459–472.

[7] B. Haynes, A. Mazumdar, M. Balazinska, L. Ceze, and A. Cheung, "Visual road: A video data management benchmark," in *Proc. Int. Conf. Manag. Data (SIGMOD)*, 2019, pp. 972–987.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[9] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1–9.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

[11] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 1421–1429.

[12] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2020, pp. 257–266.

[13] M. Hanyao, Y. Jin, Z. Qian, S. Zhang, and S. Lu, "Edge-assisted online on-device object detection for real-time video analytics," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2021, pp. 1–10.

[14] K. Du et al., "Server-driven video streaming for deep learning inference," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2020, pp. 557–570.

[15] H. Guo, S. Yao, Z. Yang, Q. Zhou, and K. Nahrstedt, "CrossRoI: Cross-camera region of interest optimization for efficient real time video analytics at scale," in *Proc. 12th ACM Multimedia Syst. Conf. (MMSys)*, 2021, pp. 186–199.

[16] 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. (MobiCom)*, 2019, pp. 1–16.

[17] X. Feng, W. Li, and S. Wei, "LiveROI: Region of interest analysis for viewport prediction in live mobile virtual reality streaming," in *Proc. 12th ACM Multimedia Syst. Conf. (MMSys)*, 2021, pp. 132–145.

[18] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," *IEEE Trans. Mobile Comput.*, vol. 19, no. 7, pp. 1586–1602, Jul. 2020.

[19] 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. (SIGCOMM)*, 2020, pp. 359–376.

[20] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "NoScope: Optimizing neural network queries over video at scale," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1–12, 2017.

[21] 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 Netw. Sensor Syst. (SenSys)*, 2015, pp. 155–168.

[22] K. Hsieh et al., "Focus: Querying large video datasets with low latency and low cost," in *Proc. 13th {USENIX} Symp. Oper. Syst. Design Implement. (OSDI)*, 2018, pp. 269–286.

[23] 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. (SIGCOMM)*, 2018, pp. 236–252.

[24] 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. (SIGCOMM)*, 2018, pp. 253–266.

[25] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2018, pp. 115–127.
[26] H. Shen et al., "Nexus: A GPU cluster engine for accelerating DNN-based video analysis," in *Proc. 27th ACM Symp. Oper. Syst. Principle (SOSP)*, 2019, pp. 322–337.
[27] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 7263–7271.
[28] M. Meddeb, "Region-of-interest-based video coding for video conference applications," Ph.D. dissertation, Dept. Comput. Sci., Telecom ParisTech, Paris, France, 2016.
[29] "Nvidia Jetson Intelligent Toy Car." Accessed: Oct. 1, 2022. [Online]. Available: https://jetbot.org
[30] "Nvidia TensorRT." Accessed: Oct. 1, 2022. [Online]. Available: https://developer.nvidia.com/tensorrt
[31] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 197–210.
[32] Y. Sun et al., "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2016, pp. 272–285.
[33] R. Pantos and W. May, "HTTP live streaming," IETF, RFC 8216, 2017.
[34] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the Internet," *IEEE Multimedia*, vol. 18, no. 4, pp. 62–67, Apr. 2011.
[35] "Flask Web Framework." Accessed: Oct. 1, 2022. [Online]. Available: https://flask.palletsprojects.com
[36] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
[37] 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. (MobiCom)*, 2015, pp. 426–438.

Ziyi Wang received the B.E. degree in networking engineering from Dalian University of Technology, Dalian, Liaoning, China, in 2018. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China.

His research interests include video streaming and edge computing.

Xiaoyu He received the B.E. degree in computer science and technology from Huazhong University of Science and Technology, Wuhan, China, in 2021. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China.

His research interests include in-network computing and traffic shaping.

Zhizhen Zhang received the B.E. degree in software engineering from Tianjin University, Tianjin, China, in 2021. He is currently pursuing the M.S. degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China.

His research interests include video analysis and multimodal learning.

Yishuo Zhang received the B.E. degree from East China Normal University, Shanghai, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Software Engineering, Beijing University of Posts and Telecommunications, Beijing, China.

His research interests include mobile computing and multimedia interpretation.

Zhen Cao received the Ph.D. degree from Peking University, Beijing, China, in 2009.

He is currently a Principle Engineer with Huawei Technologies Company Ltd., Shenzhen, China. His research interests include sensor networks, security and privacy, cellular network technologies, IPv6, and SDN/NFV.

Wei Cheng received the master's degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2018. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Tsinghua University, Beijing.

He is currently a Vice President of the Unicom (Guangdong) Industrial Internet Company Ltd., Guangzhou, China. His research interests include cloud computing, edge computing, MEC, and IDC.

Wendong Wang (Member, IEEE) received the B.E. and M.E. degrees from Beijing University of Posts and Telecommunications, Beijing, China, in 1985 and 1991, respectively.

He is currently a Full Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He has published over 200 of papers in various journals and conference proceedings. His current research interests are the next-generation network architecture, network resources management and QoS, and mobile Internet.

Yong Cui (Member, IEEE) received the B.E. and Ph.D. degrees from Tsinghua University, Beijing, China, in 1999 and 2004, respectively.

He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has published more than 100 academic articles in refereed journals and conferences. He has authored a number of RFCs. His major research interests include mobile/wireless Internet and network architecture.

Prof. Cui received two best paper awards. He also serves as the Co-Chair for IETF IPv6 Transition Softwire WG.