VaBUS: Edge-Cloud Real-Time Video Analytics via Background Understanding and Subtraction

Hanling Wang, Qing Li, *Member, IEEE*, Heyang Sun, Zuozhou Chen, Yingqian Hao, Junkun Peng, Zhenhui Yuan, Junsheng Fu, and Yong Jiang, *Member, IEEE*

***Abstract*— Edge-cloud collaborative video analytics is transforming the way data is being handled, processed, and transmitted from the ever-growing number of surveillance cameras around the world. To avoid wasting limited bandwidth on unrelated content transmission, existing video analytics solutions usually perform temporal or spatial filtering to realize aggressive compression of irrelevant pixels. However, most of them work in a context-agnostic way while being oblivious to the circumstances where the video content is happening and the context-dependent characteristics under the hood. In this work, we propose VaBUS, a real-time video analytics system that leverages the rich contextual information of surveillance cameras to reduce bandwidth consumption for semantic compression. As a task-oriented communication system, VaBUS dynamically maintains the background image of the video on the edge with minimal system overhead and sends only highly confident Region of Interests (RoIs) to the cloud through adaptive weighting and encoding. With a lightweight experience-driven learning module, VaBUS is able to achieve high offline inference accuracy even when network congestion occurs. Experimental results show that VaBUS reduces bandwidth consumption by 25.0%-76.9% while achieving 90.7% accuracy for both the object detection and human keypoint detection tasks.**

***Index Terms*— Edge-cloud collaborative computing, semantic compression, video analytics, task-oriented communication system.**

I. Introduction

THE recent advances of deep learning techniques boost the performance of many computer vision applications, such as object detection, semantic segmentation and keypoint detection. These advances facilitate the commercialization of video analytics applications including traffic control [1], video surveillance [2], and safety anomaly detection [3]. In order to make video analytics technologies ready to use, three main concerns need to be addressed: latency, bandwidth and accuracy. First, analytics applications require very low latency, as typically the output is used to immediately notify humans or an actuator control system [4], [5], [6]. Second, high-definition (HD) videos require large bandwidth for transmission. Streaming the entire video from cameras to cloud might be infeasible, especially when available bandwidth is limited [4], [7]. Finally, deep learning models for video analytics need massive computing resources that are usually limited on cameras. Inferring with state-of-the-art models usually incurs unacceptable latency, while simplified models can cause deteriorated results on edge devices. A recent experiment shows that EfficientDet-D0, a lightweight object detection model [8], achieves an F1-score of only 50% (when using EfficientDet-D7 as the ground truth) with 34ms latency, whereas a high performance model EfficientDet-D7 has higher latency at 1636ms, on an NVIDIA Jetson Xavier NX [9] device with GPU support and TensorRT acceleration.

To address these problems, edge-cloud collaborative video analytics have emerged in recent years. Unlike the client-server model that needs a large amount of network bandwidth to transmit all data for central processing and suffers from single point of failure [10], edge-cloud collaborative computing offloads part of computation tasks to the edge to alleviate the pressure on networks and improve the system robustness. As a typical edge-cloud real-time video analytics pipeline, the edge device sends only partial video content to the cloud server, which runs deep learning models and returns the inference results [11], [12], [13], [14], [15]. For these solutions, one of the key factors to consider is the nontrivial bandwidth consumption. According to [16], Singapore aims to have more than 200,000 police cameras by at least 2030. And the number of all cameras in the world is estimated to be somewhere between 10 and 100 billion in 2030 [17]. Considering the bitrate of 2 Mbps for a typical HD video feed compressed by the H.264 codec [18], 200,000 cameras would need a bandwidth of 400 Gbps, which would impose heavy pressure on the network infrastructure. Unlike traditional human-centric video streaming that aims to accurately recover original video frames, machine-centric video streaming targets at accurate analysis results (e.g., detection of humans/cars). This discrepancy allows discarding analysis-unrelated visual information

Manuscript received 31 March 2022; revised 29 September 2022; accepted 9 October 2022. Date of publication 16 November 2022; date of current version 19 December 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 61972189, in part by the Major Key Project of Peng Cheng Laboratory (PCL) under Grant PCL2021A03-1, and in part by the Shenzhen Key Laboratory of Software Defined Networking under Grant ZDSYS20140509172959989. *(Corresponding author: Qing Li.)*

Hanling Wang, Junkun Peng, and Yong Jiang are with the Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong 518055, China, and also with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China (e-mail: hl-wang21@mails.tsinghua.edu.cn; pjk20@mails.tsinghua.edu.cn; jiangy@sz.tsinghua.edu.cn).

Qing Li and Zuozhou Chen are with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China (e-mail: liq@pcl.ac.cn; chenzzh@pcl.ac.cn).

Heyang Sun is with the School of Software, Southeast University, Nanjing, Jiangsu 211189, China (e-mail: 213182539@seu.edu.cn).

Yingqian Hao is with the School of Software, Jilin University, Changchun, Jilin 130012, China (e-mail: haoyq5519@mails.jlu.edu.cn).

Zhenhui Yuan is with the Department of Computer and Information Science, Northumbria University, NE1 8ST Newcastle Upon Tyne, U.K. (e-mail: zhenhui.yuan@northumbria.ac.uk).

Junsheng Fu is with Zenseact, 41756 Gothenburg, Sweden (e-mail: junsheng.fu@zenseact.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2022.3221995.

Digital Object Identifier 10.1109/JSAC.2022.3221995

for significant bandwidth saving. As a result, the demands on bandwidth resources for transmitting video data are lowered significantly to achieve a more sustainable communication network.

By exploiting the redundancy in video frames, extensive works have been proposed to perform spatial and temporal filtering for bandwidth saving. Temporal filtering solutions (e.g., [13], [14], [19], [20], [21], and [22]) aim to remove a video frame that is duplicated or does not contain any interesting objects. In contrast, spatial filtering solutions (e.g., [11] and [12]) allocate more bits for pixels in the RoIs during encoding and transmitting. Unfortunately, most of them [12], [13], [14], [15] adopt context-agnostic approaches (i.e., treating incoming frames as independent images instead of a continuous flow of frames from one camera) during streaming, and more contextual information remains under-explored, e.g., the background of the scene, unequal probability of observing objects across different regions of the frame, and the size of objects occurring at the same location.

In this work, we explore the contextual information of video data from surveillance cameras. Since surveillance cameras generate video frames with fixed viewing angle, certain characteristics tend to persist over time, such as regions that may observe objects and the size of occurred objects at the same location. By exploiting these context-dependent characteristics, we design **VaBUS**, a new real-time **V**ideo **a**nalytics system based on **B**ackground **U**nderstanding and **S**ubtraction. Specifically, VaBUS first reconstructs the background from camera video feeds in the cloud and transfers the background image to the edge with minimum overhead, then the edge sends the cloud with only useful and optimized foreground pixels that may contain interesting objects for inference. Ideally, RoIs in a video frame only contain objects to be detected, and the bits for remaining regions will not be pushed into the network, i.e., the RoIs are optimally compressed on the semantic aspect. This stems from the idea of semantic communication [23], which aims at the successful transmission of semantic information conveyed by the source rather than the accurate reception of each single symbol or bit regardless of its meaning. As a result, VaBUS is able to support a large number of cameras for accurate and real-time analysis by leveraging edge-side computing resources.

Although removing redundancies between frames is a common approach in existing codecs (e.g., motion estimation and motion compensation in H.264/AVC [18] and H.265/HEVC [24]), VaBUS is able to further improve the compression ratio in two aspects: 1) The fluctuations of background pixel values are eliminated by removing the background from RoIs; 2) Lower encoding quality is set to regions that occur large objects without impacting the inference accuracy. In other words, VaBUS is an enhancement to existing compression codecs, which further eliminates redundancy from the semantic level to achieve better compression performance. To the best of our knowledge, the context characteristics of static background for surveillance cameras has not yet been systematically exploited by prior works in the real-time video analytics scenario.

However, the edge-cloud collaborative video analytics based on background understanding and subtraction is not a trivial task in the real-world scenario. There are mainly three challenges to tackle before practical deployment.

- First, we need a lightweight approach to effectively reconstruct the static background image from videos under the edge-cloud collaborative framework, such that we can eliminate these redundant pixels on the edge before transferring the video to the cloud. On one hand, performing background reconstruction on the edge, though possible, usually fails to meet the latency target and induces high system overhead due to limited computing resource. According to [25], background reconstruction for an RGB image of 1080×1920 on an Nvidia Jetson AGX Xavier requires 75ms on average when using CPU, which is far from being real-time. On the other hand, reconstructing the background image on the cloud faces the challenge of ensuring efficient and effective background synchronization on the edge due to changes of video illumination conditions or camera viewing angle.
- Second, we need a robust approach to generate accurate RoIs (i.e., the foreground pixels that contain interesting objects) on the edge. Accurate RoI is the key to save huge bandwidth without impacting the inference accuracy. Simple pixel-wise difference between incoming frames and the background image contains considerable detection noise. Misdetection of RoI (i.e., too few pixels sent) risks missing interested objects while false alarm of RoI (i.e., too many pixels sent) contains meaningless-for-inference pixels, causing extra bandwidth waste.
- Third, we need to design a bandwidth-aware mechanism to dynamically balance the accuracy and delay of inference tasks under variable network conditions. An offline estimation strategy is required to deal with unexpected network interruptions, which estimates the inference results on the edge when it fails to receive results from the cloud within the required time frame. Accurate estimation of inference results with lightweight but low-latency approach on the edge side is a challenging task.

To validate the feasibility of VaBUS, we implement a prototype (available at https://github.com/kongyanye/VaBUS) based on Python and C++. With the prototype, we conduct comprehensive experiments on four real-world datasets. Results show that VaBUS 1) reduces bandwidth consumption by 25.0%-76.9% while achieving 90.7% accuracy, 2) incurs only 477.5ms latency and 10% CPU usage overhead compared to a baseline approach, and 3) achieves 68% offline estimation accuracy which outperforms both the optical flow [21] and motion vector-based methods [11].

The contributions of this paper can be summarized as follows:

- We are the first to utilize background understanding and subtraction in edge-cloud collaborative video streaming for analysis purposes.
- We designed a background reconstruction scheme under the edge-cloud collaborative framework. The cloud

effectively reconstructs the background image with partial frames sent by the edge, and the edge efficiently maintains the background image with a *dynamic overlay* mechanism to reduce bandwidth consumption.
- We further increased the video compression ratio by proposing an *adaptive weighting* module and an *adaptive RoI encoding* strategy. By learning from historical detection results and frame pixel values, the *adaptive weighting* module assigns greater weights to regions that are more likely to appear objects, and the *adaptive RoI encoding* strategy assigns imbalanced encoding quality to the frame according to distribution of object size across regions.
- We improved offline inference accuracy by designing a novel experience-driven learning module, i.e., *mapping estimator* that automatically learns *shifting* and *scaling* mapping functions from previous detection results to produce accurate estimations even when network congestion occurs.

The rest of the paper is organized as follows. Related work is presented in Section 2, followed by the system overview and design details of VaBUS in Section 3. Section 4 shows the experimental results. Section 5 discusses the limitation and future work of VaBUS and Section 6 concludes the paper.

II. Related Work

A. Real-Time Video Analytics System

Although surveillance camera applications have been studied in the past decade, real-time video analytics system on edge device is a new topic. A series of techniques have been proposed to apply real-time video analytics within the edge-cloud continuum. Previous works have explored DNN-specific optimization techniques including model distillation [26], splitting [27], sharing [28] and cascading [29], adaptively tuning a set of control knobs to constantly meet accuracy and latency requirements [7], [30], [31], optimized information pruning techniques [11], [12], [13], [14], etc.

Among them, VaBUS focuses on a single aspect of the design space, i.e., information pruning. Existing works typically discard irrelevant information from the temporal perspective by frame filtering or from the spatial perspective by assigning uneven encoding quality across the frame. For example, FilterForward [13] (i.e., from temporal perspective) deployed microclassifiers on edge devices to only detect relevant events and frames that are to be transmitted to cloud servers. DDS [12] (i.e., from spatial perspective) constantly sent a low-quality stream and resent high-quality partial images based on feedback from the server. VaBUS shares the same concept of performing imbalanced encoding quality on the frame, while in a more aggressive approach by subtracting the background and assigning lower quality to large objects. As a consequence, VaBUS is able to save more bandwidth while achieving high accuracy.

To allow RoI generation to assign different encoding quality with the purpose of video analytics, a number of works have been proposed [11], [12], [32], [33]. EAAR [11] used the candidate RoIs from last frame to determine the encoding quality of the next frame. DDS [12] generated RoIs by utilizing the information returned by the server-side DNN. AccMPEG [32] trained a low-cost quality selector model to decide the appropriate quality level for a macroblock. Elf [33] employed a recurrent region proposal prediction algorithm to estimate the possible locations of objects for partitioning and offloading. VaBUS differs from them by leveraging characteristics of surveillance videos, i.e., subtracting the static background image and refining RoIs by historical detection results.

B. Background Reconstruction and Subtraction

Background reconstruction and subtraction have been extensively studied in the last decade [34]. Its goal is to detect moving objects in the scenes when the camera is static, i.e., segmentation of foreground and background. Existing approaches can be divided into two categories: traditional methods and deep learning-based methods. Popular techniques used in traditional methods include parametric method such as GMM [35], non-parametric method such as kernel density estimation [36], traditional machine learning method [37] and hybrid methods with multi-modal data [38] and model fusion [39], etc. Deep learning-based methods can be classified into supervised and unsupervised methods. Supervised methods typically adopt 2D- or 3D- CNN and ConvLSTM models [40], [41], [42], [43], and concentrate on scene-specific or scene-agnostic scenarios. Unsupervised models based on GANs and Autoencoders, have also emerged as new approaches in recent years [44], [45]. In VaBUS, we modify existing approaches for background reconstruction (i.e., GMM) to adapt to the edge-cloud collaborative framework and propose a novel adaptive weighting module for accurate background subtraction.

Background reconstruction and subtraction has been widely adopted as a tool to remove redundancy between frames for video compression in previous works [46], [47], [48], [49], [50]. These works focus on improving the compression performance within a codec, by leveraging the characteristics of surveillance video. VaBUS differs from previous works mainly in that the scenario is different. Instead of aiming to recover the original frames for user-perceived Quality of Experience (QoE), VaBUS aims to achieve high inference accuracy for analysis purposes, where background reconstruction is only one of the tools we use for scene understanding. The scenario discrepancy also leads to the difference in focus. Unlike previous works that focus on using the least bits to recover the best quality of frames, VaBUS focuses on the design of a whole system, including other criteria like inference accuracy and network failure handling.

C. Computer Vision Applications

A large number of deep learning models have been proposed for various vision applications, including image classification [51], instance segmentation [52], object detection [53], face recognition [54], image captioning [55], etc. Recent works show that it is inefficient to perform inference for every video frame. Alternatively, augmentation of inference results should be applied via the usage of tracking or temporal prediction models [21]. For real-time video analytics, several techniques have been proposed to meet the latency requirement via

local tracking on the edge devices. Glimpse [21] used the optical flow method to estimate the detection results of next frame based on previous frame. EAAR [11] adopted a more lightweight method by leveraging motion vector information from the codec. Unlike traditional (such as optical flow and motion vector-based) methods that can only estimate subtle location changes with fixed object size, VaBUS is able to deal with more drastic objection location and size change by leveraging context-dependent information, i.e., previous inference results.

As a mid-level task in computer vision, Multiple Object Tracking (MOT) paves the ground for high-level tasks such as pose estimation, action recognition, and behavior analysis. The task of MOT is partitioned into locating multiple objects, maintaining their identities, and yielding their individual trajectories given an input video [56]. Based on whether a detection model is adopted, MOT algorithms can be grouped into two sets: Detection-Based Tracking (also referred to as tracking-by-detection) [57], [58] and Detection-Free-Tracking [59], [60]. By first detecting target objects with a detection model, Detection-Based Tracking links the same object across frame into trajectories. In VaBUS, we use the existing MOT algorithm (i.e., SORT [57]) to track the detected objects for dynamic overlay of static objects and updating of mapping function.

III. VaBUS

In this section, we start with the design principles of VaBUS and how it operates at a high level. Then, we present the details of each component in VaBUS.

A. Design Principles and Overview

1) Principles of Designing VaBUS: The design rationale of VaBUS is to find a favorable tradeoff between bandwidth consumption and latency/accuracy. We compromise small degradation of latency and accuracy performance for huge bandwidth saving. Our objective is to design a real-time edge-cloud video analytics system that saves as much bandwidth as possible while still achieving high inference accuracy and low latency. To achieve this goal, we leverage the contextual information of video feeds and take the following principles into consideration.

- **Accuracy**. Provide 90% accuracy. In VaBUS, we only send partial video frames with degraded encoding quality for bandwidth saving, which inevitably causes a drop in inference accuracy. We deem 90% is of high accuracy and it is a typical target accuracy used in many previous works related to video analytics [14], [61], [62], [63], [64].
- **Latency**. Achieve the latency of one second. This is based on our observations: 1) Performing inference with state-of-the-art models directly on the edge fails to meet the real-time requirements; 2) Streaming all raw video frames to the cloud overwhelms the network infrastructure when the camera number surges [16], [17]. Under the edge-cloud collaborative framework, we deem that the second-level end-to-end latency is appropriate for typical real-time video analytics applications. This is consistent with previous works [4], [5], [6], [19], [65]. Suppose the camera gets a batch of frames at time t_1, the edge receives the returned results and finished postprocessing at time t_2. The latency is defined as $t_2 - t_1$.
- **Bandwidth**. Reduce bandwidth consumption. When accuracy and latency meet the requirements, VaBUS tries to save as much bandwidth as possible.
- **Handling Network Variance**. Provide alternatives in poor network conditions. When network variance suddenly increases, e.g., due to network congestion, the system shall have an alternative solution to provide immediate yet less accurate estimation results for the task, so as to avoid blocking the system.

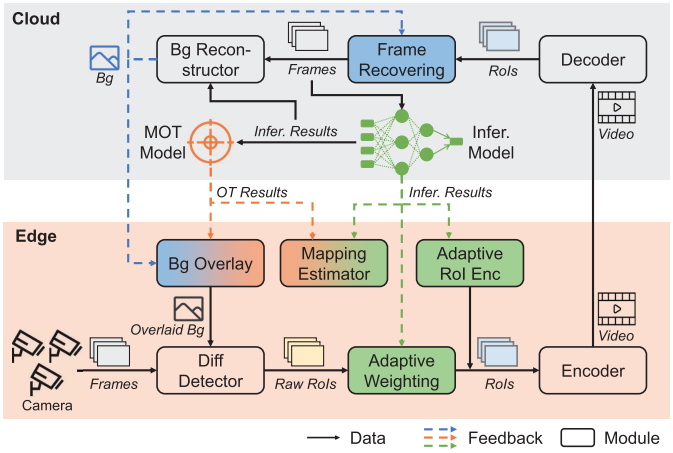


Fig. 1. System Overview of VaBUS.

2) VaBUS Overview: Figure 1 illustrates the overall architecture of VaBUS and indicates its life cycle. The operation process can be summarized as the following steps:

- The edge device receives incoming frames and generates raw RoIs by subtracting the background through a *difference detector* (referred to as *diff detector*).
- The raw RoIs are further refined by the *adaptive weighting* module to only contain interesting objects.
- Once ready, the RoIs will be encoded into a video using an *adaptive RoI encoding* (referred to as *adaptive RoI Enc*) module, which assigns imbalanced encoding quality considering object size in different regions of the frame.
- After receiving the video, the cloud will first decode the RoIs from it and then recover original frames with a *frame recovering* module.
- Upon the frames are successfully recovered, the inference model (referred to as *infer. model*) will be run to generate results. At the same time, the *background reconstructor* (referred to as *bg reconstructor*) will continuously learn to reconstruct the background and send the latest one to the edge when necessary. Besides, an extra object tracking model (referred to as *MOT model*) is also run in the cloud to recognize static objects.
- The edge receives inference results (referred to as *infer. results*) as well as object tracking results (referred to as *OT results*), which will be used to update various modules. The inference results are used to update the

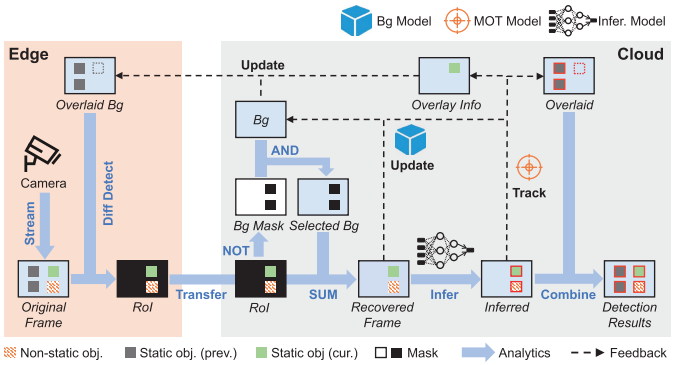


Fig. 2. Background reconstruction and dynamic overlay.

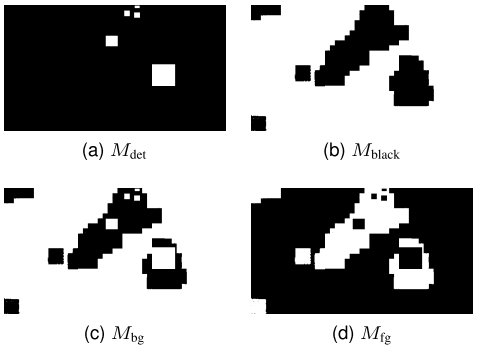


Fig. 3. Example images for M_{det}, M_{black}, M_{bg} and M_{fg}.

adaptive weighting, *adaptive RoI Enc* and *mapping estimator* module. The tracking results are used to update the *bg overlay* and *mapping estimator* module on the edge.

In general, the edge device constantly generates accurate RoIs, while the cloud is responsible for model inference and module updating through video context analysis. As a result, the edge grasps a better understanding of the scenario where the video is happening with minimal overhead and huge bandwidth saving is realized while achieving the ultimate goal. In the rest of this section, we discuss the details of each module and focus on how they are designed to meet the aforementioned principles.

B. Background Understanding and Learning

After receiving RoIs from the edge, the *frame recovering* module is firstly used to recover the original frames on the cloud. The *bg reconstructor* continually learns the background from recovered frames in the cloud and selectively updates the background at the edge. By recognizing static objects with a fast multiple-objects tracking model (referred to as *MOT Model*), non-object background are overlaid with static objects to create new temporary background which further reduces the size of generated RoIs on the edge. The whole pipeline is illustrated in Figure 2.

1) Frame Recovering and Background Updating: The background reconstructor aims to learn a static non-object scene image by removing foreground objects from video frames, where a background estimation model (referred to as *bg model*) lies in the core. Given a sequence of frames $\mathcal{X} = X_1, \cdots, X_N$ of a scene showing moving objects (e.g., cars and pedestrians), the goal of a background estimation model is to recover a clean image of the background of this scene, without any moving objects. Here the pixels of moving objects are referred to as foreground, and the remaining as background.

In VaBUS, we use the classic model proposed by [66] and implemented in OpenCV [67] as the backbone background estimation model. Note that although we only use a simple model in our implementation, more complicated and advanced background estimation model, e.g., deep learning-based models, can be used since it is deployed in the cloud with almost no resource limit. The procedures of reconstructing and updating the background are shown in Algorithm 1.

The *recovering and reconstruction phase* takes RoI I and bounding box D for a single frame as inputs. Since the RoI only contains a part of the original frame, reconstructing the background from RoI might cause wrong background estimation. Instead, we feed into bg model the frame recovered I_{rec} using both background image B with mask of M_{bg} and RoI I with mask of M_{fg} (Line 13). M_{bg} represents the pixels to be taken from the background image, which is the logical AND of M_{det} (i.e., detected regions in current RoI) and M_{black} (i.e., eliminated background regions in current RoI) (Line 7). After obtaining M_{bg}, the M_{fg} (i.e., the pixels to be taken from the RoI) can be easily calculated by taking the photographic negative of M_{bg} (Line 11). Example images for M_{det}, M_{black}, M_{bg} and M_{fg} are shown in Figure 3. The rationale behind this is twofold: removing detected objects before reconstruction avoids reconstructing the objects as background, while filling erased pixels in RoI with existing background explicitly inhibit the re-estimation of these regions. To avoid server-side computing resource waste, we only reconstruct the background when the ratio of detected to valid pixel number (i.e., non-black regions in RoI) is lower than a specific threshold λ_{det} (Line 10). In the *updating phase*, a renewed background image is considered for updating only when the last sent background is substantially different with the current one, i.e., over threshold λ_{chg}. And it would be sent to the edge only when at least λ_{gap} batches have elapsed since the last update (Line 18-21), which avoids sending updates to the edge too frequently and further reduces the communication overhead of background transmission.

2) Dynamic Overlay on Non-Object Background: The *bg reconstructor* proposed in Section III-B.1 aims to reconstruct non-object part of the video feeds, e.g., roads, trees, etc. However, detected objects could also be part of the background, e.g., cars in parking lots, standstill humans on roads. In order to save the bandwidth of transferring static detected objects in the RoI, we design a dynamic overlay mechanism to add these objects onto the background for RoI generation at the edge. It takes the bounding boxes of static objects tracked on the cloud and the original frames as inputs, then dynamically generates new background images by overlaying these static objects.

The key of identifying static objects is to track each individual object over multiple frames and check whether

Algorithm 1 Frame Recovering and Background Updating

Inputs: RoI I and bounding boxes D for a single frame, image width W and height H, λ_{roi}: background pixel threshold, λ_{det}: detect ratio threshold to update Bg model, λ_{chg}: threshold to determine whether the background is substantially changed, λ_{gap}: minimum updating interval
Outputs: Background estimation model $\mathcal{M}$, background image B
Step 1: *Recovering and Reconstruction Phase*
Initialize background image B, estimation model $\mathcal{M}$
$M_{\text{det}}^{i,j} \leftarrow \begin{cases} 1, & \text{if pixel in } D \\ 0, & \text{otherwise} \end{cases}$
$M_{\text{black}}^{i,j} \leftarrow \begin{cases} 1, & \text{if } \sum_{c=1}^{3} I_{i,j,c} < \lambda_{\text{roi}} \\ 0, & \text{otherwise} \end{cases}$
$M_{\text{bg}} \leftarrow M_{\text{det}}$ **OR** M_{black}
$n_{\text{det}} \leftarrow$ number of detected pixels in D
$n_{\text{roi}} \leftarrow$ number of non-zero pixels in I
if $n_{\text{det}}/n_{\text{roi}} < \lambda_{\text{det}}$ **then**
  $M_{\text{fg}} \leftarrow$ **NOT** M_{bg}
  $I_{\text{fg}} \leftarrow I$ **AND** M_{fg}; $I_{\text{bg}} \leftarrow B$ **AND** M_{bg}
  $I_{\text{rec}} \leftarrow$ **SUM**$(I_{\text{fg}}, I_{\text{bg}})$
  $\mathcal{M} \leftarrow$ update with I_{rec}
  $B \leftarrow$ get new background image from $\mathcal{M}$
end if
Step 2: *Updating Phase*
$d \leftarrow \frac{1}{H \times W} \sum \|B - B_{\text{last}}\|$ (B_{last}: last sent background image to the edge)
if $d > \lambda_{\text{chg}}$ and B haven't been updated for λ_{gap} rounds **then**
  update B on the edge
end if

their corresponding locations have changed or not. To this end, we use an existing online and real-time MOT algorithm, i.e., SORT [57] in the cloud to track detected objects. For each object, if the location remains the same across a certain number of frames (e.g., a batch), the object is considered as static. The ID and bounding boxes of static objects are then sent to the edge for further processing.

When a new static object $\mathcal{O}$ appears, the incoming frames are compared with the existing background image to generate RoIs (note that $\mathcal{O}$ is in the RoIs at this moment). Once the cloud receives the RoIs from the edge and finishes the inference, the inference results are used by the MOT model to recognize the static object $\mathcal{O}$ and the static objects information as well as inference results are returned to the edge. After the edge receives the inference results and static objects information, the background image is updated by overlaying the static object $\mathcal{O}$. At the same time, the detection results of object $\mathcal{O}$ are cached for future use. If the location of static object $\mathcal{O}$ remains unchanged in later incoming frames, it would be successfully removed from the RoIs to save bandwidth. Note that the recognition of static objects is done on the cloud instead of edge, which means the static objects are still in the previously sent RoIs during the initial batches of frames since it appears.

Since these static objects could be temporary, we need to remove them from the background when they are no longer static (i.e., start to moving). This can be achieved by examining the RoIs at the edge. If the generated RoIs overlap with regions of an overlaid object, it means the object has moved. Then the background is reset to be the non-object one (i.e., before overlaying) and the cached detection results are also cleared. Otherwise, the object is still static and its detection results could be used as part of inference results. In addition, when the edge receives new background image from the cloud, the overlay information will also be cleared (i.e., to the state of not containing static objects).

Dynamically overlaying objects on non-object background acts as an efficient way to generate background while reducing the communication overhead of background transmission between the edge and cloud. The edge only needs to know which object is static in order to add them to the background, and the overhead of transmitting these information is negligible. Overlaid objects are automatically removed once they move by checking the RoIs at the edge. Note that the MOT model in the cloud operates on the object level, we only implement background overlay for the object detection task. However, the idea of overlaying the background with static objects can be generalized to other tasks. For example, we can use a multiple keypoints tracking model to overlay objects for the human keypoint detection task.

C. Adaptive RoI Generation

After receiving background image B from the cloud, each input frame is compared against B for RoI generation. However, generating accurate RoIs is difficult due to inevitable background reconstruction noise. In this paper, we propose a lightweight *adaptive weighting* module that learns from past inference results to generate accurate RoIs. To further reduce the bandwidth consumption of transferring RoIs, we take the object size in RoIs into consideration and adopt adaptive RoI encoding.

1) Difference Detector: RoI generation is essential for bandwidth saving. It adopts the idea of only transferring the foreground in a frame while removing the background to reduce encoded video size, hence saving bandwidth consumption. The background part (e.g., roads and grasses) of a video, though perceived as remaining the same by human eyes, are constantly changing (i.e., the pixels are fluctuating within a small range) due to its luminance/chrominance variation. Despite these pixel differences have to be encoded in traditional codecs in order to accurately recover the original frames, they can be neglected in video analytics. By eliminating the fluctuations of background pixels (i.e., set to the same value of zero), the frames can be optimally encoded by existing compression codecs. Similar approaches have been proposed in [12], which used server feedback to determine high quality RoIs in the second run. In VaBUS, we leverage the characteristic of fixed background to remove pixels unrelated to the task in one run.

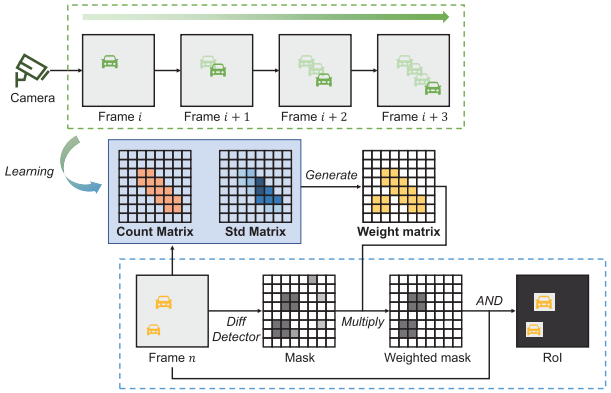


Fig. 4. Adaptive weighting learning process.

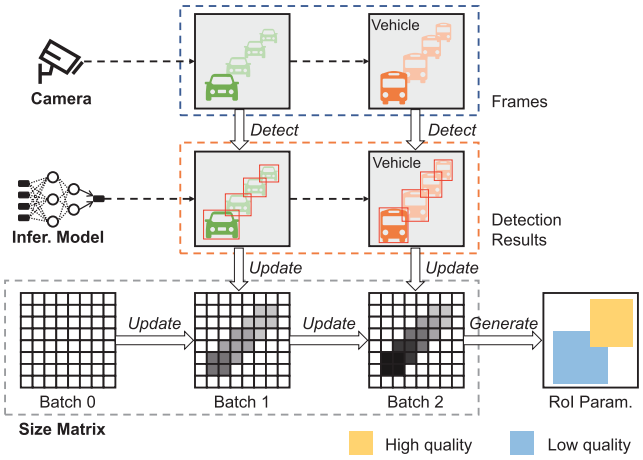


Fig. 5. Learning adaptive RoI encoding parameters.

Given a background image B and video frame I, the mask of RoI can be computed as

$$M_{\text{roi}}^{i,j} = \begin{cases} 1, & \text{if } d(I,B)^{i,j} > \lambda_{\text{roi}} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $d(I,B) = |\text{conv2d}(I) - \text{conv2d}(B))|$, conv2d denotes the convolution operation and λ_{roi} is the distance threshold to determine whether a pixel is the same as in background or not. The convolution operation creates a smooth version of B and I to remove noise and also decreases the computation cost of later procedures on the edge device. We further perform a morphological operation (i.e., dilation) on M_{roi}, to ensure the object of interest is fully visible in the RoI. With M_{roi}, we can take RoIs from frames I using M_{roi}, while setting remaining pixels to be black.

2) Adaptive Weighting: Since the reconstructed background image contains noise and the pixel value for background region is unlikely to be identical across frames, simply generating RoIs using pixel-wise image difference as in Equation 1 would produce a number of false-alarmed regions. Therefore, we propose an *adaptive weighting* module to boost the accuracy of RoIs. Specifically, we add a weight to $d(I,B)$ as

$$d(I,B) = w(I) * |\text{conv2d}(I) - \text{conv2d}(B))| \tag{2}$$

where the calculation of $w(I)$ is described below.

Firstly, we maintain a *Count* matrix C of $h \times w$ (initialized as zero matrix, h and w is the height and width of the image after the conv2d operation) to record the number of rounds since the last detected object occurring at each pixel location. A lower value in C means that the specific pixel location observes objects more recently. For each image, the values of C locating in detected regions are decreased by γ_1, and remaining values are increased by γ_2. In our implementation, we choose $\gamma_1 > \gamma_2$ to enable the value of regions which recently observed objects to increase more gradually. Under this setting, once a region (x_1, y_1, x_2, y_2) ((x_1, y_1) denotes the coordinate of upper-left corner and (x_2, y_2) denotes the coordinate of lower-right corner) observes a detected object (i.e., $C[y_1 : y_2, x_1 : x_2] - = \gamma_1$), it will take more than one round (i.e., batches of frames) to increase the values in $C[y_1 : y_2, x_1 : x_2]$ to zero. The count matrix C is used to calculate the weight w through a mapping function f that we will cover shortly.

For each incoming frame I, we also maintain a *Std* matrix S, which calculates the standard deviation of pixel values at each location in a sliding window manner. When generating weights $w(I)$ from the count matrix C, pixels in C where $S_{i,j}$ is larger than threshold λ_{std} and $C_{i,j} > 0$ will be reset to zero. The reason to do so is that high variance of pixel values (i.e., $S_{i,j} > \lambda_{\text{std}}$) may indicate the occurrence of new objects. This ensures pixels being suppressed (i.e., $C_{i,j} > 0$) can be reactivated when large value fluctuation occurs, thus avoiding miss-detection of objects that occurs in these suppressed pixels.

Finally, the weight matrix for a given image I is calculated as $w(I) = f(C)$ where $f(x) = [1 - x/\lambda_1]_0^{\lambda_2}$ is a linear mapping function which is clipped in the range $[0, \lambda_2]$. λ_1 is a positive number, controlling the suppressing speed of regions where no objects are observed. λ_2 (also a positive number) determines the maximum highlighting ratio. The weight matrix w monotonously decreases with the increase of C, enabling regions with no objects occurred and pixel value unchanged to be suppressed, while regions that continually observe objects to be highlighted. The process is illustrated in Figure 4.

3) Adaptive RoI Encoding: Encoding with unbalanced quality levels across the frame has been shown to be an effective approach of reducing bandwidth consumption in prior works [11], [12]. For compression codecs, different regions of the frame are treated equally during encoding. In the video analytics scenario, however, objects of different sizes have different impacts on the inference results, i.e., small objects require higher encoding quality for accurate detection compared to large objects. By encoding regions that require fewer details (e.g., regions with large objects) with lower quality, the inference result is the same while the bandwidth consumption is reduced. In VaBUS, background subtraction and overlay work as the first step to assign unbalanced encoding quality, i.e., setting background pixels to black which can then be optimally compressed. To further reduce the size of encoded videos, we take the object size in frames into consideration and propose adaptive RoI encoding, as shown in Figure 5.

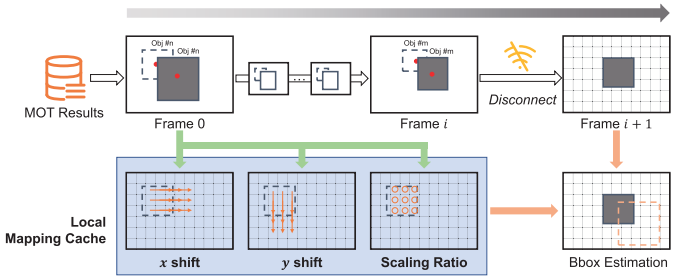


Fig. 6. Offline experience-driven learning.

The key of adaptive RoI encoding is to assign higher encoding quality to regions with small objects and lower quality to regions with large objects. When the object is large in size, deep learning models could produce accurate inference results despite of low encoding quality. However, when the object is small, deep learning models are susceptible to quality levels and might fail to detect the objects when encoding quality drops. For each frame in the cloud, we keep a size matrix Z to log the size of detected object per pixel location. For example, given an object with bounding box (x_1, y_1, x_2, y_2), Z is updated with $Z_{[y_1:y_2,x_1:x_2]} = (1-\alpha) * Z_{[y_1:y_2,x_1:x_2]} + \alpha * (x_2 - x_1) * (y_2 - y_1)$ where α is the updating ratio. When generating RoI encoding parameters, we first classify values in Z to either large or small size using a threshold λ_{size}. Regions whose values are smaller than λ_{size} are enclosed in rectangles to be set with higher encoding quality while the remaining with lower quality. The RoI encoding parameters are periodically sent to the edge device, to encode incoming frames.

Algorithm 2 Mapping Function Update

1: **Inputs:** tracked bounding box D_i^j (i.e., j^{th} object in i^{th} frame), updating rate α (e.g., 0.8), expanding ratio β (e.g., 1.05)
2: **Outputs:** local mapping cache S_x, S_y and E
3: Initialize mapping matrix S_x, S_y and E as zero matrix
4: **for** bounding box (x_1, y_1, x_2, y_2) in D_i^j **do**
5: $(x_1', y_1', x_2', y_2') \leftarrow D_{i-1}^j$
6: $m_x' \leftarrow (x_1' + x_2')/2$; $m_y' \leftarrow (y_1' + y_2')/2$
7: $m_x \leftarrow (x_1 + x_2)/2$; $m_y \leftarrow (y_1 + y_2)/2$
8: $S_x[y_1 : y_2, x_1 : x_2] \leftarrow (1-\alpha) * S_x[y_1 : y_2, x_1 : x_2] + \alpha * (m_x - m_x')$
9: $S_y[y_1 : y_2, x_1 : x_2] \leftarrow (1-\alpha) * S_y[y_1 : y_2, x_1 : x_2] + \alpha * (m_y - m_y')$
10: $r \leftarrow (x_2 - x_1)/(x_2' - x_1') * (y_2 - y_1)/(y_2' - y_1') * \beta$
11: $E[y_1 : y_2, x_1 : x_2] \leftarrow (1-\alpha) * E[y_1 : y_2, x_1 : x_2] + \alpha * r$
12: **end for**

D. Offline Estimation

The *mapping estimator* module works as an alternative approach to generate estimated inference results when the edge fails to receive response from the cloud server within the given time frame. Unlike traditional approaches that estimate inference results based on motion vectors or optical flow [11], [21], we propose a new lightweight experience-driven method to learn from previous inference results. First, the cloud runs a MOT model (same as in §III-B.2) to track each object and sends the tracking information to the edge. Then the edge device learns a mapping from the last bounding box to the current one for each individual object. When the cloud fails to deliver inference results on time, the edge would produce estimated results based on local mapping cache. The pipeline is illustrated in Figure 6.

1) Mapping Function Update: The details of updating local mapping cache based on detection results are shown in Algorithm 2. We use matrix S_x, S_y to record the shift on x- and y-axis direction respectively and E to record scaling ratio per pixel location. For each individual object, the shifts on x- and y-axis direction are calculated as the corresponding difference of center point from two adjacent frames (line 6 - 9). Scaling ratio is defined as the ratio of current bounding box size to the previous one, multiplied by an expanding ratio β to slightly enlarge the region (line 10). For smooth changing of parameters, we update each parameter with the rate of α.

2) Bounding Box Estimation: Given an initial bounding box B_0 for frame I, the *mapping estimator* module tries to predict the inference results of n consecutive frames using the local mapping cache S_x, S_y, and E, as shown in Algorithm 3. For each bounding box (x_1, y_1, x_2, y_2), we first take the average of S_x, S_y, and E from the same region as the shifting and scaling ratio (line 5 - 6). To avoid error accumulation, we compensate the shifting and expanding ratio in each round (line 7 - 8). Finally, the bounding boxes of the previous frame are shifted and scaled to generate new bounding boxes for the current frame (line 9 - 13).

Our mapping estimator is based on the premise that objects at the same location follow the same moving pattern, i.e., the direction and speed of objects at the same pixel location is similar in adjacent frames. This pattern is commonly observed, e.g., cars on the same lane have similar trajectory. In addition to predicting shifts on the x- and y-axis direction as in traditional methods, the mapping estimator is also able to predict scaling ratio, resulting in more accurate estimation.

IV. Experimental Results

A. Implementation

1) Experimental Setup: In the hardware setup, we use a Dell Precision 7920 Workstation Desktop Tower with a GeForce RTX 3090 GPU as the cloud server, and an Nvidia Jetson Xavier NX connected with a CSI camera as the edge device. Jetson Xavier NX is a typical edge intelligence device used in many previous works for video analytics [68], [69], [70], [71]. The cloud server and edge device are connected with a TP-LINK TL-SG1008D switch through a 1Gbps Ethernet cable. The operating system of cloud server and edge device are Ubuntu 20.04 and Ubuntu 18.04 respectively.

In the software setup, we implement the cloud part as an HTTP server which receives encoded videos from the edge device and returns inference results. On Jetson, we use the Multimedia [72] API for real-time video encoding where the setROIParams function [73] is used to set different RoI regions

Algorithm 3 Bounding Box Estimation

Inputs: initial bounding box D_0, prediction step n, mapping cache S_x, S_y and E, shrinking rate β_1 (e.g., 0.99) and expanding rate β_2 (e.g., 1.01)
Outputs: estimated bounding box $D_1, D_2, \cdots, D_n$
for i^{th} round in $1, 2, \cdots, n$ **do**
  for (x_1, y_1, x_2, y_2) of j^{th} object in B^j_{i-1} **do**
    $s_x \leftarrow \overline{S_x}[y_1 : y_2, x_1 : x_2]$; $s_y \leftarrow \overline{S_y}[y_1 : y_2, x_1 : x_2]$
    $e \leftarrow \overline{E_x}[y_1 : y_2, x_1 : x_2]$
    $s_x \leftarrow \beta^i_1 * s_x$; $s_y \leftarrow \beta^i_1 * s_y$
    $e \leftarrow \begin{cases} \max(1, \beta^i_1 * e), \text{ if } e > 1 \\ \min(1, \beta^i_2 * e), \text{ otherwise} \end{cases}$
    $\hat{x}_1 \leftarrow x_1 + s_x - (e-1) * (x_2 - x_1)/2$
    $\hat{x}_2 \leftarrow x_2 + s_x + (e-1) * (x_2 - x_1)/2$
    $\hat{y}_1 \leftarrow y_1 + s_y - (e-1) * (y_2 - y_1)/2$
    $\hat{y}_2 \leftarrow y_2 + s_y + (e-1) * (y_2 - y_1)/2$
    $B^j_i \leftarrow (\hat{x}_1, \hat{x}_2, \hat{y}_1, \hat{y}_2)$
  end for
end for

and QP delta value as in [11]. Besides, GStreamer [74] API is used for real-time video frame capture. In the cloud, we decode the video to frames using the VideoCapture function [75] in OpenCV. In order to improve the efficiency of the system, the various modules of VaBUS is running in pipelines and multi-threads. For deep learning inference in the cloud, the models are accelerated by TensorRT with FP16 precision. VaBUS is implemented in Python and the video encoder on Jetson is implemented in C++.

2) Tasks: To demonstrate the effectiveness of VaBUS, we evaluate the system with two different tasks: object detection (referred to as *OD*) and human keypoint detection (referred to as *KD*). For object detection, we use YOLOv3 [76] to detect two kinds of objects (i.e., persons and vehicles) and measure the accuracy by F1-score. Note that F1-score is a classification metric so we need to specify an IoU (i.e., Intersection over Union) threshold over which the bounding box detection is assumed as correct otherwise wrong before calculating the F1-score. For human keypoint detection, we use OpenPifPaf [77] and also measure the accuracy with F1-score. Similar to IoU in the object detection case, we use OKS (i.e., Object Keypoint Similarity) to determine whether the detected keypoint for a human is correct or not. Unless stated otherwise, we use the threshold of 0.5 for both IoU and OKS in our experiments. Although we use YOLOv3 and OpenPifPaf for the following experiments, the design of VaBUS is decoupled from the backend inference model, which means any other models can be plugged in for usage now that it finishes the same task.

3) Datasets: To evaluate VaBUS with various video illumination, resolution, object intensity, size, speed, etc, we use datasets from four public sources representing a number of real-world scenarios. (1) We obtain highway traffic videos from top results on YouTube [78] by searching the keyword 'highway traffic videos'. Videos not captured by surveillance cameras, e.g., dashcam videos, are removed manually.

TABLE I
SUMMARY OF DATASETS

Name	Task	Length (s)	# frames	# videos	# objs
YouTube	OD	7,157	201,768	7	2,264,926
VIRAT	OD	2,414	71,097	8	778,822
MuPoTs-3D	KD	89	2,669	4	8,370
Human3.6M	KD	743	22,291	5	23,163

TABLE II
OVERALL PERFORMANCE OF VABUS ON FOUR DATASETS

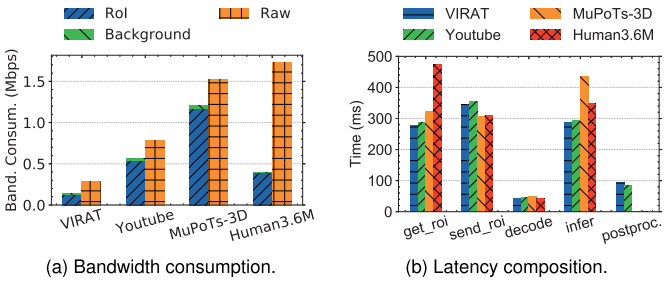
Task	Dataset	F1-score	Precision	Recall	Compress.	Latency
OD	VIRAT	88.9%	94.7%	85.2%	57.5%	1084 ms
	YouTube	92.6%	95.7%	91.6%	26.8%	1160 ms
KD	MuPoTs-3D	86.8%	86.5%	88.8%	25.0%	1154 ms
	Human3.6M	94.6%	94.4%	96.1%	76.9%	1200 ms

(2) We select a subset of videos from the VIRAT Video Dataset [79], which is a video surveillance dataset containing videos spanning across diverse resolution, background clutter, scenes and human activity/event categories. (3) We choose a subset of videos from the MuPoTs-3D Dataset [80], [81], which is a large scale dataset showing real images of sophisticated multi-person interactions and occlusions. (4) We choose a few videos from the Human3.6M Dataset [82], [83], which captures the poses of professional actors in various of scenarios, including discussing, eating, exercising, greeting, etc. The YouTube and VIRAT datasets are evaluated with object detection task, while the other two are with human keypoint detection task. Unless stated otherwise, we only use the first 10 minutes of the videos for the purpose of performance evaluation. A summary of datasets can be found in table I.

B. Overall Performance

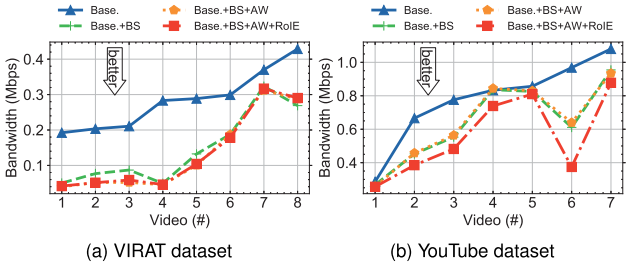
VaBUS is able to achieve high accuracy while considerably reducing the transmission size of video data. For reproducible experiments, we evaluate the overall performance of VaBUS on four datasets by extracting raw frames as camera input. Specifically, the frames are resized to 720×406 and fed into the edge device with the batch size of 15 and the frame rate of 15FPS. To measure the detection accuracy, we calculate the F1-score between the inference results of VaBUS and the ground truth results of extracted frames (which are resized to the same resolution) from videos. The reason to choose a frame rate of 15FPS is to ensure enough computing resources are available for other modules. The input frame rate is a parameter that can be adjusted according to the actual task to perform and more experiments about it can be found in Figure 13. Since the frames are processed in batches in favor of video encoding, the latency is calculated as the time difference between a batch of frames is ready and the detection results are postprocessed on the edge device.

Table II shows the overall performance of VaBUS with two tasks on the four datasets. Compression ratio (i.e., *Compress.*) is calculated as the ratio of the reduced transmission data size in VaBUS over the baseline, which represents the extent of these original frames are compressed to. In the object detection case, it can be observed that VaBUS achieves an



(a) Bandwidth consumption. (b) Latency composition.

Fig. 7. Bandwidth consumption and latency composition of VaBUS.



(a) VIRAT dataset (b) YouTube dataset

Fig. 8. Bandwidth consumption on two datasets for the object detection task (Base. stands for Baseline).

F1-score of 88.9% and 92.6% while reducing the transmission data size by 57.5% and 26.8% on the VIRAT and YouTube dataset respectively. In the human keypoint detection case, the F1-score is similar (86.8% and 94.6%) but the compression ratio of Human3.6M is much higher (76.9%). The precision is much higher than recall in the OD task, while they are more balanced in the KD task. The reason might be that the KD task is more complicated than the OD task. For OD, an object is less likely to be miss-detected once it is shown in the RoIs while a keypoint could be easily missed due to inaccurate location. The latency of OD and KD tasks on four datasets is around 1100ms (1149.5ms on average), which meets our latency requirement in §III-A. Note that the latency is actually controllable by tuning various configuration parameters including resolution, frame rate and batch size. More analysis about their effects on latency can be found in §IV-E *Impact of resolution* and *Impact of batch size and frame rate*. The results show that VaBUS is able to effectively reduce bandwidth consumption and maintain high accuracy and low latency at the same time.

Figure 7 shows the detailed composition of bandwidth consumption and latency of VaBUS. In Figure 7a, *RoI* represents encoded video size sent from Jetson to the cloud server after background subtraction and adaptive RoI encoding. *Background* is the total size of transferred background images. *Raw* shows the size of baseline approach where the frames are encoded without any processing. It is shown that *Background* occupies only a negligible amount of transferred size compared to *RoI*, which means the *background reconstruction* module causes few overhead on bandwidth. The bandwidth consumption of KD task (i.e., MuPoTs-3D and Human3.6M) is higher than OD task (i.e., VIRAT and YouTube). The reason is that human keypoint detection requires higher encoding quality in order to correctly detect the keypoints, while lower quality can be used to find bounding boxes in object detection. In Figure 7b, we show the latency experienced by each component of VaBUS. *get_roi* measures the preprocessing time for generating RoIs from input frames. *send_roi* is the total time of encoding RoIs to videos and streaming time from the edge device to cloud server. *decode* measures the time for decoding videos to frames in the cloud server, and *infer* measures the inference time of deep learning models. *postproc.* (short for postprocessing) mainly includes the time of background overlay (§III-B.2). Note that since background overlay is only implemented for the OD task in our experiments, the postprocess time is zero for the KD task. It can be observed that the latency is mainly caused by *get_roi* (340ms on average), *send_roi* (328ms on average) and *infer* (341ms on average). In our experiments, video encoding is already hardware-accelerated, but RoI generation is not optimized. Besides, deep learning model inference can be further accelerated using distributed or parallel computation techniques. Considering that the latency of RoI generation and deep learning inference latency account for 30.3% and 30.4% of the total latency respectively, an optimization room of 60.7% latency (681ms in total) can be explored. We leave the further optimization of VaBUS for future work.

C. Ablation Study

To understand each component's impact on VaBUS, we perform ablation study by examining each functioning part. We measure the accuracy (i.e., F1-score) of object detection task in four approaches: 1) the baseline solution (*Baseline*), 2) enabling background subtraction (*BS*) which includes background reconstruction and overlay, 3) enabling adaptive weighting (*AW*) and 4) enabling adaptive RoI encoding (*RoIE*). The baseline approach stands for the system using merely conventional codecs, which follows the straightforward pipeline of encoding, decoding and inferring all frames without any processing. Other experimental settings remain the same as §IV-B.

Table III shows the results of ablation study on VIRAT and YouTube datasets. It can be observed that enabling background subtraction (*BS*) substantially decreases the compression ratio (52.4% and 17.0%) while the accuracy only has a slight drop (6.4% and 2.4%). The *AW* module has different effects on different datasets (compared with *Baseline+BS*). On VIRAT dataset, the F1-score maintains about the same (88.4% and 88.1%) while the compression ratio is further increased (from 52.4% to 58.1%). On YouTube dataset, the compression ratio is rarely affected (17.0% and 16.9%) but F1-score is boosted from 92.7% to 93.6%. This is due to that the characteristics of the two datasets are different. YouTube has a larger object intensity so compression ratio is less affected, but the F1-score is increased since generated RoIs become more accurate. After adding the *RoIE* module, the compression ratio is increased for YouTube dataset from 16.9% to 26.8% while maintaining about the same for the VIRAT dataset (58.2% and 57.5%). The reason might be that the compression ratio of VIRAT is already very high (i.e., compressed by more than half compared to

TABLE III
PERFORMANCE OF VABUS ON TWO DATASETS FOR THE OBJECT DETECTION TASK

Dataset	Approaches	F1-score	Compress.	Latency	CPU
VIRAT	Baseline	94.8%	0%	635 ms	23.2%
	Baseline + BS	88.4%	52.4%	1048 ms	31.6%
	Baseline + BS + AW	88.1%	58.1%	1062 ms	31.7%
	Baseline + BS + AW + RoIE	88.9%	57.5%	1084 ms	31.3%
YouTube	Baseline	95.1%	0%	654 ms	25.1%
	Baseline + BS	92.7%	17.0%	1362 ms	44.4%
	Baseline + BS + AW	93.6%	16.9%	1268 ms	42.5%
	Baseline + BS + AW + RoIE	92.6%	26.8%	1160 ms	36.3%

Baseline) and it's difficult for *RoIE* to find target low quality regions in order to further save bandwidth. The average latency for baseline approach is 644.5ms and adding the modules (i.e., *BS*, *AW* and *RoIE*) in VaBUS increases it to 1122ms, with a rise of 477.5ms on average. Besides, our proposed modules (i.e., *Baseline+BS+AW+RoIE*) require 8.1% and 11.2% more CPU resource (9.6% on average) than the baseline approach. Note that although Jetson Xavier NX is equipped with a 384-core NVIDIA Volta GPU, our implementation does not leverage the on-board GPU resources for two reasons: 1) Most surveillance cameras are manufactured with limited computing resources [14], [84], [85]. Therefore we assume GPU support is not available for a general design; 2) For system optimization on specific devices (e.g., accelerating image-related operations with GPU), we leave it for our future work.

To further demonstrate the improvements of VaBUS over the baseline approach (i.e., conventional codecs), we show the detailed bandwidth consumption for each approach on two datasets (i.e., VIRAT and YouTube) on the object detection task in Figure 8. By enabling background subtraction (*BS*) to eliminate fluctuations of background pixel values from the semantic level, substantial bandwidth can be saved for most of the videos. The *AW* module is designed to refine the RoIs generated by *BS*, therefore it has only slight improvements for bandwidth consumption. By enabling the *RoIE* module to assign different encoding quality across the frame with the help of historical object size information, the bandwidth consumption is further dropped. It can be observed that there is a huge variance in the compression ratio for different videos (from 5.5% to 83.8%) on object detection tasks. The reason is that the actual compression ratio depends on the percentage of foreground area as well as the object size, and detailed analysis can be found in §IV-E *Impact of video genres*.

D. Offline Estimation Performance

The *Mapping Estimator* (ME) module of VaBUS is designed to estimate inference results of current batch of frames based on last batch. Figure 9 shows the performance of ME module. In Figure 9a, we compare the F1-score and estimation latency of six approaches on a sample dataset to estimate 20 batches of frames (i.e., 300 testing frames in total). The *Worst* curve is the lower bound of accuracy when there's no inference result (the F1-score is greater than zero since some frames contain no object to detect). The *Best* curve refers to the upper bound of accuracy which uses the detection results

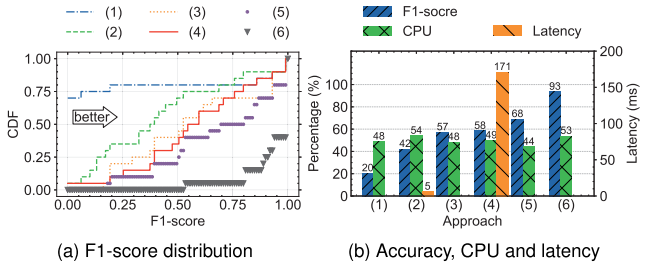


Fig. 9. Performance of mapping estimator (ME) module. 1-Worst, 2-ME, 3-MvOT, 4-OF, 5-MvOT+ME, 6-Best.

as estimation. The *ME* curve represents our proposed mapping estimator module in VaBUS. The *MvOT* curve reflects results of the motion vector-based object tracking method in [11]. In the optical flow-based (referred to as *OF*) approach, we use the dense optical flow algorithm proposed by Gunnar Farneback [86] to calculate the optical flow for all points in the frame. Since the motion vectors can be obtained in the video encoding phase with almost no computation overhead, we further boost the performance of *ME* based on estimation results of *MvOT* (i.e., *MvOT+ME*). From Figure 9a, it can be observed that there are no statistically significant differences between *MvOT* and *OF*. *ME*, though achieving lower accuracy (42%) than *MvOT* (57%) and *OF* (58%) when solely used, can be used to boost the accuracy a lot when combined with *MvOT* (i.e., 68% in the *MvOT+ME* case). In Figure 9b, it is shown the latency of *ME* is only 5ms per image on average while *OF* is as large as 171ms. Besides, *ME* can be faster when fewer objects need to be estimated, while *OF* requires computation of the whole image even though there is only one object to estimate.

E. Sensitivity to Application Settings

1) Impact of Video Genres: Figure 10 shows per-video bandwidth consumption and F1-score of fours datasets on the object detection and human keypoint detection task. In Figure 10a, it can be seen that different dataset has different distribution of compression ratio. VaBUS saves the most bandwidth on Human3.6M while shows the least on MuPoTs-3D. From Figure 10b, it can be observed that although the F1-score achieves around 90%, the compression ratio shows a large variability between videos (from −6.8% to 83.8%, −6.8% means the transmission data size is increased by 6.8%). The reason is that the compression ratio of VaBUS mainly depends

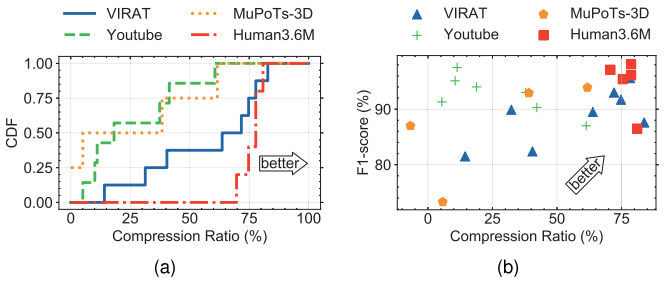


Fig. 10. Distribution of per-video bandwidth consumption and F1-score for four datasets. (a) CDF of per-video compression ratio. (b) Compression ratio and F1-score distribution.

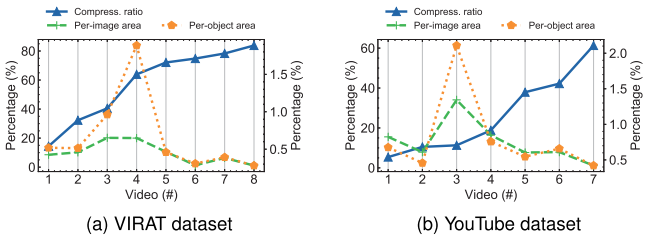


Fig. 11. Effects of per-image foreground area and per-object foreground area on compression ratio for two datasets with the object detection task.

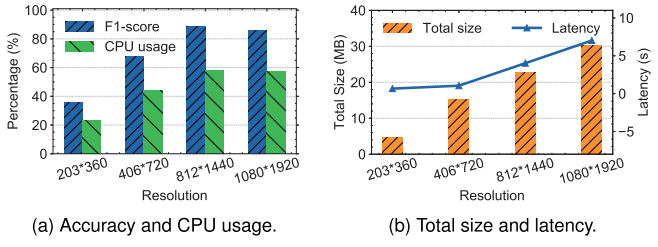


Fig. 12. System performance under various input resolution.

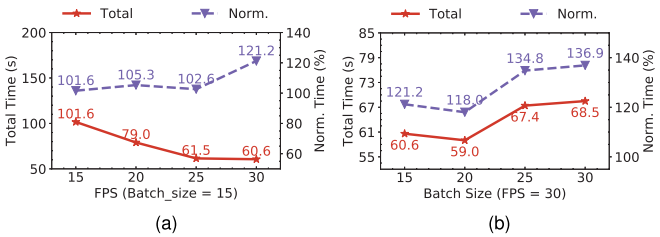


Fig. 13. Time consumption under various batch size and frame rate. (Norm.: Normalized). (a) Under various FPS. (b) Under various batch size.

on the area of background filtered by the *diff detector* module (§III-C.1). For videos that show changes only in a small part of the whole frames, VaBUS is able to save as large as 83.8% bandwidth. But for videos with moving objects distributed across the whole frame, VaBUS has few gains and may even cost slightly more bandwidth than the baseline approach due to the transmission of background image.

To elaborate on the cause of the large variance on the compression ratio between videos, we show the effects of per-image foreground area and per-object foreground area on compression ratio for two datasets with the object detection task in Figure 11. Per-image foreground area is calculated as the percentage of the covered area of all target objects in one image on average. It represents the minimal RoI region we need to generate, and usually smaller per-image foreground area means higher compression ratio (i.e., by subtracting the background). Per-object foreground area is calculated as the percentage of the covered area of a single object over the whole image on average. It represents the object size in the video, and usually larger per-object foreground area means higher compression ratio (i.e., by assigning lower encoding quality with the *RoIE* module). The compression ratio is influenced by both per-image area and per-object area. When per-object area is on the same level (i.e., the object size is about the same, e.g., Video #1,2,5,6,7,8 in Figure 11a and Video #1,2,4,5,6,7 in Figure 11b), the compression ratio increases along with the decrease of per-image area. However, when the per-object area is much larger (i.e., object size is much larger, e.g., Video #3,4 in Figure 11a and Video #3 in Figure 11b), the compression ratio can be higher even though the per-image area is larger. In this case, the per-object area dominates the performance on compression ratio.

2) Impact of Resolution: Frame resolution is a key factor to consider when streaming videos. Figure 12 shows the impact of resolution on accuracy (i.e., F1-score), CPU usage, total transmission size (i.e., *total size*) and latency respectively for the object detection task. From Figure 12a, it can be observed that increasing the resolution from 203×360 to 812×1440 improves the accuracy by 53% while a little drop (3.1%) occurs when the resolution further rises to 1080×1920. The reason might be that the system is too busy with computationally intensive tasks such that it fails to get timely feedback (e.g., the update of background image) from the cloud server (Fig 12b). Besides, the input size of our object detection model is set to be 416×416. Resolution larger than this size has no benefit during inference. The increase of resolution also significantly raises the transmission data size. From 203×360 to 1080×1920, the total transmission size is increased by 5.36 times (Fig 12b). When resolution is 203×360 or 406×720, the system runs in real time with a latency of only around one second, while the latency surges when the resolution is higher (Fig 12b). Similarly, the CPU usage is saturated when resolution is higher than or equal to 812×1440. This means increasing the resolution does not necessarily improve the system performance. It needs to be adjusted in accordance with available computing resources on the edge device and the actual application scenarios.

3) Impact of Batch Size and Frame Rate: Batch size and frame rate are two key parameters controlling the latency of VaBUS. The first one determines how fast the system consumes the incoming data and the second one controls the data input rate. Figure 13 shows the time consumed for streaming 1,500 frames under various of batch size and frame rate. *Total Time* measures the throughput of the system by time elapsed for processing (including encoding, streaming and inference, etc) all the frames, while *Norm. Time* (short for *Normalized Time*) is the ratio of *Total Time* to purely streaming time with corresponding frame rate (i.e., closer to 100% means lower latency). In Figure 13a, it can be observed that elevating frame rate from 15FPS to 30FPS increases the throughput of

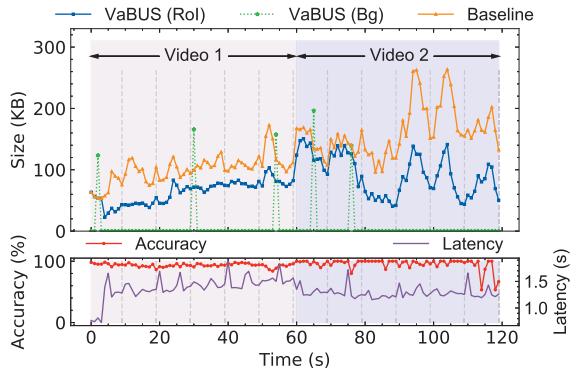


Fig. 14. Effects of background change to VaBUS.

the system and shortens the total running time (i.e., *Total Time*). Besides, the system is working in real time when frame rate is less or equal to 25FPS while significant latency is observed when frame rate reaches 30FPS (i.e., 121.2% normalized time). When frame rate is fixed to be 30FPS (Figure 13b), simply increasing batch size does not improve the performance. On the contrary, the total time increases when batch size is lifted to be more than 20. And the latency is also increased (i.e., 134.8% and 136.8% normalized time). To this end, the batch size and frame rate shall be carefully chosen according to the application scenario to consistently meet the latency requirement.

4) Impact of Background Change: VaBUS is able to automatically reconstruct the background image in the cloud and send updates to the edge when necessary. In order to investigate the effects of video background changes to VaBUS, we manually merge two one-minute clip from two different videos to simulate the sudden change of video background. The results are shown in Figure 14. It can be observed that the transmission data size (i.e., *Size*) of baseline approach and VaBUS is the same in the first four seconds, since there are no background learned to be subtracted on the edge. When the background is successfully sent to the edge on the fifth second, the RoI size is substantially reduced and keeps lower than the baseline approach hereafter. When the background changes (i.e., from Video 1 to Video 2 on the 60th second), the transmission data size surges again, and then two new background images are reconstructed and sent to the edge. For every certain periods of time (10 seconds in this case, shown as the vertical lines), the RoI encoding parameters are updated to the edge for adaptive RoI encoding (§III-C.3). This is the reason why the transmission data size of VaBUS for initial frames from Video 2 is lower than the baseline approach. Over the time frame of 2 minutes, VaBUS transferred only five background images from the cloud to the edge and the sent RoI size is consistently lower than the baseline approach. Besides, changing the background does not impact the accuracy as shown in the lower part of Figure 14. This can be explained by that when the background is changed, the edge would fail to subtract it from the input frames and streams the original frames to the cloud for inference. The accuracy continuously keeps high (94.4% on average) and latency stays between 1 second to 1.5 second, showing the superior performance of VaBUS.

5) Impact of Inference Model: To show the impact of inference model on the accuracy and compression ratio of VaBUS, we test three different deep learning models (i.e., YOLOv3, EfficientDet-D0 and EfficientDet-D3) on a randomly selected video from the YouTube dataset for the object detection task. EfficientDet-D0 and EfficientDet-D3 are both from the EfficientDet [8] series of object detection models, and both models are chosen to realize real-time inference with TensorRT acceleration on our cloud server. To eliminate the difference between different models, we use the inference results of original frames for each model as the corresponding ground truth for accuracy evaluation. Results show that EfficientDet-D0 achieves the accuracy of 92.7% and compression ratio of 17.3%, while EfficientDet-D3 achieves the accuracy of 92.9% and compression ratio of 20.7%. These results are similar to YOLOv3 where the accuracy is 94.0% and compression ratio is 18.8%. This implies that the benefits of VaBUS are agnostic to the inference model on the cloud.

F. Comparison With Other Approaches

Unlike other approaches that work for general video datasets, VaBUS focuses on videos from surveillance cameras and leverages the techniques of background subtraction and adaptive RoI encoding to significantly reduce bandwidth consumption. We compare the accuracy and latency of VaBUS under various bandwidth budgets as well as the CPU usage with three other information pruning approaches (i.e., EAAR [11], DDS [12], Elf [33]) and the traditional codec (i.e., baseline). The available bandwidth between the edge and cloud is controlled by *wondershaper* [87] which uses *tc* [88] under the hood. Note that VaBUS lies in the design space of information pruning and is independent with video analytics systems focusing on other aspects, e.g., bandwidth adaption or inference acceleration. For DDS, we use QP 26 for high-quality feeds and QP 36 for low-quality feeds as in the original paper, which we observed to achieve a good tradeoff between bandwidth consumption and latency. For EAAR, we only use the dynamic RoI encoding module (other modules like parallel streaming and inference are independent with VaBUS) and enlarge the RoIs from last detection result by one macroblock (as in the original paper). For Elf, we train an attention-based LSTM offline to converge on a set of different videos from the same dataset (i.e., VIRAT) for recurrent RoI prediction on the cloud server.

Figure 15 shows the bandwidth-accuracy and bandwidth-latency tradeoff for five approaches (i.e., Baseline, EAAR, DDS, Elf and VaBUS) on the VIRAT dataset with the object detection (OD) task using a 2-minute clip for each video. It can be observed that VaBUS achieves better bandwidth-accuracy and bandwidth-latency tradeoff compared to other four approaches. When the bandwidth is adequate (e.g., larger than 300Kbps), the accuracy and latency of VaBUS is similar to other approaches. However, with the gradual drop of available bandwidth, VaBUS outperforms other four approaches on both accuracy and latency, which

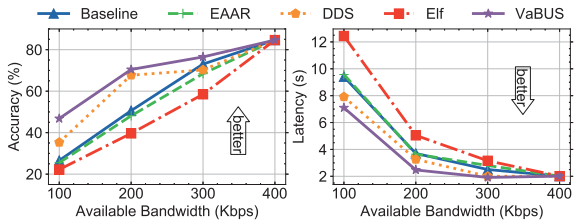


Fig. 15. Tradeoff of bandwidth consumption and accuracy/latency for Baseline, EAAR, DDS, Elf and VaBUS under various bandwidth budgets.

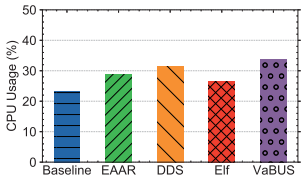


Fig. 16. CPU usage for Baseline, EAAR, DDS, Elf and VaBUS.

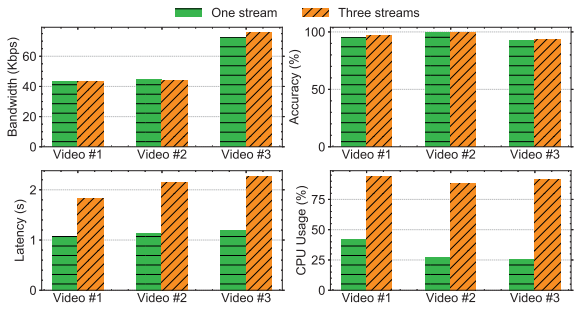


Fig. 17. Performance change when scaling to multiple streams.

shows the advantage of VaBUS's bandwidth saving. For CPU usage (Fig. 16), VaBUS is similar to other approaches with only negligible overhead.

In a nutshell, VaBUS achieves better bandwidth and accuracy/latency tradeoff than state-of-the-art video analytics systems with minimal CPU overhead.

G. Scaling to Multiple Streams

VaBUS builds on edge computing, which brings storage and computing closer to data sources. Instead of establishing a one-to-one connection between the device (i.e., camera) and the cloud to process a single video stream, VaBUS supports one-to-many connections for simultaneous analysis of multiple video streams. To demonstrate the performance of VaBUS when processing multiple streams, we evaluate the bandwidth consumption, accuracy, latency and CPU usage of streaming one video as well as streaming three videos simultaneously. The experiments are conducted on a sample video from the VIRAT dataset for the object detection task, and the results are shown in Figure 17. It can be observed that scaling from one stream to three streams has almost no impact on the bandwidth consumption and accuracy. However, the latency of analyzing three streams (2080ms) is higher than one stream (1132ms) due to the increased computation burden on the edge side. Similarly, the CPU usage for analyzing three streams (91.2%) is much higher than one stream (31.6%). The latency and the CPU usage can be further reduced from the system

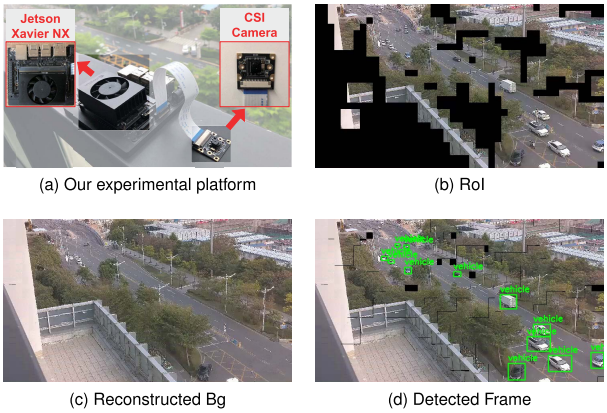


Fig. 18. Real-world deployment and qualitative results of VaBUS.

implementation aspect, which we leave for future work. This shows that VaBUS successfully scales to analyses of multiple video streams to better suit the scenario of edge-cloud video analytics.

H. Real-World Deployment

To show the effectiveness under real-world application scenarios, we test VaBUS's performance under practical settings. The edge device, i.e., Jetson Xavier NX, is connected with a CSI camera to capture video frames in resolution of 406×720 and 15FPS. The cloud server is deployed behind a remote VPS located in another city. The bandwidth between Jetson and cloud server is 10 Mbps, measured by *iperf* [89]. Figure 18a shows the actual deployment of our experimental hardware platform. Figure 18b represents the RoI sent from Jetson after background subtraction, where we can see a large number of pixels are erased and filled with black. The reconstructed background image is shown in Figure 18c, which is a clean street image. In the cloud server, we combine the RoI and background image to recover the original frame and perform inference, as shown in Figure 18d. It can be seen that the vehicles are correctly detected. The latency is 1170ms and CPU usage is 35.2%, which is consistent with our previous experiments (e.g., Table III and Figure 12). The results show VaBUS is able to effectively perform deep learning inference tasks with insignificant resource overhead under real-world settings.

V. Limitation and Future Work

A. Surveillance Camera

The amount of bandwidth saved by VaBUS relies on the assumption that the video comes from a surveillance camera. Unlike other video analytics system, e.g., DDS and EAAR, which is applicable for all videos, VaBUS focuses on a specific set of it. Although surveillance videos with fixed background account for only a proportion of existing video streams, VaBUS degrades to RoI encoded streams without background subtraction when running on videos captured by moving cameras, e.g., dashcams or drones. And slightly more bandwidth might be consumed due to the transmission of background image.

B. Tradeoff of Bandwidth Saving

Although VaBUS is able to save a large amount of bandwidth with acceptable latency of around one second, there is a tradeoff between bandwidth consumption and latency. Generating RoIs on the edge device as well as processing frames in batches inevitably increase the latency. The overhead VaBUS imposes on the system may be overwhelming for low-end edge devices with fewer available computing resources.

C. Future Work

VaBUS is designed to improve a single aspect of the video analytics pipeline, i.e., saving bandwidth by transporting only changed foreground regions of the frames. Therefore, it can be combined with other independent systems to further improve the performance, e.g., integrating AWStream [7] to dynamically adjust streaming settings and adapt to available bandwidth. Besides, more optimization can be applied on the edge device to utilize the limited resources more efficiently, e.g., implementing the system in a more high-performance language like C++ instead of Python, and moving image-related operations from CPU to the unoccupied GPU.

VI. Conclusion

Semantic compression has become essential in the deployment of real-time video analytics applications, and this work shows that huge bandwidth savings can be realized by sending only foreground of the video frames from the edge to the cloud. We have demonstrated a concrete design of VaBUS to leverage the rich contextual information of video feeds: learn the background information in the cloud, generate accurate RoI regions on the edge and utilize context-dependent characteristics for lightweight experience-drive learning. We implement the task-oriented communication system with commodity hardware, and demonstrate that 25.0%-76.9% bandwidth consumption can be saved with less than 10% accuracy degraded and about one second latency. We believe the development of such system will not only benefit edge-cloud real-time video analytics, but also facilitate a wide range of video-related applications.

References

[1] J. Barthélemy, N. Verstaevel, H. Forehead, and P. Perez, "Edge-computing video analytics for real-time traffic monitoring in a smart city," *Sensors*, vol. 19, no. 9, pp. 1–23, 2019.

[2] I. Olatunji and C. H. Cheng, "Video analytics for visual surveillance and applications: An overview and survey," in *Machine Learning Paradigms*. Cham, Switzerland: Springer, Jul. 2019, pp. 475–515.

[3] 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.

[4] G. Ananthanarayanan et al., "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, Oct. 2017.

[5] S. Jain, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. Gonzalez, "Scaling video analytics systems to large camera deployments," in *Proc. 20th Int. Workshop Mobile Comput. Syst. Appl.*, Santa Cruz, CA, USA, Feb. 2019, pp. 9–14.

[6] F. Romero, M. Zhao, N. J. Yadwadkar, and C. Kozyrakis, "Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines," in *Proc. ACM Symp. Cloud Comput.*, Seattle, WA, USA, Nov. 2021, pp. 1–17.

[7] 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.*, Budapest, Hungary, Aug. 2018, pp. 236–252.

[8] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 10781–10790.

[9] *World's Smallest AI Supercomputer: Jetson Xavier NX | NVIDIA*. Accessed: Jul. 13, 2022. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/

[10] C. H. C. Bumgardner, *Making Sense of Edge Computing*. New York, NY, USA: Manning Publications Co, 2021.

[11] 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.*, Los Cabos, Mexico, Aug. 2019, pp. 1–16.

[12] 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.

[13] C. Canel et al., "Scaling video analytics on constrained edge nodes," in *Proc. Mach. Learn. Syst.*, Santa Clara, CA, USA, Mar. 2019, pp. 406–417.

[14] 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.

[15] J. Yi, S. Choi, and Y. Lee, "Eagleeye: Wearable camera-based person identification in crowded urban spaces," in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, London, U.K., Apr. 2020, pp. 1–14.

[16] *Singapore to Double Police Cameras to More Than 200, 000 Over Next Decade | Reuters*. Accessed: Mar. 31, 2022. [Online]. Available: https://www.reuters.com/world/asia-pacific/singapore-double-police-cameras-more-than-200000-over-next-decade-2021-08-04/

[17] A. Mohan, K. Gauen, Y.-H. Lu, W. W. Li, and X. Chen, "Internet of video things in 2030: A world with many cameras," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Baltimore, United States, May 2017.

[18] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[19] C. Zhang, Q. Cao, H. Jiang, W. Zhang, J. Li, and J. Yao, "A fast filtering mechanism to improve efficiency of large-scale video analytics," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 914–928, Jun. 2020.

[20] S. Jain et al., "Spatula: Efficient cross-camera video analytics on large camera networks," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, San Jose, CA, USA, Nov. 2020, pp. 110–124.

[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.*, Seoul, South Korea, Nov. 2015, pp. 155–168.

[22] S. Jain et al., "ReXCam: Resource-efficient, cross-camera video analytics at enterprise scale," 2018, *arXiv:1811.01268*.

[23] Z. Qin, X. Tao, J. Lu, W. Tong, and G. Ye Li, "Semantic communications: Principles and challenges," 2021, *arXiv:2201.01389*.

[24] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[25] *VPI—Vision Programming Interface: Background Subtractor*. Accessed: Jul. 14, 2022. [Online]. Available: https://docs.nvidia.com/vpi/algo_background_subtractor.html

[26] M. Khani, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh, "Real-time video inference on edge devices via adaptive model streaming," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 4572–4582.

[27] R. Mehta and R. Shorey, "DeepSplit: Dynamic splitting of collaborative edge-cloud convolutional neural networks," in *Proc. Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Bengaluru, India, Jan. 2020, pp. 720–725.

[28] A. H. Jiang et al., "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *2018 USENIX Annu. Tech. Conf. (USENIX ATC)*, Boston, MA, USA, Jul. 2018, pp. 29–42.

[29] 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. 1586–1597, Mar. 2017.

[30] 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)*, Boston, MA, USA, Mar. 2017, pp. 377–392.

[31] 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.*, Budapest, Hungary, Aug. 2018, pp. 253–266.

[32] 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.*, Santa Clara, CA, USA, Aug. 2022, pp. 450–466.

[33] 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.*, New Orleans, LA, USA, Oct. 2021, pp. 201–214.

[34] B. Garcia-Garcia, T. Bouwmans, and A. J. R. Silva, "Background subtraction in real applications: Challenges, current models and future directions," *Comput. Sci. Rev.*, vol. 35, Feb. 2020, Art. no. 100204.

[35] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Proc. 17th Int. Conf. Pattern Recognit. (ICPR)*, Cambridge, U.K., Aug. 2004, pp. 28–31.

[36] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *Proc. Eur. Conf. Comput. Vis.*, Berlin, Germany, Apr. 2000, pp. 751–767.

[37] B. Han and L. S. Davis, "Density-based multifeature background subtraction with support vector machine," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 5, pp. 1017–1023, May 2011.

[38] R. Trabelsi, I. Jabri, F. Smach, and A. Bouallegue, "Efficient and fast multi-modal foreground-background segmentation using RGBD data," *Pattern Recognit. Lett.*, vol. 97, pp. 13–20, Oct. 2017.

[39] T. Akilan, Q. M. J. Wu, and Y. Yang, "Fusion-based foreground enhancement for background subtraction using multivariate multi-model Gaussian distribution," *Inf. Sci.*, vols. 430–431, pp. 414–431, Mar. 2018.

[40] M. Babaee, D. T. Dinh, and G. Rigoll, "A deep convolutional neural network for video sequence background subtraction," *Pattern Recognit.*, vol. 76, pp. 635–649, Apr. 2018.

[41] M. Braham and M. Van Droogenbroeck, "Deep background subtraction with scene-specific convolutional neural networks," in *Proc. Int. Conf. Syst., Signals Image Process. (IWSSIP)*, Bratislava, Slovakia, May 2016, pp. 1–4.

[42] D. Sakkos, H. Liu, J. Han, and L. Shao, "End-to-end video background subtraction with 3D convolutional neural networks," *Multimedia Tools Appl.*, vol. 77, no. 17, pp. 23023–23041, 2018.

[43] Y. Yang, T. Zhang, J. Hu, D. Xu, and G. Xie, "End-to-end background subtraction via a multi-scale spatio–temporal model," *IEEE Access*, vol. 7, pp. 97949–97958, 2019.

[44] M. C. Bakkay, H. A. Rashwan, H. Salmane, L. Khoudour, D. Puig, and Y. Ruichek, "BSCGAN: Deep background subtraction with conditional generative adversarial networks," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, Athens, Greece, Oct. 2018, pp. 4018–4022.

[45] J. Gracewell and M. John, "Dynamic background modeling using deep learning autoencoder network," *Multimedia Tools Appl.*, vol. 79, no. 7, pp. 4639–4659, 2020.

[46] F. Afsana, M. Paul, M. Murshed, and D. Taubman, "Efficient high-resolution video compression scheme using background and foreground layers," *IEEE Access*, vol. 9, pp. 157411–157421, 2021.

[47] L. Wu, K. Huang, H. Shen, and L. Gao, "Foreground-background parallel compression with residual encoding for surveillance video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 7, pp. 2711–2724, Jul. 2021.

[48] X. Zhang, T. Huang, Y. Tian, and W. Gao, "Background-modeling-based adaptive prediction for surveillance video coding," *IEEE Trans. Image Process.*, vol. 23, no. 2, pp. 769–784, Feb. 2014.

[49] C. Ma, D. Liu, X. Peng, L. Li, and F. Wu, "Traffic surveillance video coding with libraries of vehicles and background," *J. Vis. Commun. Image Represent.*, vol. 60, pp. 426–440, Apr. 2019.

[50] X. Zhang, L. Liang, Q. Huang, Y. Liu, T. Huang, and W. Gao, "An efficient coding scheme for surveillance videos captured by stationary cameras," *Proc. SPIE*, vol. 7744, pp. 729–738, Aug. 2010.

[51] A. A. M. Al-Saffar, H. Tao, and M. A. Talab, "Review of deep convolution neural network in image classification," in *Proc. Int. Conf. Radar, Antenna, Microw., Electron., Telecommun. (ICRAMET)*, Jakarta, Indonesia, Oct. 2017, pp. 26–31.

[52] W. Gu, S. Bai, and L. Kong, "A review on 2D instance segmentation based on deep neural networks," *Image Vis. Comput.*, vol. 120, Apr. 2022, Art. no. 104401.

[53] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.

[54] M. M. Kasar, D. Bhattacharyya, and T. Kim, "Face recognition using neural network: A review," *Int. J. Secur. Appl.*, vol. 10, no. 3, pp. 81–100, 2016.

[55] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "A comprehensive survey of deep learning for image captioning," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–36, Nov. 2019.

[56] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, "Multiple object tracking: A literature review," *Artif. Intell.*, vol. 293, Apr. 2021, Art. no. 103448.

[57] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Phoenix, AZ, USA, Sep. 2016, pp. 3464–3468.

[58] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Beijing, China, Sep. 2017, pp. 3645–3649.

[59] L. Zhang and L. van der Maaten, "Structure preserving object tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, Jun. 2013, pp. 1838–1845.

[60] W. Hu, X. Li, W. Luo, X. Zhang, S. J. Maybank, and Z. Zhang, "Single and multiple object tracking using log-Euclidean Riemannian subspace and block-division appearance model," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 12, pp. 2420–2440, Dec. 2012.

[61] T. Pathak, V. Patel, S. Kanani, S. Arya, P. Patel, and M. I. Ali, "A distributed framework to orchestrate video analytics across edge and cloud: a use case of smart doorbell," in *Proc. 10th Int. Conf. Internet Things*, New York, NY, USA, Oct. 2020, pp. 1–8.

[62] X. Sevillano, E. Màrmol, and V. Fernandez-Arguedas, "Towards smart traffic management systems: Vacant on-street parking spot detection based on video analytics," in *Proc. 17th Int. Conf. Inf. Fusion (FUSION)*, Salamanca, Spain, Jul. 2014, pp. 1–8.

[63] G. Grassi, K. Jamieson, P. Bahl, and G. Pau, "Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, New York, NY, USA, Oct. 2017, pp. 1–14.

[64] N. Agarwal and R. Netravali, "Boggart: Towards general-purpose acceleration of retrospective video analytics," 2021, *arXiv:2106.15315*.

[65] Y. J. Liang and B. Girod, "Network-adaptive low-latency video communication over best-effort networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 1, pp. 72–81, Jan. 2006.

[66] Z. Zivkovic and F. van der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern Recognit. Lett.*, vol. 27, no. 7, pp. 773–780, 2006.

[67] *Backgroundsubtractorknn Class Reference*. Accessed: Mar. 31, 2022. [Online]. Available: https://docs.opencv.org/4.x/db/d88/classcv_1_1BackgroundSubtractorKNN.html

[68] C. Xiao et al., "Method for detecting and tracking foreign objects in substation videos based on embedded AI," in *Proc. Power Syst. Green Energy Conf. (PSGEC)*, Shanghai, China, Aug. 2021, pp. 583–587.

[69] A. K. Ramasubramanian, R. Mathew, I. Preet, and N. Papakostas, "Review and application of edge AI solutions for mobile collaborative robotic platforms," *Proc. CIRP*, vol. 107, pp. 1083–1088, Jan. 2022.

[70] Y. Kortli, S. Gabsi, L. F. C. L. Y. Voon, M. Jridi, M. Merzougui, and M. Atri, "Deep embedded hybrid CNN–LSTM network for lane detection on NVIDIA Jetson Xavier NX," *Knowl.-Based Syst.*, vol. 240, Mar. 2022, Art. no. 107941.

[71] S. V.-U. Ha, N. M. Chung, T.-C. Nguyen, and H. N. Phan, "Tiny-PIRATE: A tiny model with parallelized intelligence for real-time analysis as a traffic counter," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Nashville, TN, USA, Jun. 2021, pp. 4119–4128.

[72] *Jetson Linux API Reference: Multimedia APIs | NVIDIA Docs*. Accessed: Mar. 31, 2022. [Online]. Available: https://docs.nvidia.com/jetson/l4t-multimedia/mmapi_group.html

[73] *Jetson Linux API Reference: Nvvideoencoder Class Reference | NVIDIA docs*. Accessed: Mar. 31, 2022. [Online]. Available: https://docs.nvidia.com/jetson/l4t-multimedia/classNvVideoEncoder.html#a395ee26e60ea41b374774e2cc996ce55

[74] *NVIDIA Jetson Linux Developer Guide: Multimedia | NVIDIA Docs*. Accessed: Mar. 31, 2022. [Online]. Available: https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/accelerated_gstreamer.html

[75] *Videocapture Class Reference*. Accessed: Mar. 31, 2022. [Online]. Available: https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html

[76] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[77] S. Kreiss, L. Bertoni, and A. Alahi, "OpenPifPaf: Composite fields for semantic keypoint detection and spatio–temporal association," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 1–14, Aug. 2021.
[78] *Highway Traffic Videos—Youtube*. Accessed: Mar. 31, 2022. [Online]. Available: https://www.youtube.com/results?search_query=highway+traffic+videos
[79] S. Oh et al., "A large-scale benchmark dataset for event recognition in surveillance video," in *Proc. CVPR*, Colorado Springs, CO, USA, Jun. 2011, pp. 3153–3160.
[80] D. Mehta et al., "Single-shot multi-person 3D pose estimation from monocular RGB," in *Proc. Int. Conf. 3D Vis. (3DV)*, Verona, Italy, Sep. 2018, pp. 120–130.
[81] D. Mehta et al., "Monocular 3D human pose estimation in the wild using improved CNN supervision," in *Proc. Int. Conf. 3D Vis. (3DV)*, Qingdao, China, Oct. 2017, pp. 506–516.
[82] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6M: Large scale datasets and predictive methods for 3D human sensing in natural environments," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 7, pp. 1325–1339, Jul. 2013.
[83] C. Ionescu, F. Li, and C. Sminchisescu, "Latent structured models for human pose estimation," in *Proc. Int. Conf. Comput. Vis.*, Barcelona, Spain, Nov. 2011, pp. 2220–2227.
[84] *Wyze Home Security Camera*. Accessed: Jul. 12, 2022. [Online]. Available: https://www.safehome.org/home-security-cameras/wyze/
[85] *Intelligent Machine Vision Smart Cameras for OEM*. Accessed: Jul. 12, 2022. [Online]. Available: https://fishersmith.co.uk/components/smart-cameras/
[86] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Scandin. Conf. Image Anal.*, Halmstad, Sweden, Jun. 2003.
[87] *Wondershaper*. Accessed: Sep. 29, 2022. [Online]. Available: https://github.com/magnific0/wondershaper
[88] *TC*. Accessed: Sep. 29, 2022. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html
[89] *iPerf—The TCP, UDP and SCTP Network Bandwidth Measurement Tool*. Accessed: Mar. 31, 2022. [Online]. Available: https://iperf.fr/

Hanling Wang received the B.S. degree in electronic information science and technology from Central South University, China, in 2017, and the M.S. degree in data science and information technology from Tsinghua University, China, in 2020, where he is currently pursuing the Ph.D. degree in computer science and technology under the joint program with the Peng Cheng Laboratory. His main research interests include video analytics, edge computing, semantic communication, and deep learning.

Qing Li (Member, IEEE) received the B.S. degree in computer science and technology from the Dalian University of Technology, Dalian, China, in 2008, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2013. He is currently an Associate Researcher with the Peng Cheng Laboratory, Shenzhen, China. His research interests include network function virtualization, in-network caching/computing, intelligent self-running networks, and edge computing.

Heyang Sun received the B.S. degree in software engineering from Southeast University, China, in 2022. His main research interests include machine learning systems, privacy preservation, and big data processing.

Zuozhou Chen received the B.S. degree from the Hunan University of Science and Technology in 2013. He is currently a Streaming Media Engineer with the Peng Cheng Laboratory, Shenzhen, China. His research interests include multimedia streaming, video coding, and distributed systems.

Yingqian Hao is currently pursing the B.S. degree in software engineering with the College of Software, Jilin University, China. Her main research interests are computer networks and deep learning.

Junkun Peng received the B.S. degree in information management and information system from Shanghai University, Shanghai, China, in 2015. He is currently pursuing the master's degree in computer technology with Tsinghua University. His research interests include in-network caching/computing, real-time video transmission/analysis, and smart drone/robot swarm.

Zhenhui Yuan received the B.S. degree in software engineering from Wuhan University, China, in 2008, and the Ph.D. degree in EE from Dublin City University, Ireland, in 2012. His research interests lie in wireless communication systems for mobile devices, robots, and vehicles. From 2012 and 2014, he was a Post-Doctoral Researcher funded by ERICSSON and Enterprise Ireland and a Visiting Scholar at the Network Research Laboratory (led by Prof. Mario Gerla), UCLA, USA, in 2014. He was a 3GPP Delegate on 5G at Huawei (Shanghai) Technologies Compnay Ltd., from 2014 to 2015. From 2015 to 2019, he was with the Key Laboratory of RF Circuits and Systems (Ministry of Education), Hangzhou Dianzi University, as an Associate Professor. He co-founded RobSense (Hangzhou) Technology Company Ltd., which designs and produces UAV flight controllers and IoT gateway. RobSense has received over €1m in venture investment since 2015. From 2019 to 2020, he was an Assistant Professor at Maynooth University, Ireland. In 2020, he joined Northumbria University, U.K., as a Senior Lecturer. He won the best paper awards at IEEE ICCRE 2016 and IEEE BMSB 2014. He is a Committee Member of IEEE P1954. He serves as an Associate Editor for IEEE ACCESS and IEEE TRANSACTIONS ON CONSUMER ELECTRONICS.

Junsheng Fu received the bachelor's degree in telecommunication from Hangzhou Dianzi University, China, the master's degree in signal processing from the Tampere University of Technology, Finland, and the Ph.D. degree in computer vision from Tampere University, Finland. Previously, he worked as a Computer Vision Researcher at Nokia Research, Tampere, Finland. Currently, he works as a Researcher at Zenseact (a Volvo-car-owned autonomous driving software company), Gothenburg Sweden, and focuses on the research tasks of vehicle localization and road estimation in autonomous driving.

Yong Jiang (Member, IEEE) received the B.S. and Ph.D. degrees from Tsinghua University, Beijing, China, in 1998 and 2002, respectively. He is currently a Full Professor with the Division of Information Science and Technology, Tsinghua Shenzhen International Graduate School, Shenzhen, China, and the Peng Cheng Laboratory, Department of Mathematics and Theories, Shenzhen. He mainly focuses on the future internet architecture, the Internet of Things, edge computing, and AI for networks.