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 semantics 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 challenges need to be addressed, i.e., 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. 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. Finally, deep learning models for video analytics need massive computing resources that are usually limited on cameras.

To address these problems, the typical real-time video analytics pipelines adopt an edge-cloud collaborative approach, i.e., a camera only sends partial video content to the cloud server, which runs deep learning models and returns the inference results [4]–[8]. For these solutions, one of the key factors to consider is the nontrivial bandwidth consumption. According to [9], 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 [10]. Considering a bitrate of 2 Mbps for a typical HD video feed compressed by the H.264 codec [11], 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 techniques that improve user-perceived Quality of Experience (QoE), inference-targeted video streaming techniques (i.e., machine-centric) allow discarding more irrelevant information for significant bandwidth saving.

By exploiting the redundancy in video frames, extensive works have been proposed to perform spatial and temporal filtering [4]–[7], [12]–[15]. Temporal filtering solutions aim to remove a video frame that is duplicated or does not contain any interesting objects. For example, FilterForward [6] introduced microclassifiers on edge devices to detect relevant events and only frames with matching events would be transmitted to cloud servers. In contrast, spatial filtering solutions allocate more bits for pixels in the RoIs during encoding and transmitting. For example, DDS [5] constantly sent a low-quality stream and resent high-quality partial images based on feedback from the server. In the existing literature, most works [5]–[8] adopt context-agnostic approaches (i.e., treating incoming frames as independent images instead of a continuous flow of frames from one camera) during streaming, while only a few of them has explored the contextual information of video feeds, e.g., EAAR [4] used the candidate RoIs from the last frame to determine encoding quality of the next frame. However, 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.

Manuscript received April 1, 2022. *(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 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 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, UK (email: zhenhui.yuan@northumbria.ac.uk).

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

In this work, we explore the contextual information of video data from surveillance cameras. Since surveillance cameras generate video frames with static background, 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 foreground pixels that may contain interesting objects for inference. As a result, VaBUS has the potential to significantly reduce the bandwidth consumption between the edge and cloud, while achieving high inference accuracy in the task-oriented communications scenario. 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 in the semantic aspect. 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 approach of background understanding and subtraction is not a trivial task in the real scenario. There are three challenges to tackle before the practical deployment.

- First, on the edge, we need a lightweight approach to reconstruct the meaningless-for-inference background image from the dynamic video of cameras, such that we can subtract the redundant background before transferring the video to the cloud. Due to the limited computing resources at the edge, it is difficult to perform background image reconstruction or foreground/background segmentation at the speed of 24FPS. According to [16], background reconsutruction for a RGB image of 1080×1920 resolution on a Nvidia Jetson TX2 device requires a time of 79ms on average when using CPU, which is far from being real-time. Besides, the background image could change with variable illumination conditions or different camera viewports, which triggers irregular updates. In VaBUS, we deploy a *background reconstructor* module in the cloud to constantly learn the background of video feeds and send updates to the edge. Since background reconstruction is performed in the cloud, there's no additional computation overhead on the edge except the bandwidth consumed for transferring the background image from the cloud to the edge. To minimize the bandwidth consumption and latency overhead of transferring the background image, VaBUS distinguishes non-object background from the temporary background with objects, and adopts a *dynamic overlay* mechanism to dynamically generate overlaid background on the edge for RoI generation. Besides, deploying background reconstruction model in the cloud with abundant computing resources enables the usage of more advanced models, e.g., deep learning-based models, to achieve more accurate background reconstruction.
- Second, we need a robust approach on the edge to compute accurate RoIs, i.e., the foreground pixels that contain interesting objects. Simple pixel-wise difference between the incoming frames and the background image contains considerable detection noise. Misdetection of RoIs (i.e., too few pixels sent) would cause omission of interested objects while false alarms (i.e., too many pixels sent) might add redundant bits to RoIs, causing extra bandwidth waste. Based on pixel-wise difference results, VaBUS leverages an *adaptive weighting* module to assign greater weights to regions that are more likely to appear interesting objects. Specifically, it automatically learns from previous detection results and input frames to calculate the weights. To further reduce bandwidth consumption, VaBUS takes the object size into consideration and dynamically adjusts the encoding parameters with an *adaptive RoI encoding* module.
- 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 needed 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. Unlike previous works that use either optical flow [15] or motion vector-based [4] estimation methods, VaBUS adopts a lightweight experience-driven learning approach (i.e., *mapping estimator*) to learn from previous detection results and produce estimations. Specifically, it learns the *shifting* and *scaling* mapping matrix per pixel location based on box-to-box mapping according to the object tracking results in the cloud. When the edge fails to receive timely detection results from the cloud, the edge would be able to estimate the detection results of the current frame based on the learned mapping functions.

To validate the feasibility of VaBUS, we implement a prototype[1] 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 and motion vector-based methods.

The contributions of this paper can be summarized as follows:

- Proposing a new system that leverages rich contextual information of video feeds from surveillance cameras for real-time edge-cloud video analytics.
- Developing a *background reconstructor* that effectively learns the background in the cloud, and a *background overlay* mechanism to dynamically generate background on the edge.
- Designing an *adaptive weighting* module which assigns different weights across the frame to generate accurate

[1]The prototype will be made public after the paper is accepted.

RoIs, and an *adaptive RoI encoding* strategy that takes object size into consideration during encoding to further save bandwidth.
- Proposing *mapping estimator*, a new mapping-based offline estimation method that automatically learns 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

A series of techniques have been proposed to use real-time video analytics within the edge-cloud continuum. Previous works have explored DNN-specific optimization techniques including model distillation [17], splitting [18], sharing [19] and cascading [20], adaptively tuning a set of control knobs to constantly meet accuracy and latency requirements [21]–[23], optimized information pruning techniques [4]–[7], etc. 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. VaBUS shares the same concept of performing uneven-quality encoding, while in a more aggressive approach by leveraging the contextual information, i.e., completely removing background pixels that are unrelated to the task. As a consequence, VaBUS is able to save more bandwidth while achieving the same accuracy.

B. Background subtraction and foreground/background segmentation

Background subtraction has been extensively studied in the last decade [24]. The goal of background subtraction is to detect moving objects in the scenes when the camera is static, i.e., segmentation of foreground and background. Previous works 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 [25], non-parametric method such as kernel density estimation [26], traditional machine learning method [27] and hybrid methods with multi-modal data [28] and model fusion [29], etc. Deep learning-based methods can be classified into supervised and unsupervised methods. Supervised methods typically adopt 2D- or 3D- CNN and ConvLSTM models [30]–[33], 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 [34], [35]. Instead of directly performing the foreground and background segmentation, VaBUS adopts a two-phase strategy to utilize the computing resources on both the edge and cloud, i.e., learns a rough estimate of the true background image on the cloud and then tries to produce accurate foreground detection on the edge.

C. Computer Vision applications

A large number of deep learning models have been proposed for various vision applications, including image classification [36], instance segmentation [37], object detection [38], face recognition [39], image captioning [40], 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 [15]. For real-time video analytics, several techniques have been proposed to meet the latency requirement via local tracking on the edge devices. Glimpse [15] used the optical flow method to estimate the detection results of next frame based on previous frame. EAAR [4] adopted a more lightweight method by leveraging motion vector information from the codec. Elf [41] proposed a LSTM-based model to predict possible object locations in each frame. 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.

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: Our objective is to design a real-time edge-cloud video analytics system that saves as much bandwidth as possible while 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. The goal of video analytics systems is to perform deep learning inference. In VaBUS, we only send partial video frames to minimize data transmission, which tends to cause a drop in accuracy.
- **Latency**. Achieve a latency of one second. Latency is a key component for real-time systems which require timely feedback.
- **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.

2) VaBUS Overview: Figure 1 illustrates the VaBUS architecture and indicates its life cycle. The edge constantly generates RoIs by subtracting the background from incoming frames through a *difference detector* (referred to as *diff detector*). An extra *adaptive weighting* module is used to boost

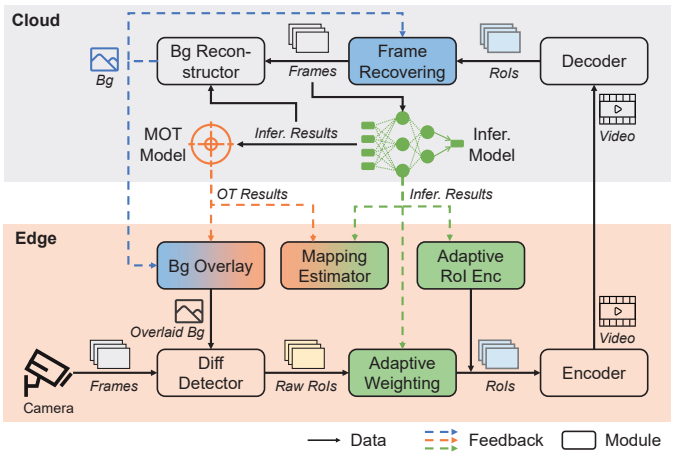


Fig. 1. System Overview of VaBUS.

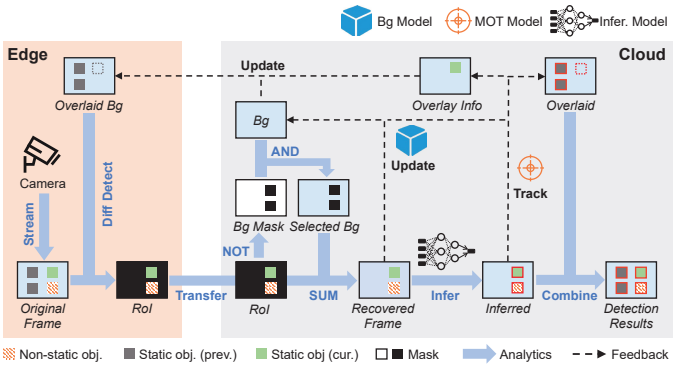


Fig. 2. Background reconstruction and dynamic overlay.

the accuracy of raw RoIs. Considering that static objects can also be part of the background, we use a *background overlay* (referred to as *bg overlay*) module to dynamically add these objects to the non-object background and remove them from the overlaid background when the objects are no longer static. Once ready, the RoIs will be encoded into a video using an *adaptive RoI encoding* (referred to as *adaptive RoI Enc*) module, which takes the object size into consideration when assigning uneven encoding quality across the frame. After receiving the video, the cloud will first decode the RoIs from it then recover original frames with a *frame recovering* module. Once the frames are ready, the inference model (referred to as *infer. model*) will be run to generate results. These inference results (referred to as *infer. results*) are the output of user-specified task and will also be used to update the *adaptive weighting*, *adaptive RoI Enc* and *mapping estimator* module. 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 running in the cloud to detect the motion of each individual object. The tracking results will be used to update the *bg overlay* and *mapping estimator* module on the edge. In general, the edge device constantly generates RoIs, while the cloud is responsible for inference and video context analysis to synchronize various modules. 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 [42] and implemented in OpenCV [43] 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 using both background image B and RoI I. The regions (M_{bg}) with detected objects (M_{det}) and also being set to black in RoI (M_{black}) are replaced with corresponding pixels in existing background for reconstruction (Line 14). The remaining regions in RoI are kept unchanged (M_{fg}). 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

Algorithm 1 Frame Recovering and Background Updating

1: **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
2: **Outputs:** Background estimation model $\mathcal{M}$, background image B
3: **Step 1**: *Recovering and Reconstruction Phase*
4: Initialize background image B, estimation model $\mathcal{M}$
5: $M_{\text{det}}^{i,j} \leftarrow \begin{cases} 1, & \text{if pixel in } D \\ 0, & \text{otherwise} \end{cases}$
6: $M_{\text{black}}^{i,j} \leftarrow \begin{cases} 1, & \text{if } \sum_{c=1}^{3} I_{i,j,c} < \lambda_{\text{roi}} \\ 0, & \text{otherwise} \end{cases}$
7: $M_{\text{bg}} \leftarrow M_{\text{det}}$ **OR** M_{black}
8: $n_{\text{det}} \leftarrow$ number of detected pixels in D
9: $n_{\text{roi}} \leftarrow$ number of non-zero pixels in I
10: **if** $n_{\text{det}}/n_{\text{roi}} < \lambda_{\text{det}}$ **then**
11: $M_{\text{fg}} \leftarrow$ **NOT** M_{bg}
12: $I_{\text{fg}} \leftarrow I$ **AND** M_{fg}; $I_{\text{bg}} \leftarrow B$ **AND** M_{bg}
13: $I_{\text{rec}} \leftarrow$ **SUM**$(I_{\text{fg}}, I_{\text{bg}})$
14: $\mathcal{M} \leftarrow$ update with I_{rec}
15: $B \leftarrow$ get new background image from $\mathcal{M}$
16: **end if**
17: **Step 2**: *Updating Phase*
18: $d \leftarrow \frac{1}{H \times W} \sum \|B - B_{\text{last}}\|$ (B_{last}: last sent background image to the edge)
19: **if** $d > \lambda_{\text{chg}}$ and B haven't been updated for λ_{gap} rounds **then**
20: update B on the edge
21: **end if**

(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-B1 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 their corresponding locations have changed or not. We use a multiple-objects tracking (MOT) algorithm [44] 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.

After receiving the information of static objects, the edge will add these objects to the background by modifying pixels at corresponding locations. At the same time, the detection results of these objects are cached for later use. 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 highly confident regions in a frame while setting remaining regions to black to reduce encoded video size, hence saving bandwidth consumption. Similar approaches have been proposed in [5], 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.

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

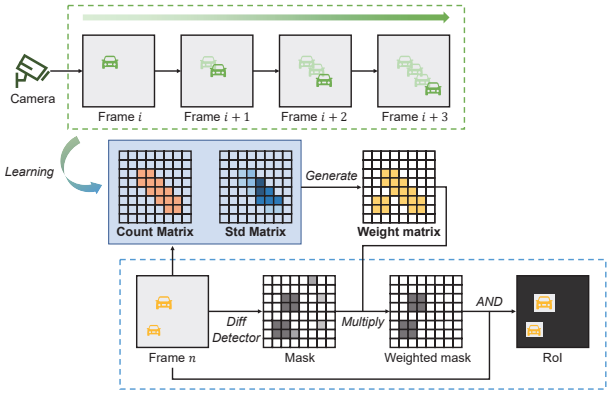


Fig. 3. Adaptive weighting learning process.

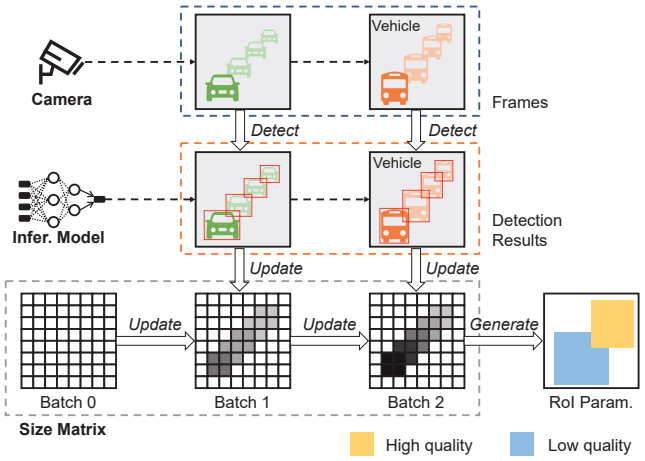


Fig. 4. Learning adaptive RoI encoding parameters.

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 λ_1, and remaining values are increased by λ_2. In our implementation, we choose $\lambda_1 > \lambda_2$ 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., $C[y_1 : y_2, x_1 : x_2]$ -= λ_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 3.

3) Adaptive RoI encoding: Encoding with unbalanced quality level across the frame has been shown as an effective approach of reducing bandwidth consumption in prior works [4], [5]. Through assigning higher encoding quality to regions that require more details (e.g., regions with small objects) and lower quality to remaining regions, the inference accuracy can be improved 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 4.

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

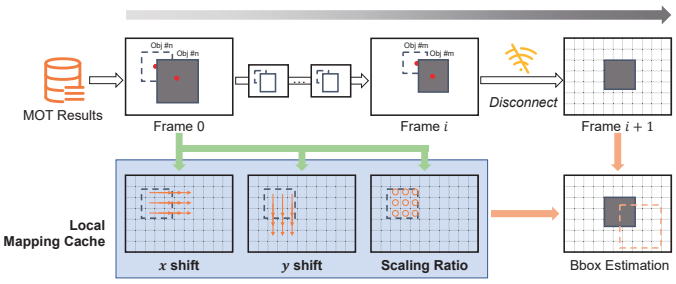


Fig. 5. Offline experience-driven learning.

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.

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 a given time frame. Unlike traditional approaches that estimate inference results based on motion vectors or optical flow [4], [15], 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-B2) 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 5.

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.

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**

Algorithm 3 Bounding Box Estimation

1: **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)
2: **Outputs:** estimated bounding box $D_1, D_2, \cdots, D_n$
3: **for** i^{th} round in $1, 2, \cdots, n$ **do**
4: **for** (x_1, y_1, x_2, y_2) of j^{th} object in B_{i-1}^j **do**
5: $s_x \leftarrow \overline{S_x}[y_1 : y_2, x_1 : x_2]$; $s_y \leftarrow \overline{S_y}[y_1 : y_2, x_1 : x_2]$
6: $e \leftarrow \overline{E_x}[y_1 : y_2, x_1 : x_2]$
7: $s_x \leftarrow \beta_1^i * s_x$; $s_y \leftarrow \beta_1^i * s_y$
8: $e \leftarrow \begin{cases} \max(1, \beta_1^i * e), & \text{if } e > 1 \\ \min(1, \beta_2^i * e), & \text{otherwise} \end{cases}$
9: $\hat{x}_1 \leftarrow x_1 + s_x - (e-1) * (x_2 - x_1)/2$
10: $\hat{x}_2 \leftarrow x_2 + s_x + (e-1) * (x_2 - x_1)/2$
11: $\hat{y}_1 \leftarrow y_1 + s_y - (e-1) * (y_2 - y_1)/2$
12: $\hat{y}_2 \leftarrow y_2 + s_y + (e-1) * (y_2 - y_1)/2$
13: $B_i^j \leftarrow (\hat{x}_1, \hat{x}_2, \hat{y}_1, \hat{y}_2)$
14: **end for**
15: **end for**

IV. Experimental Results

A. Implementation

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. The cloud server and edge device are connected with a TP-LINK TL-SG1008D router through a 1Gbps Ethernet cable. The operating system of cloud server and edge device are Ubuntu 20.04 and Ubuntu 18.04 respectively.

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

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 [45] API for real-time video encoding where the setROIParams function [46] is used to set different RoI regions and QP delta value as in [4]. Besides, GStreamer [47] API is used for real-time video frame capture. In the cloud, we decode the video to frames using the VideoCapture function [48] 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++.

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 [49] 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 [50] 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.

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 [51] by searching the keyword 'highway traffic videos'. Videos not captured by surveillance cameras, e.g., dashcam videos, are removed manually. (2) We select a subset of videos from the VIRAT Video Dataset [52], 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 [53], [54], 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 [55], [56], 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 10. 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 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. The results show that VaBUS is able to effectively reduce bandwidth consumption and maintain high accuracy and low latency at the same time.

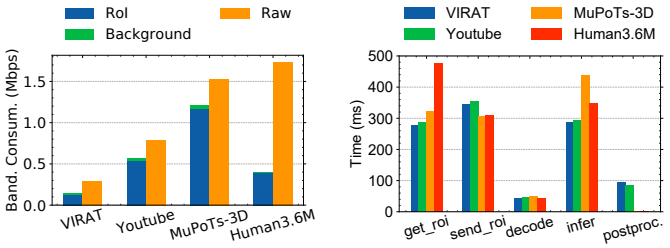
Figure 6 shows the detailed composition of bandwidth consumption and latency of VaBUS. In Figure 6a, *RoI* represents encoded video size sent from Jetson to the cloud server after background subtraction and adaptive RoI encoding. *Background* is the total size of transferred background images. *Raw* 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

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

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

Dataset	Approaches	F1-score	Compress.	Latency	CPU	GPU
VIRAT	Baseline	94.8%	0%	635 ms	23.2%	2.1%
	Baseline + BS	88.4%	52.4%	1048 ms	31.6%	0.0%
	Baseline + BS + AW	88.1%	58.1%	1062 ms	31.7%	0.0%
	Baseline + BS + AW + RoIE	88.9%	57.5%	1084 ms	31.3%	0.0%
YouTube	Baseline	95.1%	0%	654 ms	25.1%	3.1%
	Baseline + BS	92.7%	17.0%	1362 ms	44.4%	2.1%
	Baseline + BS + AW	93.6%	16.9%	1268 ms	42.5%	0.0%
	Baseline + BS + AW + RoIE	92.6%	26.8%	1160 ms	36.3%	0.0%



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

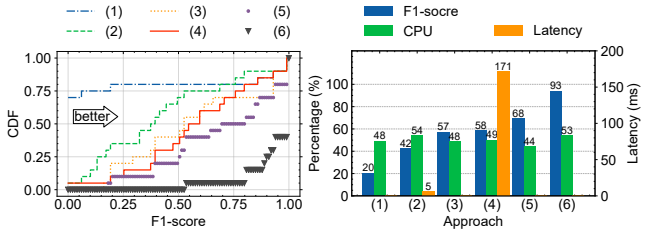
Fig. 6. Bandwidth consumption and latency composition of VaBUS.

boxes in object detection. In Figure 6b, 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-B2). 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

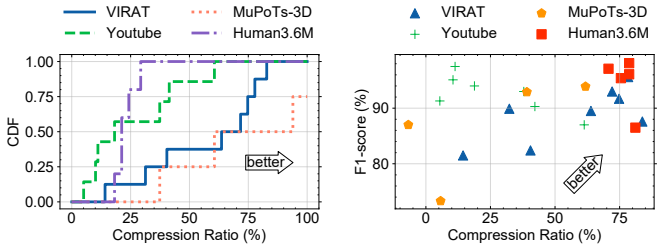
To understand each component's impact on VaBUS, we perform ablation study by examining each functioning part. We measure the accuracy (i.e., F1-score) of object detection task in four approaches: 1) the baseline solution (*Baseline*), 2) enabling background subtraction (*BS*) which includes background reconstruction and overlay, 3) enabling adaptive weighting (*AW*) and 4) enabling adaptive RoI encoding (*RoIE*). The baseline approach 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 *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. Since our methods don't rely on GPU, the utilization of GPU resource is always low.



(a) F1-score distribution (b) Accuracy, CPU and latency

Fig. 7. Performance of mapping estimator (ME) module. 1-Worst, 2-ME, 3-MvOT, 4-OF, 5-MvOT+ME, 6-Best.

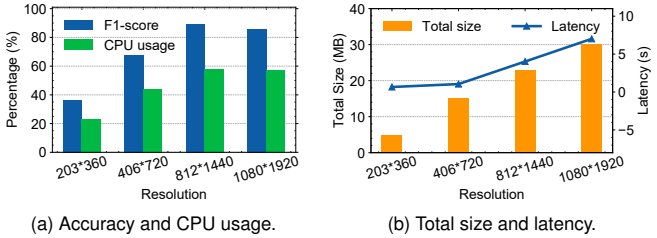


(a) (b)

Fig. 8. Distribution of per-video bandwidth consumption and F1-score for four datasets.

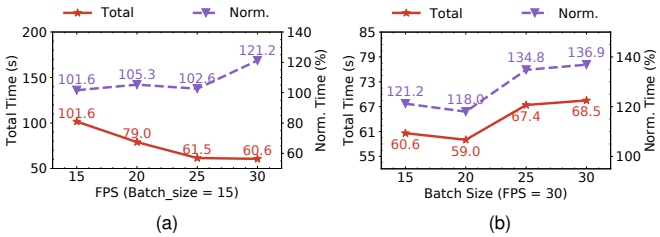
D. Offline Estimation Performance

The *Mapping Estimator* (ME) module of VaBUS is designed to estimate inference results of current batch of frames based on last batch. Figure 7 shows the performance of ME module. In Figure 7a, 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 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 [4]. In the optical flow-based (referred to as *OF*) approach, we use the dense optical flow algorithm proposed by Gunnar Farneback [57] 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 7a, 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 7b, 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.



(a) Accuracy and CPU usage. (b) Total size and latency.

Fig. 9. System performance under various input resolution.



(a) (b)

Fig. 10. Time consumption under various batch size and frame rate. (Norm. : Normalized)

E. Sensitivity to Application Settings

Impact of video genres. Figure 8 shows per-video bandwidth consumption and F1-score of fours datasets on the object detection and human keypoint detection task. In Figure 8a, it can bee seen that different dataset has different distribution of compression ratio. VaBUS saves the most bandwidth on Human3.6M while shows the least on YouTube. From Figure 8b, 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 on the area of background filtered by the *diff detector* module (§III-C1). 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.

Impact of resolution. Frame resolution is a key factor to consider when streaming videos. Figure 9 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 9a, 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 9b). 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 9b). When resolution is 203×360 or 406×720, the system runs in real time with a latency of only around

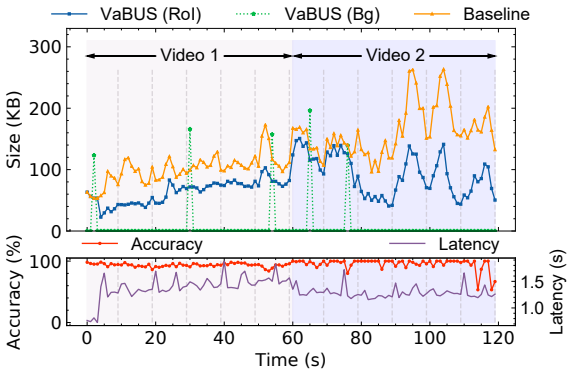


Fig. 11. Effects of background change to VaBUS.

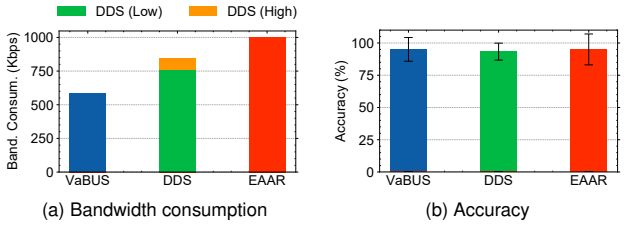


Fig. 12. Comparison with other approaches.

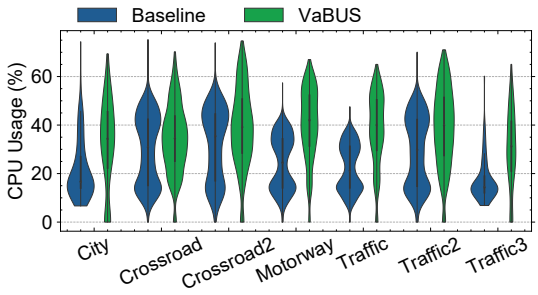


Fig. 13. CPU resources consumption of VaBUS.

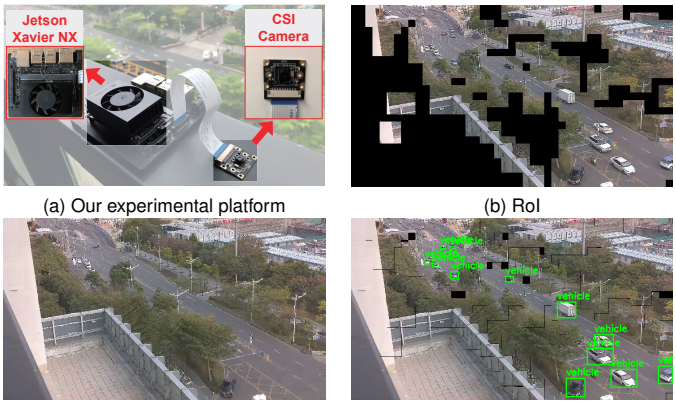
one second, while the latency surges when the resolution is higher (Fig 9b). Similarly, the CPU usage is saturated when resolution is higher than or equal to 812×1440. This means increasing the resolution does not necessarily improve the system performance. It needs to be adjusted in accordance with available computing resources on the edge device and the actual application scenarios.

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 10 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 10a, it can be observed that elevating frame rate from 15FPS to 30FPS increases the throughput of 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 10b), 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.

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 11. 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-C3). 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 bsaeline approach. Besides, changing the background does not impact the accuracy as shown in the lower part of Figure 11. 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.

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 bandwidth consumption and accuracy of VaBUS with two other information pruning approaches, i.e., DDS [5] and EAAR [4]. 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 23 for high quality feeds and QP 33 for low quality



(a) Our experimental platform (b) RoI

(c) Reconstructed Bg (d) Detected Frame

Fig. 14. Real-world deployment and qualitative results of VaBUS.

feeds. 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). From Figure 12, it is shown the accuracy of the three approaches is similar but VaBUS consumes 31.1% and 41.7% less bandwidth compared with DDS and EAAR respectively. The reason is that the bandwidth needed for transferring the full background content in DDS and EAAR are saved in VaBUS.

G. Resource Consumption

To investigate the system resource usage of VaBUS, we show the distribution of CPU usage of 7 videos from the YouTube dataset in Figure 13. It is shown that the CPU usage mostly stays in the range of 10%-60% for both the baseline approach and VaBUS. Unlike other video analytics systems e.g., DDS [5] and EAAR [4] that use minimal system resources, we trade a small amount of computing resources for substantial bandwidth saving. From Figure 13, it can be observed that the overhead of VaBUS is 10%-20% compared to the baseline approach, while the exact value changes along with different videos. The system resource usage of baseline approach as well as the modules of VaBUS can be further optimized to run more efficiently and we leave the optimization for future work.

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* [58]. Figure 14a shows the actual deployment of our experimental hardware platform. Figure 14b 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 14c, 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 14d. 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 9). 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

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 a considerable 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.

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.

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 [22] 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., 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.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China under grant No. 61972189, Guangdong Province Key Area R&D Program under grant No. 2018B010113001, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

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*, Jul. 2019, pp. 475–515.

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

[4] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, Los Cabos, Mexico, Aug. 2019.

[5] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, Virtual Event, USA, Jul. 2020.

[6] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. Dulloor, "Scaling video analytics on constrained edge nodes," in *Proceedings of Machine Learning and Systems*, Palo Alto, USA, Mar. 2019.

[7] 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 *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, Virtual Event, USA, Jul. 2020.

[8] J. Yi, S. Choi, and Y. Lee, "Eagleeye: Wearable camera-based person identification in crowded urban spaces," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, London, United Kingdom, Apr. 2020.

[9] "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/

[10] 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 *2017 IEEE international symposium on circuits and systems (ISCAS)*, Baltimore, USA, May. 2017.

[11] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[12] 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 Transactions on Computers*, vol. 69, no. 6, pp. 914–928, 2020.

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

[14] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. Gonzalez, "Rexcam: Resource-efficient, cross-camera video analytics at enterprise scale," *ArXiv*, vol. abs/1811.01268, 2018.

[15] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, Seoul, South Korea, Nov. 2015.

[16] "Vpi - vision programming interface: Background subtractor," Accessed Mar. 31, 2022. [Online]. Available: https://docs.nvidia.com/vpi/algo_background_subtractor.html

[17] M. Khani, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh, "Real-time video inference on edge devices via adaptive model streaming," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, Virtual Event, USA, Oct. 2021.

[18] R. Mehta and R. Shorey, "Deepsplit: Dynamic splitting of collaborative edge-cloud convolutional neural networks," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, Bengaluru, India, Jan. 2020.

[19] A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger, "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, USA, Jul. 2018.

[20] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: Optimizing neural network queries over video at scale," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1586—-1597, 2017.

[21] 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 *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, USA, Mar. 2017.

[22] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, Budapest, Hungary, Aug. 2018.

[23] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, Budapest, Hungary, Aug. 2018.

[24] B. Garcia-Garcia, T. Bouwmans, and A. J. R. Silva, "Background subtraction in real applications: Challenges, current models and future directions," *Computer Science Review*, vol. 35, p. 100204, 2020.

[25] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Cambridge, UK, Aug. 2004.

[26] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *European conference on computer vision*, Berlin, Heidelberg, Apr. 2000.

[27] B. Han and L. S. Davis, "Density-based multifeature background subtraction with support vector machine," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 5, pp. 1017–1023, 2011.

[28] R. Trabelsi, I. Jabri, F. Smach, and A. Bouallegue, "Efficient and fast multi-modal foreground-background segmentation using rgbd data," *Pattern Recognition Letters*, vol. 97, pp. 13–20, 2017.

[29] T. Akilan, Q. J. Wu, and Y. Yang, "Fusion-based foreground enhancement for background subtraction using multivariate multi-model gaussian distribution," *Information Sciences*, vol. 430, pp. 414–431, 2018.

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

[31] M. Braham and M. Van Droogenbroeck, "Deep background subtraction with scene-specific convolutional neural networks," in *2016 international conference on systems, signals and image processing (IWSSIP)*, Bratislava, Slovakia, May. 2016.

[32] D. Sakkos, H. Liu, J. Han, and L. Shao, "End-to-end video background subtraction with 3d convolutional neural networks," *Multimedia Tools and Applications*, vol. 77, no. 17, pp. 23 023–23 041, 2018.

[33] 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. 97 949–97 958, 2019.

[34] 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 *2018 25th IEEE International Conference on Image Processing (ICIP)*, Athens, Greece, Oct. 2018.

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

[36] A. A. M. Al-Saffar, H. Tao, and M. A. Talab, "Review of deep convolution neural network in image classification," in *2017 International conference on radar, antenna, microwave, electronics, and telecommunications (ICRAMET)*, Jakarta, Indonesia, Oct. 2017.

[37] W. Gu, S. Bai, and L. Kong, "A review on 2d instance segmentation based on deep neural networks," *Image and Vision Computing*, vol. 120, p. 104401, 2022.

[38] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[39] M. M. Kasar, D. Bhattacharyya, and T. Kim, "Face recognition using neural network: A review," *International Journal of Security and Its Applications*, vol. 10, no. 3, pp. 81–100, 2016.

[40] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "A comprehensive survey of deep learning for image captioning," *ACM Computing Surveys (CsUR)*, vol. 51, no. 6, pp. 1–36, 2019.
[41] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, and Y. Zhang, "Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, New Orleans, United States, Oct. 2021.
[42] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.
[43] "Opencv: cv::backgroundsubtractorknn class reference," Accessed Mar. 31, 2022. [Online]. Available: https://docs.opencv.org/4.x/db/d88/classcv_1_1BackgroundSubtractorKNN.html
[44] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*, Phoenix, USA,, Sep. 2016.
[45] "Jetson linux api reference: Multimedia apis — nvidia docs," Accessed Mar. 31, 2022. [Online]. Available: https://docs.nvidia.com/jetson/l4t-multimedia/mmapi_group.html
[46] "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
[47] "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
[48] "Opencv: cv::videocapture class reference," Accessed Mar. 31, 2022. [Online]. Available: https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html
[49] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *ArXiv*, vol. abs/1804.02767, 2018.
[50] S. Kreiss, L. Bertoni, and A. Alahi, "Openpifpaf: Composite fields for semantic keypoint detection and spatio-temporal association," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–14, 2021.
[51] "highway traffic videos - youtube," Accessed Mar. 31, 2022. [Online]. Available: https://www.youtube.com/results?search_query=highway+traffic+videos
[52] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis *et al.*, "A large-scale benchmark dataset for event recognition in surveillance video," in *CVPR 2011*, Colorado, USA, June 2011.
[53] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, S. Sridhar, G. Pons-Moll, and C. Theobalt, "Single-shot multi-person 3d pose estimation from monocular rgb," in *2018 International Conference on 3D Vision (3DV)*, Verona, Italy, Sep. 2018.
[54] D. Mehta, H. Rhodin, D. Casas, P. Fua, O. Sotnychenko, W. Xu, and C. Theobalt, "Monocular 3d human pose estimation in the wild using improved cnn supervision," in *2017 international conference on 3D vision (3DV)*, Qingdao, China, Oct 2017.
[55] 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 transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1325–1339, 2013.
[56] C. Ionescu, F. Li, and C. Sminchisescu, "Latent structured models for human pose estimation," in *2011 International Conference on Computer Vision*, Barcelona, Spain, Nov. 2011.
[57] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Scandinavian conference on Image analysis*, Halmstad, Sweden, Jun. 2003.
[58] "iperf - the tcp, udp and sctp network bandwidth measurement tool," Accessed Mar. 31, 2022. [Online]. Available: https://iperf.fr/