

Online Edge-Assisted Video Analytics on Mobile Agents via Differential Video Encoding

Hongzi Zhu , Senior Member, IEEE, Jiangang Shen , Member, IEEE, Liang Zhang , Member, IEEE, Yunzhe Li , Shan Chang , Jie Wu , Fellow, IEEE, and Minyi Guo , Fellow, IEEE

Abstract—Ensuring stable and high-quality real-time video analytics for computationally constrained mobile agents is essential. However, limited computing resources and network bandwidth present significant challenges in meeting the objective of low response time and high inference accuracy. In this paper, we present DiVE, an edge-assisted video analytics system that utilizes motion vectors calculated by video codec to extract foregrounds and differentially encode frames. DiVE removes rotational components from motion vectors by solving over-determined linear equations and filters noisy motion vectors based on the observation that motion vectors of static objects point to the same point when the ego agent purely translates. To distinguish foregrounds from backgrounds, DiVE estimates the ground based on observations that all foregrounds stand on the ground and motion vectors on static objects at the same height have the same normalized magnitude. DiVE then uses region-growing-based clustering to identify foreground objects. An adaptive bitrate allocation method is applied to optimize accuracy under estimated bandwidth. We implement a prototype and conduct extensive experiments to evaluate the performance of DiVE. The results demonstrate that DiVE can improve detection accuracy by up to 19.0% and reduce response time by up to 56.0% compared with other video analytics schemes in real-world traces.

Index Terms—Deep learning, mobile agents, motion vector, video analytics, video streaming.

I. INTRODUCTION

RECENT years have witnessed a rapid increase of various types of mobile agents, such as autonomous driving agent on a normal vehicle, embodied AI agent on a robot, and smart agent on a drone, which may exist as an independent module and feature their hosts with many appealing intelligent mobile applications. Among many of the others, live video streams generated from mobile agents need fast treatment. For example, autonomous driving requires the system to have a comprehensive understanding of the scene and all related objects as quickly as possible in the real world. However, most mobile agents

can only conduct limited computation [1]. Therefore, computer vision tasks such as object detection, classification and semantic segmentation need to be accomplished nearly in real time for fast video analytics by offloading most computational workload to an edge server.

However, an effective edge-assisted online video analytics system needs to meet three rigid requirements. First, the system should achieve high analysis accuracy as inaccurate results may lead to severe consequences, such as car crashes. Second, the system should have a low response time even with a highly unstable wireless communication connection, especially for fast moving agents. Last but not least, the system should be lightweight and scalable given the limited computational power of a mobile agent and the potential huge number of agents in the system.

In the literature, existing edge-assisted video analytics schemes can be classed into three categories, i.e., frame filtering at camera side, video compressing based on server feedback, and local compressed model assisted video analytics. The frame filtering methods [2], [3], [4] adopt simple logic to recognize key frames and send them to the edge for model inference. These methods perform poorly in highly dynamic scenarios due to the detection results of other regular frames rely heavily on tracking the bounding box of key frame. Existing compression-based methods [5], [6], [7] allocate more bits to regions of a video frame that the edge deems to significantly impact inference accuracy, while reducing the transmission bits for the remaining regions. However, these methods depend on guidance from the edge, leading to huge latency. The last category of video analytics system rely on light-weight models [8], [9], [10] deployed on mobile devices to handle simple sub-tasks like large object detection or appearance of target objects. These methods inherently require significant computing power and demonstrate diminished accuracy when processing complex video content. Other studies [11], [12] adjust video encoding configurations like resolution or frame rate to chase maximum inference accuracy based on current available bandwidth. Nevertheless, such methods require offline profiling and cannot handle live video streams.

In this paper, we propose DiVE, a novel differential video encoding scheme for online edge-assisted video analytics on mobile agents. The core idea of DiVE is to leverage the low-cost motion vectors generated by video codec to extract foreground regions of interest so that the best image quality is reserved for the foreground while the bitrate of the final encoded video fits the current uplink bandwidth. As a result, DiVE only conducts limited computation for basic analytics but can achieve

Received 23 June 2025; revised 16 January 2026; accepted 7 March 2026. Date of publication 12 March 2026; date of current version 6 July 2026. This work was supported by the Natural Science Foundation of China under Grant 62432008, Grant U25B2021, and Grant 62472083. Recommended for acceptance by X. Han. (Corresponding author: Hongzi Zhu.)

Hongzi Zhu, Jiangang Shen, Yunzhe Li, and Minyi Guo are with the School of Computer Science, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: hongzi@sjtu.edu.cn).

Liang Zhang and Shan Chang are with the School of Computer Science and Technology, Donghua University, Shanghai 201620, China.

Jie Wu is with Cloud Computing Research Institute, China Telecom, Beijing 100083, China.

Digital Object Identifier 10.1109/TMC.2026.3672968

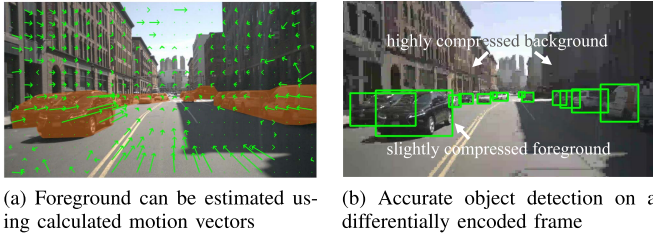


Fig. 1. An illustration of object detection on a DiVE-encoded video, where (a) the region of foreground objects in each frame of the video can be well estimated, using motion vectors obtained from a traditional video codec; and (b) each frame is differentially encoded with the foreground remaining clear and sharp yet the background heavily blurred, so that the bitrate of the live video stream adaptively fits the uplink bandwidth without affecting the final object detection accuracy.

low response time and high video analytics accuracy at the same time. Fig. 1 illustrates an example of object detection on a DiVE-encoded video where each frame is differentially encoded with the foreground remaining clear and sharp yet the background heavily blurred without affecting the final object detection accuracy.

There are two main challenges in designing DiVE. First, although motion vectors have distinct features for identifying different objects, they are very coarse and vulnerable to image noise and rotation of the camera, which makes it hard to use. To tackle this challenge, we conduct extensive empirical studies and have the key observation that when the agent only translates in space, all motion vectors on static objects point to the same point, referred to as focus of expansion (FOE), which be well utilized to filter out noise motion vectors. Furthermore, when the agent moves forward, its pitch and yaw angles can be well estimated by solving over-determined linear equations with carefully selected motion vectors. As a result, the rotational component can be effectively removed from a motion vector caused by compound motion.

Second, it is difficult to distinguish the foreground regions from the background with only motion vectors as foreground objects may have distinct motion and different image textures. We have the following two key observations: 1) all foreground objects stand on the ground and 2) non-rotational motion vectors on objects of the same height (e.g., the ground) in the real world have the same normalized magnitude. To deal with this challenge, we first estimate the ground region based on the above observations. Then, we propose a region-growing-based clustering algorithm to identify foreground objects, starting from those object seeds standing within the ground region. Finally, convex hulls of similar objects can be generated, forming multiple foreground regions.

We implemented a prototype of DiVE agent as a dashcam on four typical embedded devices, respectively, i.e., Raspberry Pi 5, Jetson Xavier Nano, Jetson Orin Nano and Jetson Orin NX. The DiVE agent is lightweight and written in C++. We conduct extensive experiments on two public driving video datasets, i.e., RobotCar and nuScenes, and three real-world datasets of distinct scenarios collected in Shanghai city. Results demonstrate that DiVE can well adapt the fast change of uplink condition even when hard handovers of base stations occur. DiVE can achieve

a large accuracy gain of up to 19.0% and reduce 56.0% response time, compared with the state-of-the-art edge-assisted video analytics methods. We highlight the main contributions made in this paper as follows: 1) a lightweight motion vector based foreground extraction scheme is proposed 2) an adaptive video encoding scheme is proposed 3) a prototype is implemented and extensive experiments are conducted, demonstrating the efficacy of DiVE.

II. PROBLEM DEFINITION AND PRELIMINARIES

A. System Model and Problem Definition

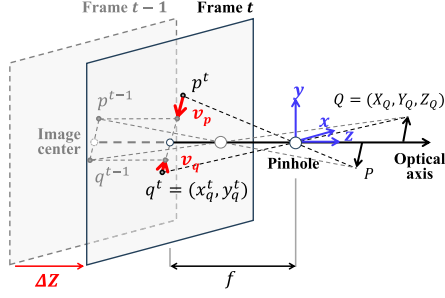
We consider two types of entities in an edge-assisted video analytics system:

- *Mobile Agents*: A mobile agent, e.g., a smart dashcam mounted on the windshield of a vehicle, is capable of generating live video streams while demanding video analytics such as object detection in real time. The mobile agent has constrained computational power but can run basic video encoding operation such as H.264 or MPEG-4. In addition, the agent can exchange information with an edge server via 4 G/5 G mobile communication networks. We consider that mobile agents may move in a high speed, which causes the uplink bandwidth to dramatically change when uploading encoded video streams to an edge server for analytics.
- *Serverless Edge Computing*: Upon receiving video analytics requests, a unified serverless computing fabric (SCF) for the edge-cloud continuum has sufficient computational power for video decoding and online deep neural network (DNN) model inference, and returns the results to mobile agents with low end-to-end model inference and communication latency.

Given the dynamic uplink bandwidth, the online edge-assisted video analytics problem is defined as: how to successfully encode and transmit a live video stream at a mobile agent so that the edge server can successfully decode the received video and achieve the required analytics accuracy using existing pre-trained DNN models, while meeting the end-to-end real-time constraint at the same time.

B. Video Encoding

To compress a video, frames are organized into structures known as Groups of Pictures (GoPs). Each GoP begins with an *I*-frame, which is encoded independently without any reference to other frames. Following the *I*-frame are multiple *P*-frames and *B*-frames where only the differences from reference frames are encoded. Given a new frame and a reference frame, the encoder processes the new frame in unit of coding blocks (we utilize coding block to refer to the basic coding unit for block-based video coding standards, e.g., macroblock in H.264 and Coding Tree Unit in H.265) in the following three steps. First, for each coding block in the current frame, it searches for the most similar coding block in the reference frame so that the amount of residual data needed for encoding is reduced. The displacement from the current coding block to the corresponding coding block in the reference frame is represented by a two-dimensional *motion*

Fig. 2. Motion vectors of static objects caused by camera translation in z -axis.

vector. Second, given the required bitrate and the differential encoding requirement, represented as a quantizer parameter (QP) offset map where a coding block should be compressed harder would have a positive value, it determines the specific QP value for each coding block. Finally, it performs entropy encoding on the transformed and quantized data [13].

C. Motion Vectors Caused by Camera Translation

We can characterize a general camera with a simple pinhole camera model. As illustrated in Fig. 2, a static point Q in the physical world with coordinates (X_Q, Y_Q, Z_Q) in the world coordinate system is projected through the pinhole to a point q on the image plane of the camera with image coordinates (x_q^t, y_q^t) at frame t . If the camera coordinate system coincides with the world coordinate system, (X_Q, Y_Q, Z_Q) and (x_q^t, y_q^t) satisfies the equations as follows

$$x_q^t = f \frac{X_Q}{Z_Q}, \quad y_q^t = f \frac{Y_Q}{Z_Q}, \quad (1)$$

where f is the focal length represented in the units of pixels.

When the camera translates with a displacement of ΔZ along z -axis from frame $t-1$ to frame t , the projection point moves from q^{t-1} to q^t with a motion vector of v_q , which can be decomposed to vx_q and vy_q along the x -axis and y -axis, respectively, calculated as

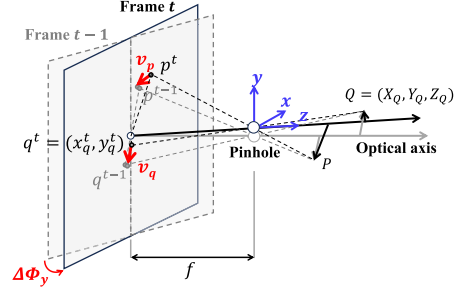
$$vx_q = \frac{\Delta Z x_q^t}{Z_Q}, \quad vy_q = \frac{\Delta Z y_q^t}{Z_Q}. \quad (2)$$

In general, if the camera translates in an arbitrary direction in the physical world with the respective displacement of ΔX , ΔY and ΔZ along each axis, vx_q and vy_q can be calculated as

$$vx_q = \frac{\Delta Z}{Z_Q} \left(x_q^t - \frac{\Delta X f}{\Delta Z} \right), \quad vy_q = \frac{\Delta Z}{Z_Q} \left(y_q^t - \frac{\Delta Y f}{\Delta Z} \right). \quad (3)$$

We have the following observation:

Observation 1: When the camera translates forward, motion vectors of static objects point to the same point, referred as focus of expansion (FOE), which also coincides with the vanishing point [14]. Moreover, the magnitude of these vectors are proportional to both the depth of corresponding objects and the distance between the projected points and FOE on the image plane.

Fig. 3. Motion vectors of static objects caused by camera rotation in y -axis.

From (3), the position of FOE on the image plane is $(\frac{\Delta X f}{\Delta Z}, \frac{\Delta Y f}{\Delta Z})$.¹ Fig. 4(a) depicts an example of motion vectors calculated when a camera mounted on a vehicle moves forward. It can also be seen that Observation 1 does not hold for motion vectors calculated on moving objects. On one hand, these vectors are distinctive and can be utilized for identifying these moving objects. On the other hand, they are more complicated caused by the relative movement between these moving objects and the camera. In addition, motion vectors in regions with plain textures are hard to calculate and seem noisy.

D. Motion Vectors Caused by Camera Rotation

When the camera rotates $\Delta \phi_y$ around its y -axis as illustrated in Fig. 3, vx_q and vy_q are calculated as

$$vx_q = -\Delta \phi_y f - \frac{\Delta \phi_y x_q^t}{f}, \quad vy_q = -\frac{\Delta \phi_y x_q^t y_q^t}{f}. \quad (4)$$

Generally, if the camera rotates around each axis with the respective degrees of $\Delta \phi_x$, $\Delta \phi_y$ and $\Delta \phi_z$, vx_q and vy_q can be calculated as

$$vx_q = -\Delta \phi_y f + \Delta \phi_z y_q^t + \frac{\Delta \phi_x x_q^t y_q^t}{f} - \frac{\Delta \phi_y x_q^t}{f},$$

$$vy_q = \Delta \phi_x f - \Delta \phi_z x_q^t - \frac{\Delta \phi_y x_q^t y_q^t}{f} + \frac{\Delta \phi_x y_q^t}{f}. \quad (5)$$

Fig. 4(b) depicts an example of motion vectors calculated when the vehicle makes a left turn. It can be seen that rotation also violates the motion vector distribution described in Observation 1, which makes it hard to identify objects of interest.

E. Datasets

We consider three public mobile video datasets for study as follows:

1) **KITTI:** The **KITTI** [15] dataset is a widely used benchmark in autonomous driving research. The **KITTI** dataset is collected by driving in rural areas and on highways around

¹Note that the FOE can represent the intersection of the camera's optical axis and the mobile agent's forward translation direction and is fixed once the camera is rigidly mounted on the mobile agent. FOE can be obtained through a one-time offline calibration process. During calibration, the agent moves straight forward and the FOE is estimated by applying RANSAC-based intersection fitting on the motion vectors in each frame. FOEs obtained from multiple frames are then refined with RANSAC to achieve the final robust estimation of FOE.

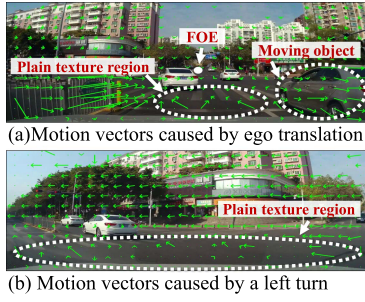


Fig. 4. Examples of calculated motion vectors.

Karlsruhe. The video data is captured with a frame rate of 10 frames per second (FPS) and image resolution is 1242×375 pixels. In the KITTI dataset, inertial measurement unit (IMU) data is sampled at 100 Hz, including three-axis linear acceleration and three-axis angular velocity. The IMU data is recorded with timestamps to enable precise synchronization with camera frames, ensuring temporal alignment between every frame and its corresponding IMU measurements.

2) *nuScenes*: The *nuScenes* [16] dataset is a public large-scale dataset for autonomous driving. The *nuScenes* dataset is collected in Singapore and Boston with a frame rate of 12 FPS, and image resolution is 1600×900 pixels. In the *nuScenes* dataset, every sixth video frame is sampled as a keyframe and annotated. The annotations include 3D bounding boxes i.e., position, dimensions and orientation, and object categories e.g., vehicles, pedestrians. The *nuScenes* test set is consisted of 150 video clips with each length of 20s under various traffic scenarios.

3) *RobotCar*: The *RobotCar* [17] dataset is an autonomous driving dataset collected in Oxford, U.K. over one year on a 10 km route. The *RobotCar* dataset consists of approximately 1000 km of recorded driving and video data is captured with a frame rate of 16 FPS under various weather conditions. In the *RobotCar* dataset, each video clip is labeled with weather conditions. The total number of frames is over 20 million and image resolution is 1280×960 pixels.

4) *DiVESet*: We collect a real-world driving dataset along with LTE CAT.1 bandwidth traces in Shanghai called *DiVESet* for study. The video data is captured with a frame rate of 30 FPS under three typical traffic scenarios, i.e., campus, urban and highway. In each scenario, the real-world bandwidth traces are collected with each trace containing ten driving video clips of five minutes and the corresponding LTE CAT.1 bandwidth measures and the ID of base stations recorded at an interval of one second. The image resolution is 1280×720 pixels. The GPS coordinates are also recorded at an interval of one second.

III. DESIGN OF DiVE

A. Overview

The core idea of DiVE is for a resource-constrained mobile agent to leverage motion vectors output by the video codec to identify foreground objects in each frame, and adaptively choose different compression ratios between the foreground and

the rest of a frame, according to the current uplink bandwidth. As a result, the encoded video can best fit the dynamic uplink condition while maintaining supreme image quality for these regions of interest, achieving high model inference accuracy at the edge server and low end-to-end response time. Fig. 5 depicts the architecture of DiVE, where a mobile agent uploads differentially encoded video to an edge server for analytics and the server returns detection results back to the agent for various downstream tasks. To this end, one mobile agent integrates four effective components:

Preprocessing: For each frame, given the raw motion vectors generated by the video codec, the motion status of the agent is estimated according to the distribution of motion vectors. The motion vectors are corrected by eliminating corresponding rotation components.

Foreground Extraction (FE): Given the fact that foreground objects stand on the ground and similar motion vectors can be observed on the same object, FE first utilizes a unique feature of motion vectors to neatly estimate the ground region in each frame. Then, objects are identified by clustering similar motion vectors starting from the ground region. Based on the detected motion status, if the agent comes to a stop when no motion vectors can be used for ground estimation, FE reuses the latest estimated foreground and clusters similar motion vectors as moving objects which are also treated as foregrounds.

Adaptive Video Encoding (AVE): Given the identified foreground regions in each frame, AVE first measures the uplink bandwidth and then assigns different compression ratios, referred to as QP values, for the foreground regions and for the rest of the frame, respectively, resulting in an encoded video with detailed foregrounds that best fits the current uplink condition.

Motion-vector-based Offline Tracking (MOT): When an encoded frame cannot be transmitted to the edge server in time due to a severely impaired uplink, MOT conducts local object tracking based on motion vectors to derive the detection result of the frame.

B. Preprocessing

As analyzed in Section II, utilizing raw motion vectors has the potential to distinguish objects of interest in the foreground but they are susceptible to disturbance such as camera rotation and plain image texture.

1) *Motion Vector Calculation*: When a new frame $\mathcal{F}^t$ is captured, existing block-matching motion estimation algorithms [18] can be used to calculate the set of motion vectors $\mathcal{M}^t$ for $\mathcal{F}^t$ using previous frame $\mathcal{F}^{t-1}$ as a reference frame.

2) *Ego Motion Judgement*: It is necessary to judge whether the mobile agent is in motion as Observation 1 stands only when the agent translates in space. To this end, we analyze the non-zero motion vector ratio of a frame, denoted as η , defined as the ratio of the number of coding blocks with non-zero motion vector to the number of total coding blocks. We randomly select 50 video clips from the *nuScenes* test set and manually divide video frames into two categories, i.e., ego agent is moving and ego agent is static. We calculate η of each frame and perform a categorized analysis based on the motion state of ego agent.

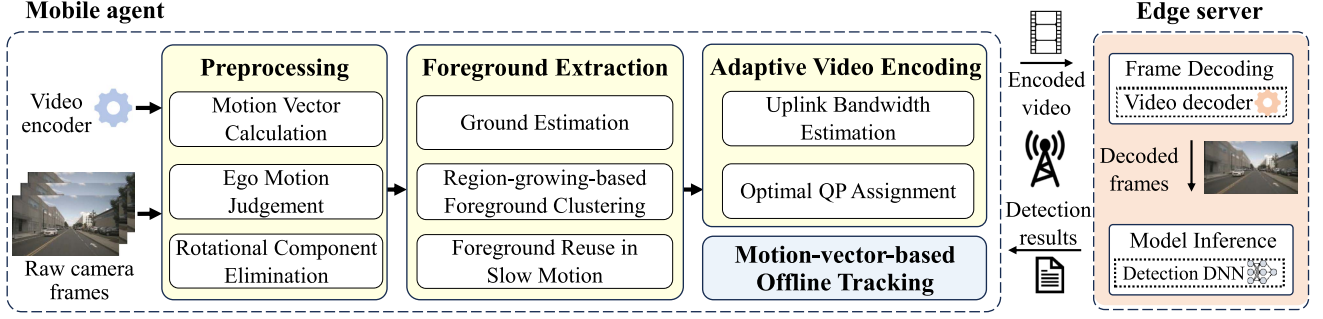


Fig. 5. Architecture of DiVe.

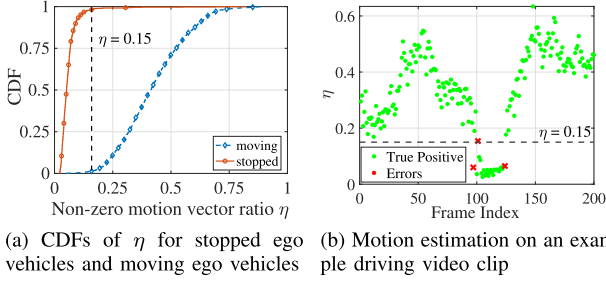


Fig. 6. The non-zero motion vector ratio can be utilized for ego motion detection.

Fig. 6(a) plots the cumulative distribution functions (CDFs) of η . It can be seen that, with high probability (over 98%), whether the agent is in motion can be judged by simply comparing whether η is larger than a threshold (e.g., $\eta > 0.15$). Fig. 6(b) depicts the η as a function of time in an example driving video clip where the ego agent comes to a stop for a while and then starts to accelerate. It can be seen that motion judged by η is highly consistent with the ground truth.

3) *Rotational Component Elimination*: If the mobile agent, such as a vehicle, can only translate along its z -axis (i.e., $\Delta X = 0$ and $\Delta Y = 0$) and rotate around its x - and y -axis (i.e., $\Delta\phi_z = 0$), motion vectors combining (3) and (5) can be simplified as

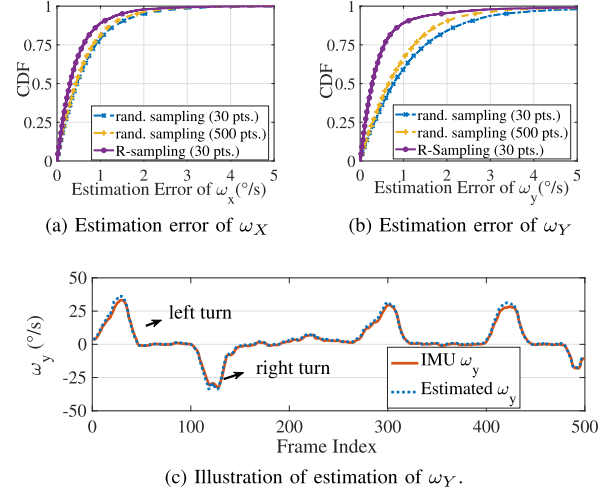
$$\begin{aligned} vx_q &= \frac{\Delta Z x_q^t}{Z_q} - \Delta\phi_y f + \frac{\Delta\phi_x x_q^t y_q^t}{f} - \frac{\Delta\phi_y x_q^t^2}{f}, \\ vy_q &= \frac{\Delta Z y_q^t}{Z_q} + \Delta\phi_x f - \frac{\Delta\phi_y x_q^t y_q^t}{f} + \frac{\Delta\phi_x y_q^t^2}{f}. \end{aligned} \quad (6)$$

In (6), $\frac{\Delta Z}{Z}$ can be eliminated and we have

$$x_q^t f \Delta\phi_x + y_q^t f \Delta\phi_y = y_q^t vx_q - x_q^t vy_q, \quad (7)$$

where all variables are known except $\Delta\phi_x$ and $\Delta\phi_y$ and the equation can be treated as a linear equation about $\Delta\phi_x$ and $\Delta\phi_y$. Therefore, $\Delta\phi_x$ and $\Delta\phi_y$ can be obtained by solving over-determined linear equations with a set of more than two motion vectors.

As noisy motion vectors present, how to select such a set of motion vectors that can lead to accurate $\Delta\phi_x$ and $\Delta\phi_y$ is non-trivial. We propose a simple yet effective method, called *R*-sampling, where k non-zero motion vectors that have the shortest distance to the fixed FOE are selected. The rationale

Fig. 7. Efficiency illustration of *R*-sampling.

of *R*-sampling lies in the fact that vectors near FOE normally correspond to points at very large depth along the translation direction. Since the translational component of motion vectors is inversely proportional to depth, its magnitude diminishes as depth increases, whereas the rotational component remains depth-invariant. Consequently, motion vectors sampled near the FOE are dominated by the rotational component and are more suitable for rotation estimation. The pseudo code of *R*-sampling is shown in Algorithm 1. We then adopt RANSAC [19] to solve the over-determined linear equations.

We conduct experiments on KITTI to evaluate the performance of *R*-sampling. We randomly select 6 video clips, 6332 frames in the KITTI dataset to conduct our experiments and utilize IMU data to calculate the ground truth of rotation speed of ego agent at each frame. We set $k = 30$ in *R*-sampling and $k = 30, 500$ in random sampling, respectively. Fig. 7(a) and (b) plot the CDFs of estimated errors of rotational speed around x - and y -axis of the camera, denoted as ω_x and ω_y , respectively. It can be seen from the results that *R*-sampling with 30 samples can achieve higher rotational speed estimation accuracy, compared with random sampling with 500 points. Fig. 7(c) also illustrates an instance of the estimated rotational speed ω_y .

Given the estimated $\Delta\phi_x$ and $\Delta\phi_y$, rotational components in (6) can be effectively removed and we can achieve the translation components of motion vectors.

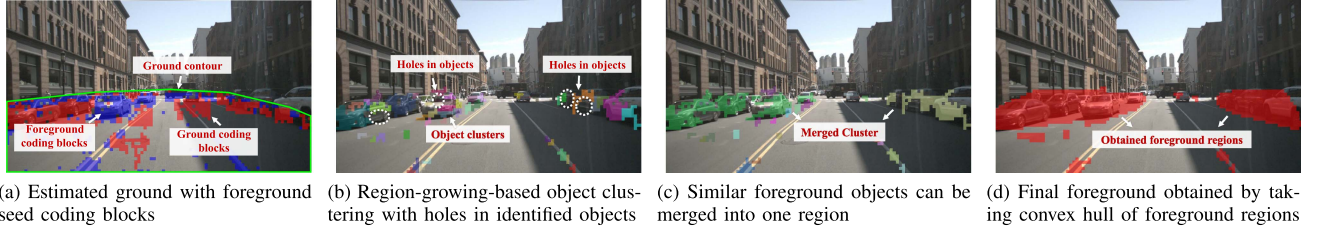


Fig. 8. Illustration of foreground estimation with corrected motion vectors.

Algorithm 1: Procedure of R -Sampling.

Input : Motion vector set $\mathcal{V} = \{(\mathbf{p}_i, \mathbf{v}_i)\}$, FOE $\mathbf{F}$, number of sampled motion vectors k
Output: sampled motion vector set $\mathcal{V}_S$

- 1 Initialize sampled motion vector set $\mathcal{V}_S \leftarrow \{\}$, none-zero motion vector set $\mathcal{V}_N \leftarrow \{\}$
- 2 // Select none-zero motion vectors
- 3 **for** $(\mathbf{p}_i, \mathbf{v}_i) \in \mathcal{V}$ **do**
- 4 **if** $\|\mathbf{v}_i\| > 0$ **then**
- 5 $\mathcal{V}_N.append((\mathbf{p}_i, \mathbf{v}_i))$
- 6 // Sort $\mathcal{V}_N$ with distance to $\mathbf{F}$
- 7 Sort $\mathcal{V}_N$ using key: $\|\mathbf{p}_i - \mathbf{F}\|$
- 8 // Select top- k motion vectors from $\mathcal{V}_N$ as sampled motion vectors
- 9 **for** $i \in 0, 1, \dots, k-1$ **do**
- 10 $\mathcal{V}_S.append(\mathcal{V}_N[i])$
- 11 **return** $\mathcal{V}_S$

C. Foreground Extraction

1) *Ground Estimation:* When motion vectors only contain translational component, we have one key observation as follows:

Observation 2: Motion vectors of objects of the same height in the physical world have the same normalized magnitude.

Let $R_{q^t} = \sqrt{(x_q^t - \frac{\Delta X f}{\Delta Z})^2 + (y_q^t - \frac{\Delta Y f}{\Delta Z})^2}$ denote the distance between an image point q on frame $\mathcal{F}^t$ and the FOE. Then, motion vector $\mathbf{v}_q^t$ can be normalized by $R_{q^t}y$,

$$\frac{\mathbf{v}_q^t}{R_{q^t}y} = \frac{\Delta \mathbf{Z}}{fY_Q}. \quad (8)$$

It can be seen that the normalized vector is only related to Y_Q as $\Delta Z, f$ are the same for all points, which supports Observation 2.

As ground can be treated with the same and smallest height in the scene (when the ego agent is located on a flat road while encountering sloped regions which do not have the same height with the flat road, the sloped road surface itself and objects on those slopes, e.g., pedestrians and vehicles, are all treated as foreground), to estimate ground, we need to identify these vectors with the smallest normalized magnitude. However, motion vectors in regions with plain texture can be rather noisy. Moreover, these motion vectors on the ground can hardly be

absolutely equal. To tackle these issues, we first filter out those random vectors that do not point to the FOE and then adopt the Triangle method [20] to statistically establish a magnitude threshold. If a motion vector has a normalized magnitude that is less than the threshold, the corresponding coding block is considered as a ground coding block.

After obtaining a set of ground coding blocks, we then utilize a convex hull generation algorithm, e.g., Sklansky's algorithm [21] to generate the convex hull of the ground region. Non-ground coding blocks locate within the ground region in frame $\mathcal{F}^t$ constitute a set of foreground seed coding blocks, denoted as $\mathcal{S}^t$. Fig. 8(a) illustrates the result of ground estimation where red and blue cells represent ground and foreground coding blocks, and green polygon represents the ground convex.

2) *Region-Growing-Based Foreground Clustering:* Given $\mathcal{S}^t$, we utilize region-growing-based clustering algorithm to obtain foreground objects of interest. Specifically, for each coding block $b_i \in \mathcal{S}^t$, a breadth first search (BFS) is conducted around the four neighboring coding block of b_i . If the motion vector of a neighboring block b_j is similar to that of b_i and the average motion vector of the current cluster that b_i belongs to, b_j is appended to the cluster and the average motion vector of the current cluster is also updated. The latter condition is to avoid over-growing of an object. Fig. 8(b) illustrates the object clustering result of the previous example, where different colors represent different foreground objects. As shown in Fig. 8(b), many identified foreground objects may have holes as motion vectors are sparse and coarse. To obtain more complete foreground, as illustrated in Fig. 8(c), we further iteratively merge clusters with similar directions of their average motion vectors until no more clusters can be merged.² Finally, we use the Sklansky's algorithm to generate the convex contour for each merged cluster and obtain the foreground regions, as shown in Fig. 8(d).

D. Adaptive Video Encoding

1) *Uplink Bandwidth Estimation:* The quality of wireless uplink varies as the mobile agent moves. To best encode the video, the agent needs to continuously estimate the uplink bandwidth over time. Specifically, the uplink bandwidth is estimated with the amount of encoded data that has been successfully transmitted to the edge server. In DiVE, we measure successfully transmitted bytes with a sliding time window of 2 ms and accumulate measurements over a control interval of 100 ms. The

²When the ego agent is static and region-growing-based clustering is used to extract moving objects, this step is skipped.

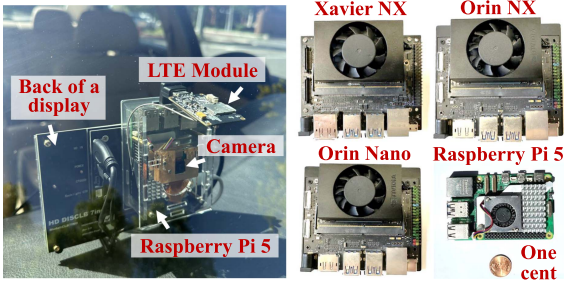


Fig. 9. Implementation of DiVE as a smart dashcam using four typical embedded devices.

uplink bandwidth is calculated by dividing the total transmitted bytes by the control interval.

2) *Optimal QP Assignment*: Given the estimated foreground regions and the current uplink bandwidth, as introduced in Subsection II-B, the optimal QP assignment for each coding block is equivalent to assign the optimal QP offset map to the foreground and the background coding blocks. We denote the optimal delta QP value between the foreground and the background coding blocks as δ . Based on the observation that larger extracted foregrounds are more likely to cover more real foregrounds, we design an adaptive δ which is proportional to the size of extracted foregrounds. More specifically, δ equals to the proportion of foreground size to the size of the whole frame multiplying a constant coefficient α . Then we fix the QP offset of foreground coding blocks to 0 and assign background coding blocks with a QP offset value equal to δ . After the frame $\mathcal{F}^t$ is encoded, it is put into the transmitting queue waiting to be sent to the edge server.

E. Motion-Vector-Based Offline Tracking

When encountering link outages, such as hard handovers between base stations or multipath fading, the agent should be able to detect this situation and act accordingly. Specifically, a timer is set up when an encoded frame becomes the first frame in the transmitting queue. If the timer times up before the frame is sent out, the agent considers that a link outage occurs. In this case, local motion vector based tracking is conducted for this and after frames until the link is recovered. More specifically, the mean of those motion vectors that reside in a detected bounding box in the previous frame is calculated, which is used to move the detected bounding box from the previous location in frame $\mathcal{F}^{t-1}$ to the corresponding new location in frame $\mathcal{F}^t$.

IV. IMPLEMENTATION

We use a desktop computer with an Intel i9-10980XE@3.8 GHz CPU and one Nvidia GTX 3090Ti GPU as the edge server. The video analytics service is built with Python. Specifically, we utilize PyAV for video decoding and Pytorch for model inference. A YOLOv5x model [22] pre-trained on BDD dataset [23] is deployed for car and pedestrian detection. We implement an agent prototype of DiVE as a dashcam deployed on the dashboard of a vehicle, as depicted in Fig. 9, using four different embedded devices, respectively, i.e., a Raspberry Pi 5 with an ARM Cortex-A76

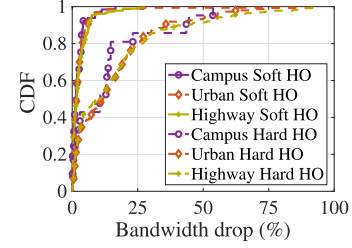


Fig. 10. CDFs of bandwidth drops caused by handovers in all three traces.

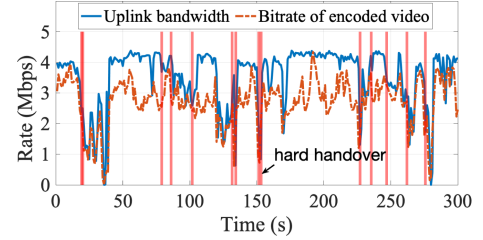


Fig. 11. One record of LTE CAT.1 measured bandwidth and the bitrate of the corresponding encoded video using DiVE.

CPU, a Jetson Xavier NX with an ARM Carmel CPU and two NVENC hardware encoders, a Jetson Orin Nano with an ARM Cortex-A78 CPU, and a Jetson Orin NX with an ARM Cortex-A78 CPU and a hardware encoder. Moreover, a CSI camera that supports different resolutions is used to capture video frames. The agent is connected to the server through a Quectel EC200N-CN LTE CAT.1 module. Agent code is written in C++ on these devices. We use *libx264* library on the Raspberry Pi 5 and Jetson Nano and use Nvidia Jetpack and Multimedia API on Nvidia Jetson TX2 to extract motion vectors and execute differential encoding. The GOP size is set to 10. *WebSocket* is used for full-duplex real-time communication. We employ multi-threading to handle video encoding and data transmission on both the client and server sides.

The prototype is also utilized for collection of real-world dataset *DiVESet* introduced in Section II-E4. Specifically, the LTE CAT.1 bandwidth is measured on the agent with iPerf and the round-trip time (RTT) is measured by pinging the server and the average measured RTT is 8.8 ms. Three real-world bandwidth traces are collected in Shanghai city, i.e., on our campus, in urban scenario and in highway scenario, respectively. Fig. 10 plots the CDFs of bandwidth drops caused by soft and hard handovers. It can be seen that hard handovers can cause severe bandwidth drop. For example, over 20% handovers cause a bandwidth drop of above 25%. Fig. 11 plots one trace of LTE CAT.1 bandwidth measured in the urban scenario and the bitrate of the corresponding encoded video using DiVE. It can be seen that the bitrate of DiVE-encoded video well adapts to the bandwidth change of LTE CAT.1 uplink.

V. PERFORMANCE EVALUATION

A. Methodology

We conduct our experiments on RobotCar, nuScenes and DiVESet to evaluate the performance of DiVE. For nuScenes, we remove video clips collected at night in test set as almost all

TABLE I
SUMMARY OF DATASETS

Name	# FPS	# videos	# frames	# cars	# peds.
nuScenes	12	50	9605	45605	10221
RobotCar	16	8	8150	19365	25423
DiVESet-Campus	30	10	9000	13770	6879
DiVESet-Urban	30	10	9000	49363	6015
DiVESet-Highway	30	10	9000	34029	-

motion vectors are calculated to be zero at night and randomly select 50 video clips from the rest video clips. For RobotCar, we randomly select 8 video clips, each consisting of approximately 1000 frames. For DiVESet, we utilize traces collected in campus, urban and highway to conduct our experiments. The details of datasets are presented in Table I.

We compare DiVE with the following video analytics schemes:

- O^3 [4]: A mobile object detection system that uploads key frames to the cloud server for object detection and uses key frame detection results to correct object tracking results for other frames.
- EAAR [3]: An object detection system designed for mobile AR devices to conduct parallel streaming and inference on key frames. It utilizes the cached detection results of key frames for ROI encoding and conducts object tracking on other frames. The default QP is 30 and 40 for high-quality and low-quality encoding.
- DDS [5]: A video analytics system for deep learning inference, which first uploads low-quality videos to the cloud server for detection and utilizes the feedback detected regions to guide the edge device to re-upload the corresponding high-quality videos of the corresponding part to the cloud server for more accurate detection.
- EdgeDuet [10]: An edge-assisted object detection system designed for mobile devices, which utilizes a light-weight detection model for large object detection and differentially encodes the frame with low quality on region containing large objects. The encoded frame is then uploaded to the edge server for large model inference. Kernel Correlation Filter (KCF) based object tracker is used for achieving local frame detection results.

For fair comparison, we utilize $x264$ for video/key frame encoding in all schemes. For O^3 and EAAR, we utilize the same motion vector based tracking methods. For DDS, we use the frame-level transmission which is the same as DiVE instead of raw segment-level transmission which will lead to high latency. For EdgeDuet, we use a YOLOv5n model as local large object detector.

We consider the following two metrics to evaluate the performance of different schemes:

- **Precision:** We utilize Average Precision (AP), a popular metric for the object detection task, to represent the precision of DNN inference results. We utilize the object detection results of raw frames at the edge server as the ground truth to calculate AP.³

³Note that we utilize DNN inference result on raw frames as ground truth rather than human-annotated labels. Thus, any detection precision loss is caused by frame processing of video analytic system rather than DNN itself. This is consistent with previous work [9], [11].

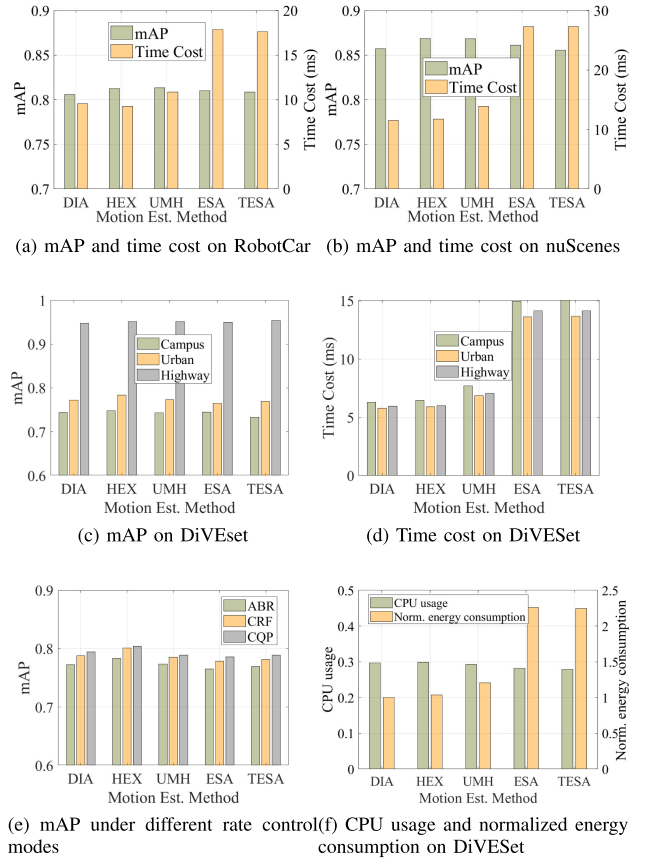


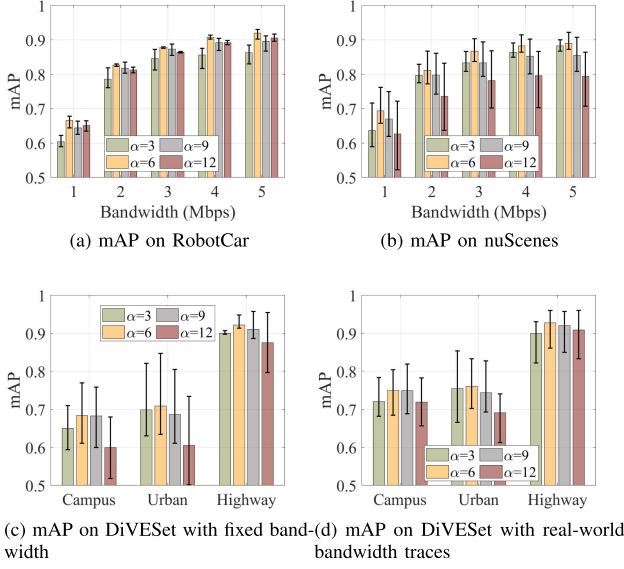
Fig. 12. Effect of different motion estimation methods.

- **Response Time:** We define Response Time as the average duration from the moment a frame is captured by the camera until the mobile agent receives the final inference result.

B. Parameter Configuration

1) **Effect of Motion Estimation Method:** We execute a set of experiments to evaluate the performance of different motion estimation methods. In the $x264$ encoder, there are five motion estimation methods, i.e., diamond search algorithm (DIA), hexagon-based search algorithm (HEX), uneven multi-hexagon search algorithm (UMH), transformed exhaustive search algorithm (TESA) and exhaustive search algorithm (ESA), ranked in ascending order of computational complexity.

We first conduct experiments on all three datasets to evaluate the performance of different motion estimation algorithm. We use each motion estimation method to extract motion vectors when encoding each frame on RobotCar, nuScenes and DiVESet. We set bandwidth to 2 Mbps on nuScenes and RobotCar and use real-world bandwidth traces on DiVESet. Fig. 12 plots the mAP and time cost of each motion estimation method on three datasets. The result shows that HEX and UMH achieve almost the same mAP and higher than that of DIA, ESA and TESA while HEX has lower time cost than UMH on both datasets. The reason is that simple motion estimation method like DIA will lead to low quality of motion vectors while overcomplicated motion estimation methods like ESA and TESA will also lead

Fig. 13. Effect of Constant Coefficient α .

to more noise in motion vectors as motion estimation methods are designed for obtaining minimal residual data but not real object matching.

We also conduct an experiment on DiVESet to demonstrate the performance of different motion estimation methods under all three different rate control modes in $x264$, i.e., Average Bitrate (ABR) mode, Constant Rate Factor (CRF) mode and Constant Quantization Parameter (CQP) mode. We use real-world bandwidth traces for ABR mode and set QP value of foreground to 26 for CQP and CRF mode to conduct the experiment on Raspberry Pi 5 and calculate the mAP, CPU usage and normalized energy consumption of different motion estimation methods. Fig. 12(e)(f) plots the mAP, CPU usage and normalized energy consumption of each motion estimation method. The result shows that all motion estimation methods have almost the same CPU usage while HEX achieves the highest AP under all rate control modes and almost as low normalized energy consumption as DIA. Therefore, we choose HEX for motion estimation in DiVE to achieve a good trade-off between time cost, energy consumption and quality of calculated motion vectors.

C. Effect of Constant Coefficient α

In this experiment, we explore the effect of constant coefficient α in Optimal QP Assignment. We set the motion estimation method to HEX and conduct experiments on all three datasets to evaluate the performance of different α value. We vary α from 3 to 12 with an interval of 3 to conduct the experiments. For nuScenes and RobotCar, we vary bandwidth from 1Mbps to 5Mbps with an interval of 1 Mbps. For DiVESet, we utilize real-world bandwidth traces and fixed 2Mbps bandwidth to conduct experiments. Fig. 13 plots the mAP with different α under each bandwidth on all three datasets. It can be seen from the result that $\alpha = 6$ achieves the highest mAP across all bandwidth settings on three datasets. The reason is that low α will lead to insufficient bitrate allocation for foregrounds while too large α

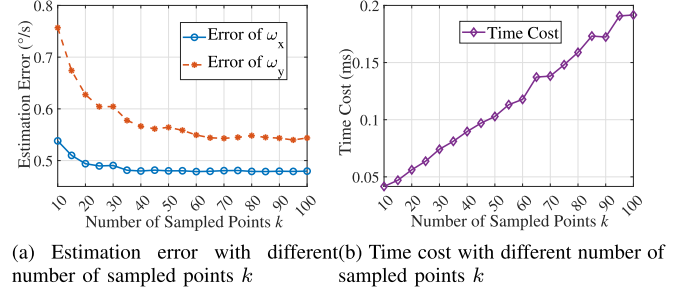
Fig. 14. Effect of different number of sampled points k .

TABLE II
CLASSIFICATION ACCURACY OF EGO MOTION JUDGEMENT IN DIFFERENT SCENARIOS

Motion Category	Precision	Recall	F1-score
Campus-Moving	0.996	0.998	0.997
Campus-Static	0.979	0.964	0.971
Urban-Moving	0.988	0.984	0.986
Urban-Static	0.950	0.961	0.955
Highway-Moving	0.994	0.987	0.990
Highway-Static	-	-	-

may lead to few detection accuracy increase on foregrounds but low detection accuracy on backgrounds. Therefore, we set $\alpha = 6$ in DiVE for all experiments in our work.

1) *Effect of the Number of Sampled Points in R-Sampling:* In this experiment, we explore the effect of number of R-sampling points k in Rotational Component Elimination. We conduct rotation estimation on KITTI and vary the number of sampled points k from 10 to 100 with an interval of 5. We utilize the same video clips as experiments in Section III-B. We calculate the estimation error of rotational speed and time cost of RANSAC under each setting. Fig. 14 plots the results. It can be seen from the result that the estimation error decreases when k increases and the time cost is linear to k . Moreover, the estimation error converges when k reaches 70. The reason is that R-sampling is efficient and a few accurate motion vectors will lead to high motion estimation accuracy, more coarse motion vectors will not increase the motion estimation accuracy any more. As a result, we choose $k = 70$ in R-sampling to achieve good rotation estimation accuracy while maintaining as low time cost as possible.

D. Effectiveness of Ego Motion Judgement

We evaluate the effectiveness of Ego Motion Judgement in DiVE through experiments on campus, urban and highway in DiVESet. The ground truth of motion state of the ego vehicle is achieved with the logged GPS coordinates. We calculate the precision and recall of the classification results under each scenario and the results are illustrated in Table II. It can be seen that Ego Motion Judgement in DiVE can achieve an average precision of 0.981 and an average recall of 0.979. Note that the lowest F1-score is achieved when the ego vehicle is static in urban. The reason is that heavy traffic in urban will lead to more motion in the frame and cause a slight accuracy decrease. Overall, the average F1-score is 0.980 which demonstrates the superior performance and robustness of Ego Motion Judgement.

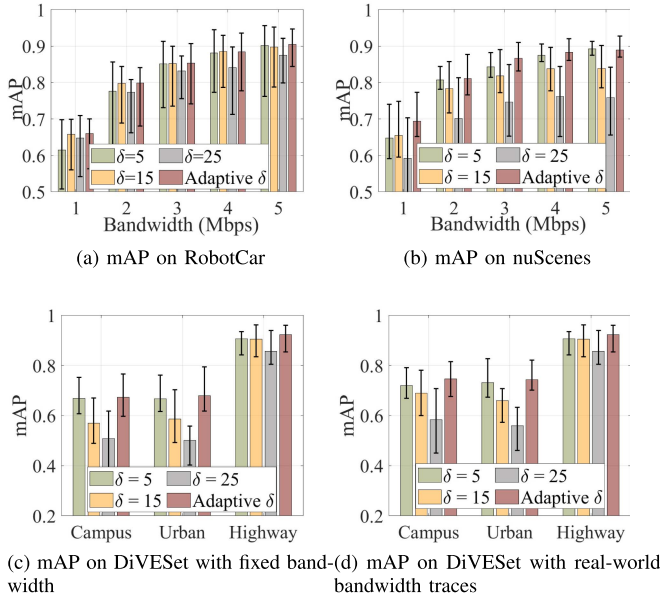


Fig. 15. Effectiveness of Optimal QP Assignment.

E. Effectiveness of Optimal QP Assignment

We conduct experiments on all three datasets to evaluate the effectiveness of Optimal QP Assignment. We vary δ from 5 to 25 with an interval of 10 along with adaptive δ in our design. For nuScenes and RobotCar, we vary bandwidth from 1 Mbps to 5 Mbps with an interval of 1 Mbps. For DiVESet, we utilize real-world bandwidth traces and fixed 2 Mbps bandwidth to conduct experiments. Fig. 15 plots the mAP with different δ under each bandwidth on all three datasets. It can be seen from the result that adaptive δ achieves the highest mAP under most bandwidth settings. Moreover, adaptive δ achieves much better performance than $\delta = 5$ when the bandwidth is 1 Mbps and the reason is that higher bitrate for foregrounds can lead to more inference accuracy benefits when the bandwidth is low. The experiment results demonstrate the efficiency of Optimal QP Assignment.

F. Effectiveness of Foreground Extraction

In this experiment, we set x264 encoder to Constant Rate Factor (CRF) mode which can fix the coding block encoding quality with fixed QP value and fix the QP value of foregrounds to 0 to achieve almost the same quality with raw frames. The QP value of background coding blocks varies from 4 to 36 with an interval of 8. Fig. 16 plots the AP of varying QP value of backgrounds and the average normalized video size on all three datasets. We set the normalized video size to 1 when the background QP value equals to 4. The result shows that the AP decreases slowly with the increase of the QP value of backgrounds. When the QP value drops to 20, the AP of the pedestrian and car is 0.985, 0.993 on RobotCar, 0.974, 0.979 on nuScenes, the AP of pedestrian on Campus and Urban is 0.967, 0.947 and the AP of car is 0.979, 0.981, 0.982 on Campus, Urban and Highway, respectively, which shows nearly no loss on the accuracy of detection. The normalized video size is 0.396, 0.397, 0.544 on RobotCar, nuScenes and DiVESet, respectively,

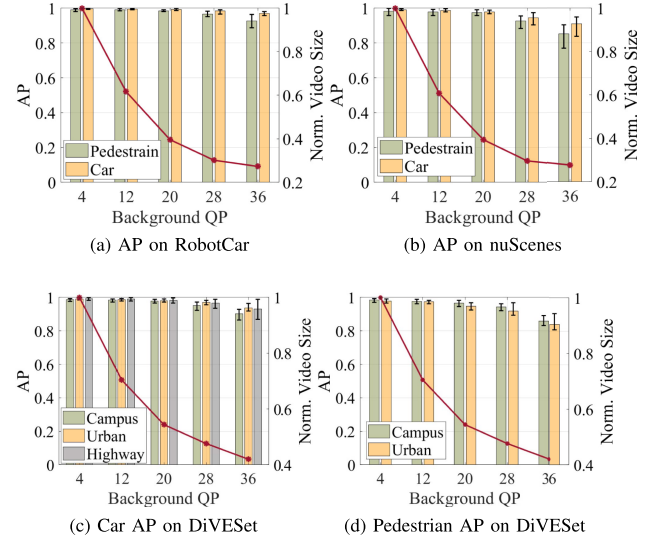


Fig. 16. Effectiveness of Foreground Extraction.

which shows significant video size reduction compared with raw video. Even when the QP value of backgrounds drops to 36 which leads to severe quality loss of video frames, the AP of pedestrian and car can still be up to 0.926, 0.970 on RobotCar, and 0.852, 0.911 on nuScenes, and the AP of pedestrian is 0.859, 0.842 on Campus and Urban, the AP of Car is 0.902, 0.936, 0.930 on Campus, Urban and Highway, respectively. The normalized video size is 0.273, 0.277, 0.420 on RobotCar, nuScenes and DiVESet, respectively. The result demonstrates the effectiveness of Foreground Extraction in DiVE.

G. Effectiveness of Motion-Vector-Based Offline Tracking

We conduct experiments on nuScenes, RobotCar and DiVESet to evaluate the effectiveness of Motion-vector-based Offline Tracking in DiVE. For nuScenes and RobotCar, we set the bandwidth to 2 Mbps and add 1 s link interruptions, i.e., bandwidth drops to 0 Mbps to the network. The interval between two interruptions is set from 5 s to 20 s with an interval of 5 s to simulate different mobile wireless network scenarios. For DiVESet, we utilize the recorded bandwidth traces to conduct the experiment. Fig. 17 plots the AP of car and pedestrian detection in different scenarios. It can be seen from the results that when offline tracking is enabled, the mAP increases on all network scenarios. Specifically, the mAP increases by 12.8% and 8.6% in detection accuracy on RobotCar and nuScenes with link interruption interval of 5 s when Motion-vector-based Offline Tracking is enabled. The AP increases by 4.4% and 2.5% in detection accuracy on pedestrian and car when Motion-vector-based Offline Tracking is enabled on DiVESet. Fig. 17(d) plots a bandwidth trace lasting for 100 s measured in urban scenario and the inference accuracy of DiVE with Motion-vector-based Offline Tracking enabled and disabled. The result shows that when Motion-vector-based Offline Tracking is enabled, DiVE can still maintain high inference accuracy when bandwidth drops severely in a short time, while severe accuracy decrease occurs without offline tracking.

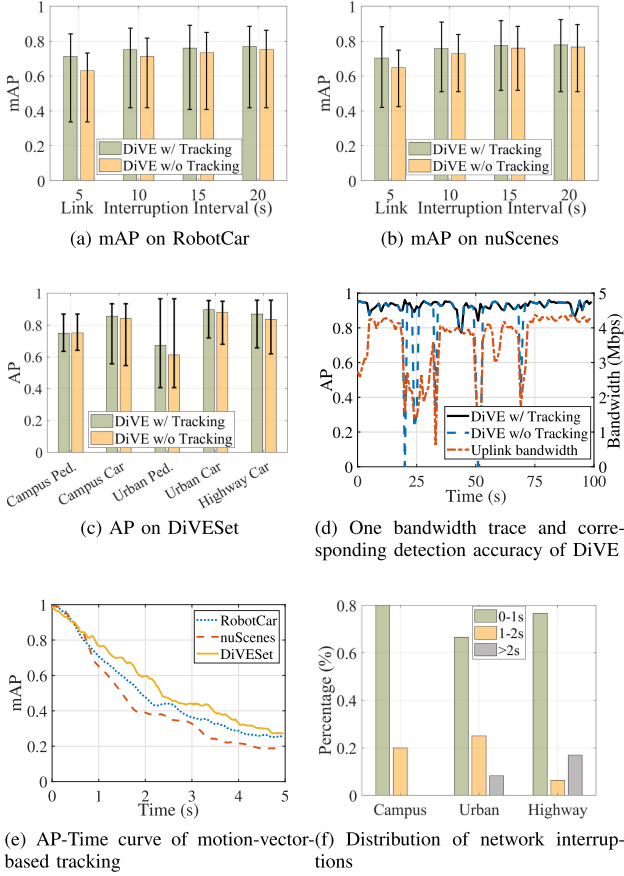


Fig. 17. Effectiveness of Motion-vector-based Offline Tracking.

We also conduct an experiment to illustrate the performance of motion-vector-based tracking. We maintain motion-vector-based tracking for 5 seconds and plot the AP-time curve on all three datasets. Fig. 17(e) plots the result. It can be seen from the result that tracking AP on all datasets is larger than 0.65 when tracking duration is 1 s and larger than 0.4 when duration is 2 seconds. Moreover, we also calculate the distribution of network interruptions on real-world bandwidth traces of DiVESet. Fig. 17(f) plots the distribution of network interruptions in different scenarios. It can be seen from the result that link interruptions less than 1 s account for more than 65% in all scenarios. Therefore, Motion-vector-based Offline Tracking can handle most of network interruptions in real-world scenarios.

H. Impact of Different Motion State

We manually divide all three datasets into three different motion states, i.e., static, moving straight, turning, and record the AP of pedestrian and car detection results under different motion states in Fig. 18. In these experiments, video bitrate is set to 2 Mbps. The result shows that DiVE consistently achieves pedestrian detection AP exceeding 0.6 across all datasets, with an average pedestrian detection AP of 0.759, 0.714 and 0.792 when the ego vehicle is stationary, moving straight and turning. The car detection AP is higher than 0.8 in all datasets and the average car detection AP is 0.927, 0.909 and 0.888 when the

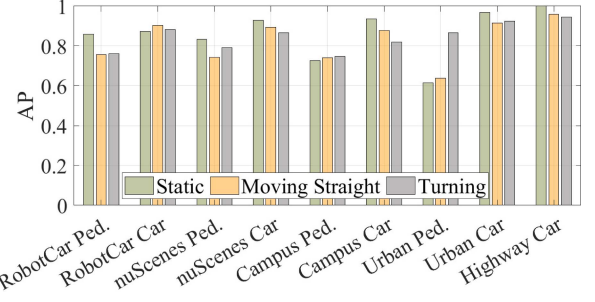


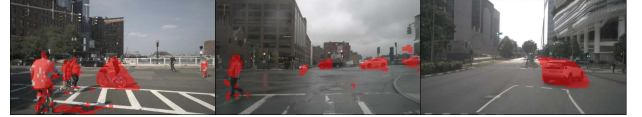
Fig. 18. Impact of different motion states.



(a) Extracted foregrounds when vehicle is moving straight.



(b) Extracted foregrounds when vehicle is turning.



(c) Extracted foregrounds when vehicle is static.

Fig. 19. Illustration of extracted foreground under different motion states.

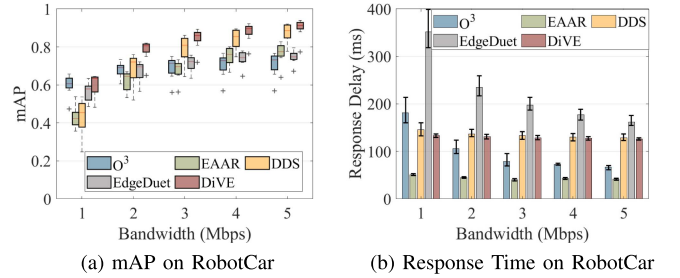


Fig. 20. Comparison of different schemes under different bandwidth on RobotCar.

ego vehicle is static, moving straight and turning. Moreover, the highest car detection AP is achieved when the ego vehicle is static. The reason is that the ego vehicle stops in most cases due to a red light and other moving vehicles can be captured as foreground more accurately. Fig. 19 illustrates some samples of extracted foregrounds when the ego vehicle is moving straight, turning and static.

I. End-to-End Performance Evaluation

We evaluate the end-to-end performance of different video analytics schemes on accuracy and response time across varying network scenarios and multiple datasets. Fig. 20 and Fig. 21 plot the AP and response time of different schemes across varying

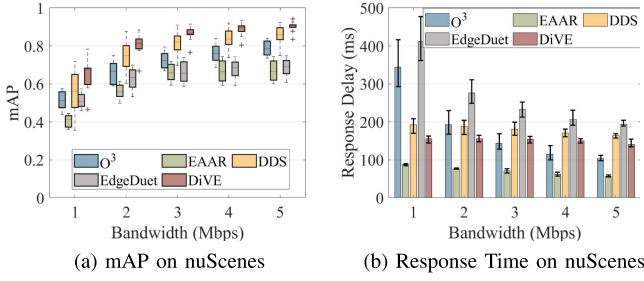


Fig. 21. Comparison of different schemes under different bandwidth on nuScenes.

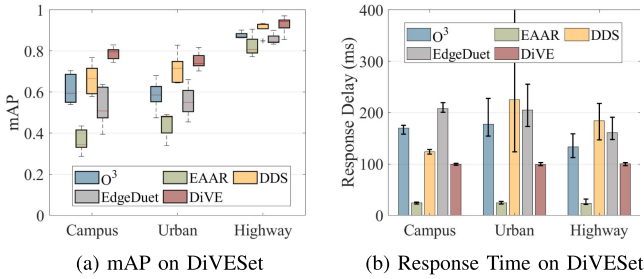


Fig. 22. Comparison of different schemes under different bandwidth on DiVESet.

network bandwidth from 1 Mbps to 5 Mbps with an interval of 1 Mbps. It can be seen from the result that DiVE achieves the highest mAP under each network scenario on both RobotCar and nuScenes. When compared to DDS, which exhibits the most similar performance with DiVE, DiVE improves mAP by 2.8%–39.1% on RobotCar and by 4.7%–17.6% on nuScenes. Notably, DiVE outperforms other schemes especially when bandwidth is low.

Fig. 20(b) and Fig. 21(b) show that DiVE can achieve lower than 134 ms/156 ms response time under different bandwidth, which is 1.7%–8.4% and 14.0%–19.1% lower than DDS on RobotCar and nuScenes, respectively. Although EAAR can achieve lower response time than DiVE, its detection mAP is significantly lower than DiVE. We explain the above results as follows: O^3 and EAAR achieve poor precision under different bandwidth because they do not utilize the temporary redundancy in the video. Instead, DiVE captures this redundant information and utilizes them to aid in differential video encoding. Moreover, as EdgeDuet needs to run model inference on mobile agent, the response time of EdgeDuet is significantly larger than DiVE and this leads to more local frame tracking which causes severe drop of detection accuracy. Although local tracking has a very low time cost, prolonged tracking will lead to a severe accuracy decrease. The DDS scheme requires two times of uploading for final detection results, which leads to high response time.

Fig. 22 plots the mAP and response time of different schemes in campus, urban, and highway scenarios with real-world bandwidth traces on DiVESet. The results show that DiVE achieves the highest mAP in all scenarios. Compared with the best-performing comparison scheme DDS, DiVE improves the mAP by 19.0%, 5.6% and 1.7% on Campus, Urban and Highway datasets. Moreover, DiVE achieves lower than 101 ms response

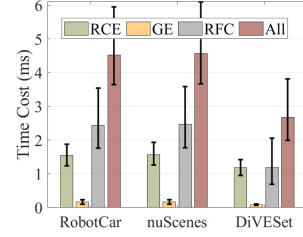


Fig. 23. Overhead of core components in DiVE.

time under different bandwidth on all three datasets, which is 19.4%, 56.0% and 45.7% lower than DDS on Campus, Urban and Highway datasets. The significant response time gain benefits from the network monitoring and local tracking scheme in DiVE to deal with bandwidth fluctuation.

J. Overhead Analysis of DiVE

We conduct an experiment to demonstrate the overhead of core components of DiVE on all three datasets on Raspberry Pi 5. Fig. 23 illustrates the overhead of Rotation Component Elimination (RCE), Ground Estimation (GE), Region-growing-based Foreground Clustering (RFC) and all extra computation of DiVE. It can be seen from the result that the average overhead of RCE, GE, RFC is less than 1.6 ms, 0.2 ms, 2.5 ms on all three datasets and the average overhead of all extra computation of DiVE is less than 4.5 ms, 4.6 ms, 2.7 ms on RobotCar, nuScenes and DiVESet. The result demonstrates that DiVE is light-weight and causes low extra computation overhead.

K. Processing Speed and Resource Consumption

We evaluate the real-time processing performance and resource consumption of DiVE on four mobile devices, i.e., Raspberry Pi 5 (RPI 5), Jetson Xavier NX, Orin Nano, and Orin NX. We set the frame captured by the camera to three different resolutions, i.e., 480P (640x480), 720P (1280x720), and 1080P (1920x1080). Fig. 24 reports the average number of FPS and usage of CPU and GPU on all devices. It can be seen from Fig. 24(a) that when the camera resolution is 480P, the processing speed of DiVE is at least 68FPS and can be up to 109FPS on RPI 5. When the resolution is 720p, the process speed is more than 30FPS on all devices. Even when the resolution is 1080P, the process speed can still be at least 14FPS. It can also be seen from Fig. 24(b) that on RPI 5 and Orin Nano which only use CPU for process, the CPU usage increases when resolution increases and the maximum usage of CPU is 42.5% on RPI 5. As we utilize NVENC for MV extraction and video encoding on Xavier NX and Orin NX, the GPU usage on these two devices is about 20% and the CPU usage is lower than that on the other two devices.

VI. DISCUSSION

In this section, we discuss issues that DiVE may encounter in practice.

Limitations and potential solutions of DiVE: The limitations of DiVE mainly arise from two aspects. First, as almost all

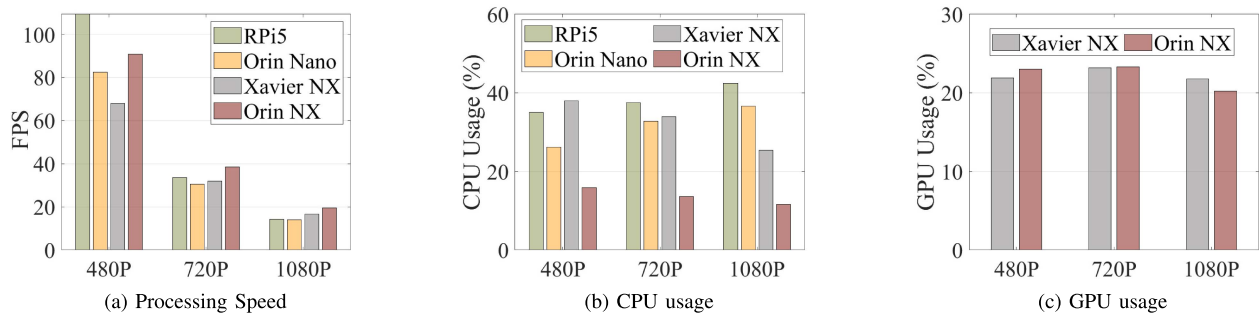


Fig. 24. Process speed and CPU/GPU usage of DiVE under different video resolution.

motion vectors are calculated to be zero at low illumination scenario, e.g., night, and DiVE solely relies on motion vectors to extract foregrounds and differentially encode frames. Therefore, DiVE can not work normally in such situation. One potential solution is to compress video frames uniformly and stream the compressed frames for DNN inference as a complement of DiVE at low illumination scenario.

Second, the core assumption of DiVE is foreground objects like pedestrian, vehicles, etc, stand on the ground. Therefore, objects suspended above the ground, such as traffic lights and traffic signs, are not explicitly modeled and may not be extracted as foregrounds successfully in DiVE. One potential solution is to utilize lightweight rule-based detection module operating on the extracted backgrounds by DiVE. For example, traffic lights can be identified using color and shape cues, and traffic signs can be recognized by simple geometric templates [24]. Such detectors can run in real time on CPU without increasing much latency, making them compatible with the DiVE framework.

Applicability of DiVE for various video codecs: In this work, DiVE is implemented based on *x264* which is an open-source video codec that implements the H.264 video coding standard. Although newer video coding standards like H.265, H.266 introduce additional features compared to H.264, e.g., tile feature in H.265, but they all share the same fundamental video encoding pipeline, i.e., block-based motion estimation and compensation, transform and quantization of residuals and entropy encoding, in which motion estimation remains a mandatory step. As DiVE simply relies on motion vectors rather than any other codec-specific data, the proposed foreground extraction and differential encoding mechanism can be naturally ported to other codecs.

VII. RELATED WORK

Traditional Video Streaming: Plenty of research is done to increase the Quality of Experience (QoE) of users for watching video under fluctuated bandwidth [25], [26], [27], [28], [29], [30]. Sun et al. [25] proposed a bandwidth prediction system called CS2P which clusters current network bandwidth to a pre-learned cluster and utilizes a HMM model to predict the evolution of bandwidth for proper bitrate selection of video. Mao et al. [26] proposed a reinforcement learning based video streaming system called Pensieve which utilizes a neural network to select bitrates for future video chunks based on observations of

history bandwidth. Yeo et al. [27] proposed a video streaming system called NAS which utilizes a super-resolution model at client side to up-scale received low-resolution video frames for better QoE. Yeo et al. [29] designed a system called NEMO which applies neural super-resolution to a few selected frames and utilizes the outputs to benefit other frames. These video streaming schemes are designed to improve QoE of humans under various bandwidth and do not consider about how to keep high accuracy of DNN inference on mobile agents.

Some studies utilize Markov Decision Process (MDP) or Deep Reinforcement Learning (DRL) to handle live video streaming [31], [32]. Such video streaming schemes also focus on improving the QoE of humans. Moreover, the adaptation granularity of MDP and DRL based video streaming schemes are usually at video chunk level (seconds) which can not satisfy the requirement of frame level decision on mobile agent. Besides, MDP and DRL based video streaming schemes will lead to significant additional computational overhead and are hard to be deployed on computationally constrained mobile agents.

Video Analytics Systems: Differing from traditional video streaming system designed for humans. A large number of studies focus on maintaining high accuracy of DNN inference at edge device by uploading video to the cloud server for inference [2], [4], [5], [6], [7], [8], [9], [10], [11], [12], [33], [34], [35], [36], [37], [38]. Some studies try to reconfigure video encoding parameters like resolution, frame rate and QP value during streaming to meet the bandwidth variation. Zhang et al. [11] designed a system called AWStream which combines offline and online profiling to choose encoding parameters. Li et al. [12] designed Reducto, an on-camera filtering system which filters out frames based on per frame difference. However, such schemes do not consider the importance difference inner frame and usually need offline profiling which can not be utilized for real-time video analytics.

Another direction is to choose key frames for uploading and inference based on difference between frames and conduct optical flow/motion vector based tracking with cached key frame detection results locally [2], [3], [4]. Such methods do not fully utilize the temporary redundancy between video frames and simple object tracking will lead to severe accuracy degradation on long frame sequences.

Besides, running small models on edge device for assistance like large object detection [10], estimation of importance of coding blocks [9] is also proposed in previous studies. Such

methods require for high computility at edge device and can not adapt to various mobile scenarios due to poor performance of cheap model.

Some studies also focus on foreground extraction and utilize differential encoding to reduce bandwidth consumption. Wang et al. [37] designed a system called VaBUS which constructs the background region of the surveillance camera and uploads differentially encoded frames. Wang et al. [36] proposed an algorithm called Orchestra, which divides video into different zones and assigns different qualities for encoding. However, these methods can only work with static backgrounds like video captured by surveillance camera and can not copy with highly dynamic mobile scenarios.

VIII. CONCLUSION

In this paper, a video analytics system designed for mobile agents called DiVE is proposed. In DiVE, mobile agent determines its motion state and extracts interested foregrounds based on low-cost motion vectors calculated by video codec for differential encoding. Multiple observations and results of empirical studies are utilized to correct motion vectors and cluster foregrounds. As a result, DiVE can copy with coarse motion vectors and extract foregrounds accurately. It can be easily implemented on various mobile devices. Extensive experiment results demonstrate that DiVE can achieve high inference accuracy with low response time under various network scenarios.

REFERENCES

- [1] X. Wang, J. Ye, and J. C. Lui, "Online learning aided decentralized multi-user task offloading for mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 4, pp. 3328–3342, Apr. 2024.
- [2] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. ACM Sensys*, 2015, pp. 155–168.
- [3] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. ACM SIGCOMM*, 2019, pp. 1–16.
- [4] M. Hanyao, Y. Jin, Z. Qian, S. Zhang, and S. Lu, "Edge-assisted online on-device object detection for real-time video analytics," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2021, pp. 1–10.
- [5] K. Du et al., "Server-driven video streaming for deep learning inference," in *Proc. ACM SIGCOMM*, 2020, pp. 557–570.
- [6] W. Zhang et al., "Elf: Accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *Proc. ACM Mobicom*, 2021, pp. 201–214.
- [7] S. Liu, T. Wang, J. Li, D. Sun, M. Srivastava, and T. Abdelzaher, "Adamask: Enabling machine-centric video streaming with adaptive frame masking for DNN inference offloading," in *Proc. 30th ACM Int. Conf. Multimedia*, 2022, pp. 3035–3044.
- [8] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: Optimizing neural network queries over video at scale," *Vldb Endowment*, vol. 10, no. 11, pp. 1586–1597, 2017.
- [9] K. Du, Q. Zhang, A. Arapin, H. Wang, Z. Xia, and J. Jiang, "AccMPEG: Optimizing video encoding for accurate video analytics," in *Proc. Mach. Learn. Syst.*, 2022, pp. 450–466.
- [10] Z. Yang et al., "EdgeDuet: Tiling small object detection for edge assisted autonomous mobile vision," *IEEE/ACM Trans. Netw.*, vol. 31, no. 4, pp. 1765–1778, Aug. 2023.
- [11] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzyniek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proc. ACM SIGCOMM*, 2018, pp. 236–252.
- [12] Y. Li et al., "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proc. ACM SIGCOMM*, 2020, pp. 359–376.
- [13] Y. Cheng et al., "Grace: Loss-resilient real-time video through neural codecs," in *Proc. USENIX Symp. Networked Syst. Des. Implementation*, 2024, pp. 509–531.
- [14] L.-L. Wang and W.-H. Tsai, "Camera calibration by vanishing lines for 3-D computer vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 4, pp. 370–376, Apr. 1991.
- [15] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3354–3361.
- [16] H. Caesar et al., "nuscenes: A multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11621–11631.
- [17] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 year, 1000 km: The oxford RobotCar dataset," *Int. J. Robot. Res.*, vol. 36, no. 1, pp. 3–15, 2017.
- [18] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Trans. Image Process.*, vol. 9, no. 2, pp. 287–290, Feb. 2000.
- [19] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [20] G. W. Zack, W. E. Rogers, and S. A. Latt, "Automatic measurement of sister chromatid exchange frequency," *J. Histochemistry Cytochemistry*, vol. 25, no. 7, pp. 741–753, 1977.
- [21] J. Sklansky, "Finding the convex hull of a simple polygon," *J. Algorithms*, vol. 1, no. 2, pp. 79–83, 1982.
- [22] "YOLOv5," 2026. Online. Available: <https://github.com/ultralytics/yolov5>
- [23] F. Yu et al., "Bdd100 k: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2636–2645.
- [24] Y. Saadna and A. Behloul, "An overview of traffic sign detection and classification methods," *Int. J. Multimedia Inf. Retrieval*, vol. 6, no. 3, pp. 193–210, 2017.
- [25] Y. Sun et al., "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc. ACM SIGCOMM*, 2016, pp. 272–285.
- [26] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. ACM SIGCOMM*, 2017, pp. 197–210.
- [27] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware internet video delivery," in *Proc. USENIX Operating Syst. Des. Implementation*, 2018, pp. 645–661.
- [28] X. Zhang, Y. Ou, S. Sen, and J. Jiang, "Sensei: Aligning video streaming quality with dynamic user sensitivity," in *Proc. USENIX Networked Syst. Des. Implementation 21*, 2021, pp. 303–320.
- [29] H. Yeo, C. J. Chong, Y. Jung, J. Ye, and D. Han, "NEMO: Enabling neural-enhanced video streaming on commodity mobile devices," in *Proc. ACM Mobicom*, 2020, pp. 1–14.
- [30] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, "Neural-enhanced live streaming: Improving live video ingest via online learning," in *Proc. ACM SIGCOMM*, 2020, pp. 107–125.
- [31] K. Khan and W. Goodridge, "S-MDP: Streaming with markov decision processes," *IEEE Trans. Multimedia*, vol. 21, no. 8, pp. 2012–2025, Aug. 2019.
- [32] L. Cui, D. Su, S. Yang, Z. Wang, and Z. Ming, "TCLiVi: Transmission control in live video streaming based on deep reinforcement learning," *IEEE Trans. Multimedia*, vol. 23, pp. 651–663, 2021.
- [33] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proc. ACM Mobicom*, 2015, pp. 426–438.
- [34] T. Yuan, L. Mi, W. Wang, H. Dai, and X. Fu, "AccDecoder: Accelerated decoding for neural-enhanced video analytics," in *Proc. IEEE Conf. Comput. Commun.*, 2023, pp. 1–10.
- [35] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the edge-cloud barrier for real-time advanced vision analytics," in *Proc. USENIX HotCloud Workshop*, 2019, p. 18.
- [36] W. Wang, B. Wang, L. Zhang, and H. Huang, "Sensitivity-aware spatial quality adaptation for live video analytics," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 8, pp. 2474–2484, Aug. 2022.
- [37] H. Wang et al., "VaBUS: Edge-cloud real-time video analytics via background understanding and subtraction," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 1, pp. 90–106, Jan. 2023.
- [38] X. Xiao, Y. Zuo, M. Yan, W. Wang, J. He, and Q. Zhang, "Task-oriented video compressive streaming for real-time semantic segmentation," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 14396–14413, Dec. 2024.



Hongzi Zhu (Senior Member, IEEE) received the BS and ME degrees in computer science from Jilin University, China, in 2001 and 2004, respectively, and the PhD degree in computer science from Shanghai Jiao Tong University, in 2009. He was a postdoctoral fellow with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology in 2009 and also with the Department of Electrical and Computer Engineering, University of Waterloo in 2010. He is currently a professor with the School of Computer Science, Shanghai Jiao Tong University. His research interests include Internet of Things and mobile computing. He was the recipient of the Best Paper Award from IEEE ICDCS'25, ACM SenSys'24, and IEEE GLOBECOM'16. He is an associate editor for *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Vehicular Technology*, and *IEEE Internet of Things Journal*. He is an IEEE VTC distinguished lecturer. For more information, please visit <http://lion.sjtu.edu.cn>.



Jiangang Shen (Member, IEEE) received the BS degree from Shanghai Jiaotong University, in 2019. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Shanghai Jiaotong University. His research interests include Internet of Things and wireless network.



Liang Zhang (Member, IEEE) received the BE degree in computer science and technology from Northeastern University, China, in 2018. She is currently working toward the PhD degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. Her research interests include stream processing and resource scheduling in the cloud or edge computing environment.



Yunzhe Li Yunzhe Li received the BS degree in computer science from Wuhan University, in 2021. He is currently working toward the PhD degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include mobile sensing and edge computing.



Shan Chang received the PhD degree in computer software and theory from Xian Jiaotong University, in 2013. From 2009 to 2010, she was a visiting scholar with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. She was also a visiting scholar with the BBRC Research Lab, University of Waterloo, from 2010 to 2011. She is currently an associate professor with the Department of Computer Science and Technology, Donghua University, Shanghai. Her research interests include security and privacy in mobile networks and sensor networks. She is a member of IEEE.



Jie Wu (Fellow, IEEE) is currently the director of the Center for Networked Computing and Laura H. Carnell professor with Temple University. His research interests include mobile computing wireless networks, routing protocols, network trust and security, distributed algorithms, applied machine learning, and cloud computing. He is on the editorial board of *IEEE/ACM Transactions on Networking*.



Minyi Guo (Fellow, IEEE) received the PhD degree in computer science from the University of Tsukuba, Japan. He is currently Zhiyuan Chair Professor with the School of Computer Science, Shanghai Jiao Tong University, China. His research interests include parallel/distributed computing, compiler optimizations, embedded systems, pervasive computing, Big Data, and cloud computing. He is on the editorial board of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing* and *Journal of Parallel and Distributed Computing*.