RT3C: Real-Time Crowd Counting in Multi-Scene Video Streams via Cloud-Edge-Device Collaboration

Rui Wang, Yixue Hao, *Senior Member, IEEE*, Yiming Miao, *Member, IEEE*, Long Hu, and Min Chen, *Fellow, IEEE*

***Abstract*—Recently, the advancements in edge computing have boosted the deployment of video analysis systems based on deep learning, which breaks the limitation of the constrained communication and computing resources of local devices. However, processing multi-scene high-resolution video streams in crowd surveillance remains a significant challenge since it is difficult to formulate dynamic video content and communication environment to support offloading decisions. To bridge the gap between applications and modeling, this paper presents a Real-Time Cloud-edge-device Collaboration framework, which enables fast and accurate Crowd counting (RT3C) on the real dataset. RT3C comprises key frame detection, adaptive patch partition, patch encoder and decoder and computation offloading decision, designed to divide key frames into a minimum number of patches and determine the offloading location of patches. A Real-Time Multi-Agent Actor-Critic (RTMAAC) algorithm based on multi-agent reinforcement learning is proposed to decide whether to compute patches with a lightweight model on edge or a large model on cloud. Unlike traditional approaches ignoring the contents, RTMAAC is a dynamic online decision algorithm based on context of the network and video. Extensive experiments demonstrate that RT3C effectively discriminates the valid frames and optimizes offloading decisions in complex environments, outperforming other baseline algorithms on the two crowd counting datasets. In summary, RT3C provides a promising framework for multi-scene video streams, which can be extended to other applications to realize video computation based on deep models.**

***Index Terms*—Cloud-edge-device collaboration, crowd counting, multi-agent deep reinforcement learning, multi-scene video streams.**

Manuscript received 22 June 2023; revised 16 January 2024; accepted 3 March 2024. Date of publication 18 March 2024; date of current version 8 August 2024. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62176101, Grant 62276109, Grant 62322205, Grant 62272177, Grant 62202410, and Grant 62311530344, in part by the National Key Research and Development Program of China under Grant 2023YFB4503400, and in part by Shenzhen Science and Technology Program under Grant JCYJ20220530143808019. The work of Dr. Miao was supported in part by the Shenzhen Institute of Artificial Intelligence and Robotics for Society (AIRS). *(Corresponding author: Long Hu.)*

Rui Wang is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: ruiwang2020@hust.edu.cn).

Yixue Hao and Long Hu are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and also with the Pazhou Laboratory, Guangzhou 510640, China (e-mail: yixuehao@hust.edu.cn; hulong@hust.edu.cn).

Yiming Miao is with the Shenzhen Institute of Artiffcial Intelligence and Robotics for Society (AIRS), School of Data Science, The Chinese University of Hong Kong, Shenzhen, Guangdong 518172, China (e-mail: yimingmiao@ieee.org).

Min Chen is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China, and also with the Pazhou Laboratory, Guangzhou 510640, China (e-mail: minchen@ieee.org).

Digital Object Identifier 10.1109/TSC.2024.3377156

I. Introduction

DEEP learning has made great progress in computer vision, further promoting the rapid developments of crowd management systems. Intelligent crowd monitoring involves the real-time analysis of ultra-high-definition video streams to realize crowd detection, behavior recognition, tracking, and counting, which have the strict demands of the system delay and computation performance [1].

There are many aspects to improve the computation and response performance of the video analysis system such as making computation on collection devices, optimizing transmission link and computation model, and improving feedback and display of results [2], [3], [4]. For example, there are many redundant frames in a 30FPS video stream, and some works perform frame filtering on the devices to reduce transmission costs based on the video contents and the distinctive features such as edge, pixel, and area [5]. Some works focus on the Region of Interest (RoI) in the image by capturing the target area of the associated frames, and then making computation offloading decisions according to task and content priority with the limited bandwidth resource [6], [7], [8], [9]. Unfortunately, although the target can be located according to context, the newly appearing targets cannot be captured, thereby reducing the detection performance. Besides, considering the differences in inference ability of deep models and target attributes, some works aim to use multi-models to compute the RoI in the frame to execute inference [10], [11]. And there are also some computation offloading researches for computation-intensive tasks, which formulates the offloading decision-making issue in the dynamic network environment by considering minimizing the resource consumption. These models are generally oriented to generalized computation-intensive tasks, which are solved by heuristic algorithms [12], [13] and deep learning methods [14], [15] represented by deep reinforcement learning. However, when these models are applied to specific scenarios, there is still a gap between formulating scenarios and solution schemes. For the typical crowd counting task in monitoring, the researchers pay more attention to designing models with high accuracy and seldom consider the deployment of deep counting model combined with video content and network environment [16], [17]. Once the deep counting

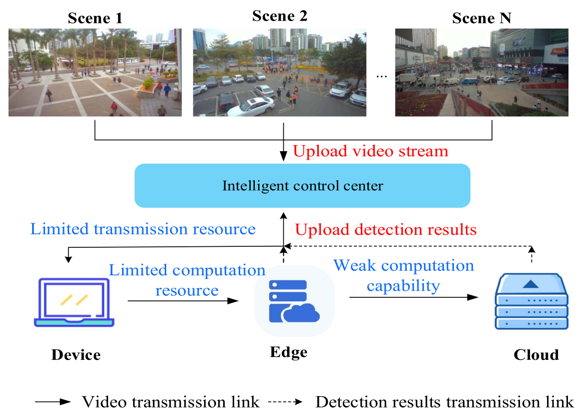

Scene 1
Scene 2
Scene N
Upload video stream
Intelligent control center
Limited transmission resource
Upload detection results
Limited computation resource
Weak computation capability
Device
Edge
Cloud
Video transmission link
Detection results transmission link

Fig. 1. Real-time monitoring system served for multi-scene video streams. The intelligent control center decides the computation offloading locations of the multi-scene video frames based on the context of the devices, edge and cloud.

models are used to execute inference tasks in the real-time video system based on edge computing, they cannot efficiently handle the massive data due to the dynamics of the communication and video frames.

In addition, there are many cameras deployed in the key area to undertake monitoring tasks. The previous researches are mainly designed for single video streams for surveillance, especially in crowd counting and object detection. When multi-scene video streams are required to be monitored, it is more difficult to utilize the resources and context to achieve efficient computation. Based on the network environment and video content, we aim to solve multi-scene crowd computation issues served for video surveillance centers. As shown in Fig. 1, it is a typical architecture for multi-scene video streams in the real-time monitoring system. The high-definition cameras are deployed in multiple scenes and are responsible for collecting and uploading video to the intelligent control center. Due to the weak computation capability and limited transmission resources of local devices, some key frames are uploaded to the edge or cloud to compute with deep models, and non-key frames are computed in the local device. The control center decides the offloading location of video frames based on the awareness of the communication system and cognition of the frame context. The results of device, edge and cloud are aggregated to the control center to realize the frame analysis and presentation. There are few works about modeling multi-scene video streams, especially considering the network environment and video content with limited computation resources and communication resources.

This paper aims to explore a cloud-edge-device collaborative optimization scheme to handle the multi-video streams to achieve real-time crowd monitoring. However, there is a gap between the simulated environment and the real world due to the dynamic communication system and multiple frames. Overall there are the following challenges in the multi-scene real-time crowd monitoring system: (1) The key frame and the RoI of the key frame is not easy to recognize and locate on the local side due to the limited resource of the local device and computation-intensive deep models. (2) The computation offloading location of the key frame is hard to decide due to the dynamics of network environment, video contents, and deep models.

In this paper, we propose a **R**eal-**T**ime **C**rowd counting framework based on the **C**loud-Edge-Device **C**ollaboration (RT3C) to tackle the above-mentioned challenges. First, we propose a key frame detection algorithm that judges whether the frame is a key frame with more changes in the number of people than the previous key frame. Then, to further reduce the consumption of bandwidth resources, we build an adaptive patch partition algorithm to divide the key frame into a minimum number of patches according to the potential crowd distribution, and then use the patch encoding algorithm to encode them. Last but not least, to solve the computation offloading decision problem of key frames of multiple video streams, we propose a Real-Time Multi-Agent Actor-Critic (RTMAAC) based on Multi-Agent Reinforcement Learning (MARL). By combining the context information and network resources of each frame, RTMAAC determines whether the patches of key frames are computed by the lightweight model deployed on the edge or the large model deployed on the cloud. The proposed RT3C provides a solution to realize the offloading decision in the multi-scene video stream processing based on edge computing for the computation-intensive deep learning application beyond crowd counting in the complex communication environment. To the best of our knowledge, it is the first time to establish a parallel processing architecture based on deep models for multi-scene video streams. The contributions of this paper are summarized as follows:

- We propose a real-time video analysis system RT3C via cloud-edge-device collaboration for multi-scene video streams to realize performance-efficient deep computation-intensive tasks.
- We formulate key frame detection, adaptive patch partition, and patch encoding modules to reduce transmission and computation of invalid regions and pixels.
- We design an online computation offloading decision algorithm RTMAAC for patch computation, which takes into account the dynamics of the network environment and the performances of deep models to achieve a trade-off between accuracy and delay.
- We evaluate RT3C on real-world crowd video datasets and experimental results show that our system achieves the best detection accuracy with the constraints of ultra-low latency compared to other baseline methods.

The remainder of this paper is organized as follows: The related work is discussed in Section II. The system overview of RT3C is presented in Section III. The pre-processing for single-scene video stream is illustrated in Section IV. The computation offloading task for multi-scene video streams and the RTMAAC algorithm is depicted in Section V. And the experiments are shown in Section VI. Finally, we conclude our research and prospects the future work in Section VII.

II. Related Work

This section reviews the related work on video stream analysis, the deep models of crowd counting, and the decision-making

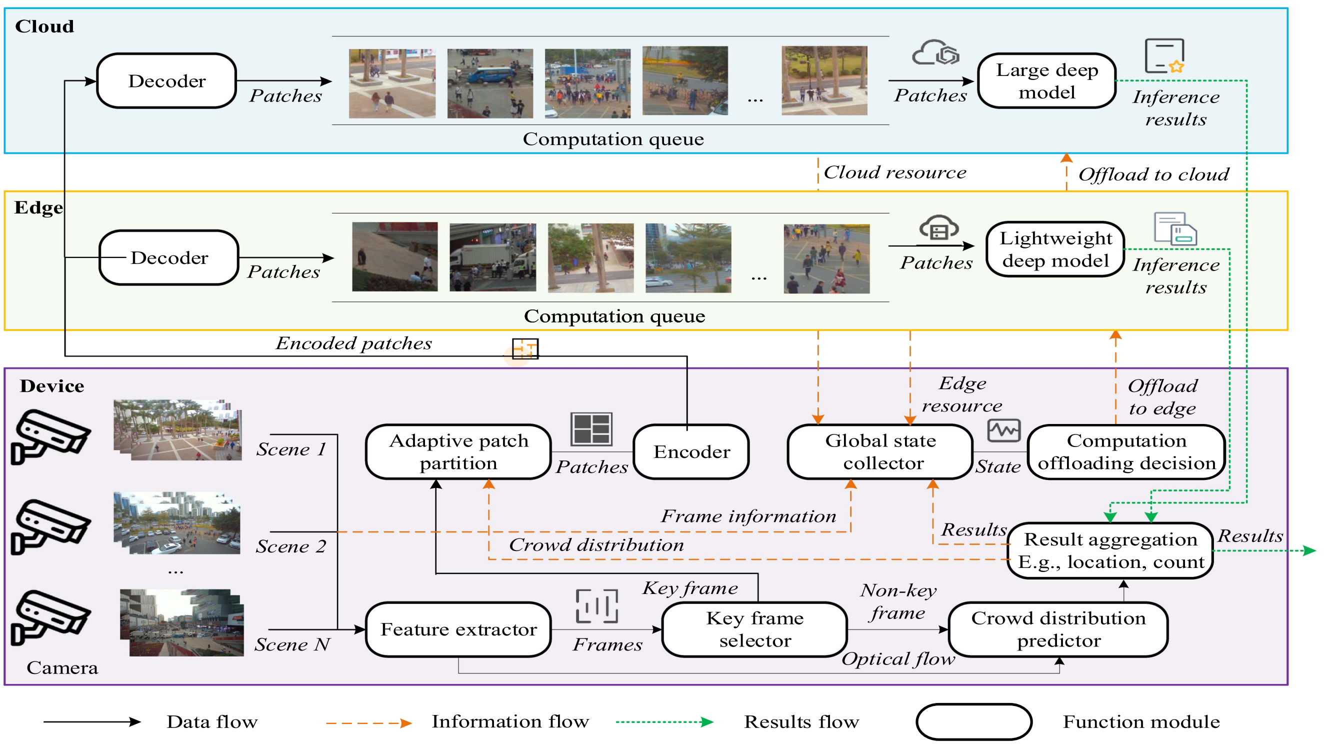


Fig. 2. System overview of RT3C. The four critical modules: (1) Key frame detection including feature extractor, key frame selector and crowd distribution predictor. (2) Adaptive patch partition. (3) Encoder and decoder. (4) Computation offloading decision.

methods especially the reinforcement learning algorithm for computation offloading tasks.

A. Real-Time Video Analysis

Many methods have been proposed to meet the demands of latency and performance in the real-time video analysis systems [11], [18], [19], [20], [21], [22]. ReMIX builds a flexible region division framework for 4K video target detection to improve accuracy and reduce the delay [8]. EdgeDuet realizes an automatic object detection system with RoI encoding by a local detector and remote deep model [10]. Ekya presents an elastic resource scheduler and provides a resource allocation scheme on edge servers [23]. Reducto filters the frame on the local device using low-level features [5]. VaBUS is a video analysis framework based on edge-cloud collaboration by understanding the foreground and background of the frame and performing adaptive RoI encoding [7]. These systems are all designed to reduce computation delay and improve accuracy while they cannot combine system environment and video context to make comprehensive computation decisions. RT3C is a cloud-edge-device collaboration framework by considering the communication and computation environment to realize the trade-off between delay and accuracy for multi-scene video streams.

B. Deep Model for Crowd Counting and Detection

There are many deep models designed for crowd behavior analysis, such as detection, tracking, and counting with multiple deployment requirements [25], [26], [27]. The Yolo algorithm is an end-to-end model, which realizes end-to-end target detection and classification to satisfy the real-time requirements to process video stream [28], [29], [30]. In terms of crowd tracking, in [24], a head detector, the Headhunter framework is proposed to track small heads in congested scenes. In [31], a crowd flow estimator is formulated based on optimal transmission distance to make real-time estimation of crowd count. In terms of crowd counting in video streams, spatiotemporal information is used to track and count crowds in consecutive frames [32], [33]. In addition, in [34], the fusion feature of spatial information extracted by a 2D network and temporal information extracted by a 1D network is used to estimate the number of people in video streams. In this paper, due to the demands of real-time detection, we use the Yolov7-based lightweight model and large model [35], which are deployed on the edge and cloud to detect and count people in the scenes.

C. Computation Offloading

The computation offloading task is to decide the offloading location and the number of offloading tasks in the heterogeneous network environment for the computation-intensive tasks [36], [37], [38], [39], [40], [41]. Aiming for the uncertainty of edge nodes and the dynamics of computation tasks, a distributed algorithm based on model-free deep reinforcement learning is proposed to generate offloading decisions for each device [42]. In [43], an actor-critic framework-based offloading scheme for intelligent computation is proposed, which is applied to Internet of Things applications. In [44], multidimensional resource management for UAV-assisted vehicle networks is studied, using Multi-agent Deep Deterministic Policy Gradient (MADDPG) to

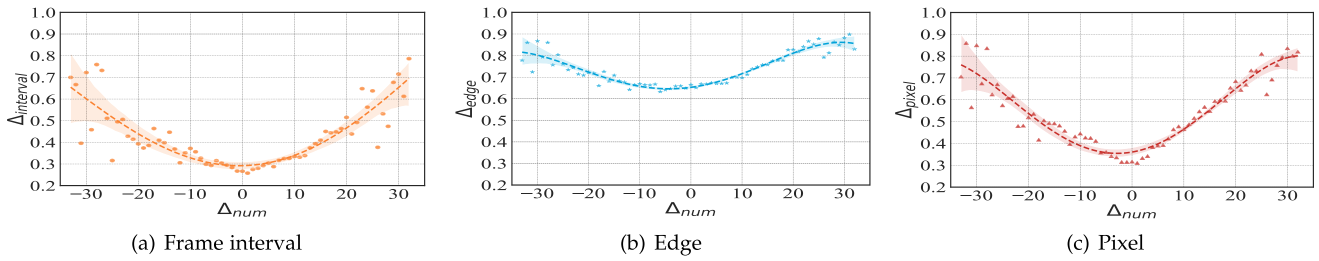


Fig. 3. Fitting curve of change of crowd count Δ_{num} and basic features including $\Delta_{interval}$, Δ_{edge} and Δ_{pixel}. The curves are fitted on the CroHD dataset [24] which can be used for key frame detection according to the estimated count change.

make resource allocations for offloading tasks to meet heterogeneous requirements. In [14], a multi-task learning approach based on neural network is proposed to jointly solve the computation offloading decision and computation resource allocation in multi-access edge computing. In [15], a multi-agent online learning algorithm is proposed to solve the optimal resource allocation problem. Most of the researches simulated the offloading task as a unified expression without considering the specific task content, quantity and accuracy, which have a great influence on offloading decision. In this paper, we adopt an online MARL algorithm to jointly optimize the crowd detection accuracy and processing delay under ultra-low latency based on edge computing.

III. RT3C: System Overview

The overview of RT3C is shown in Fig. 2. The device receives frames from fixed high-definition cameras in multiple scenes, the feature extractor extracts the basic features of the image, and the key frame selector determines whether the frame is a key frame. If it is a non-key frame, the crowd distribution predictor estimates the location and the number of people in the current frame according to the previous key frame. If it is a key frame, the adaptive patch partition module divides the potential crowd area into a minimum number of patches with a fixed partition size. The global state collector aggregates the network environment and state information of the frames, and hand them over to the computation offloading decision module to determine the location where the patches of the key frame are computed. The patches of key frames need to be encoded by the encoder in the local device and decoded by the decoder on the edge or cloud. Lightweight and large deep models are deployed on edge and cloud respectively, and their inference abilities are different. The patches enter into the computation queue step by step, which are computed by the deep model once time to obtain the results. Once the detection results of patches are obtained, the purification operation is performed through the aggregation module to eliminate the influence of overlapping areas among the patches. When frames at a time slot in multiple scenes are complete, the next frames are processed by the above steps. In the next two sections, we discuss the details of each module taking the crowd counting application as an example. By the way, RT3C can be applied to other computation-intensive tasks beyond crowd counting, such as target tracking, target detection, etc.

IV. Pre-Processing for Single-Scene Video Stream

In this section, we illustrate the critical modules including key frame detection, adaptive patch partition and patch encoding in RT3C for each single-scene stream.

A. *Key Frame Detection*

The key frame is defined according to the change of crowd motion flow in the scene. If there is much change in the crowd count between the current and the previous frame, it is a key frame. Otherwise, it is a non-key frame. For the first frame, it is set as a key frame required to be computed with the deep model. There are three modules in the key frame detection, described in the following.

Feature extractor: The basic frame features are frame interval, edge and pixel. The frame interval refers to the number of the frame sequence. The edge refers to the most obvious part of the local intensity in the image. And the pixel refers to the pixel value of the grayscale image.

Key frame selector: The labeled dataset is used to obtain features and their corresponding count difference Δ_{num}. After making statistical analysis and experiments, the four-term polynomial fits the changes of crowd count and basic features. The difference of features is $\Delta_{feature}$, and the change of the number of people is Δ_{num}, then the relationship between $\Delta_{feature}$ and Δ_{num} is:

$$\Delta_{feature} = \omega_0 + \omega_1\Delta_{num} + \omega_2\Delta_{num}^2 + \omega_3\Delta_{num}^3 + \omega_4\Delta_{num}^4, \quad (1)$$

where ω_0, ω_1, ω_2, ω_3 and ω_4 are the fitting parameters of the polynomial, which are trained on the labeled dataset. And the $\Delta_{feature}$ includes the $\Delta_{interval}$, Δ_{edge} and Δ_{pixel}. Fig. 3 shows the training points and the four-term polynomial fitting curve of each feature on the CroHD dataset [24]. The difference in the number of people between frames is the difference in the feature value between the frames using the Equation (1). $\bar{\Delta}_{num}$ is the average count difference of the three features. When it is greater than the count threshold ϵ, the current frame is the key frame, otherwise it is a non-key frame, expressed as:

$$\kappa = \begin{cases} 1, & \bar{\Delta}_{num} > \epsilon, \\ 0, & \text{otherwise}, \end{cases} \quad (2)$$

where the threshold $\epsilon \sim \hat{Y}_{num}$, $\hat{Y}_{num}$ is the estimated number of people in the scene.

Algorithm 1: Adaptive Patch Partition.

```
1  function Adaptive_partition($locs$, $Img$, $\rho_1$, $\rho_2$, $MinPts_1$, $MinPts_2$, $\Lambda$)
   Input:
     • $locs$: The coordinates of the face location.
     • $Img$: The image required to make a partition.
     • $\rho_1$ and $\rho_2$: The neighborhood radiuses of DBSCAN.
     • $MinPts_1$ and $MinPts_2$: Thresholds of the number of objects near $\rho_1$ and $\rho_2$.
     • $\Lambda$: The patch resolution size of the partition.
   Output:
     • $\Theta$: The partition lines over the whole image.
   // $\Omega_0$ is the abnormal point set and the $\Omega_1$ is the clustered point sets.
2  $\Omega_0, \Omega_1 \leftarrow$ DBSCAN($locs$, $\rho_1$, $MinPts_1$);
   // $\Omega_0$ is the abnormal point set and the $\Omega_2$ is the clustered point sets.
3  $\Omega_0, \Omega_2 \leftarrow$ DBSCAN($\Omega_0$, $\rho_2$, $MinPts_2$);
4  $\Omega \leftarrow$ calculate the union of $\Omega_1$ and $\Omega_2$;
5  $\Gamma \leftarrow$ detect boundary lines of each cluster of $Img$;
6  Initialize the empty borderline set $bsts$;
7  for each borderline $l$ in $\Gamma$ do
8      $l' \leftarrow$ pad the region $l$ of the $Img$ based on $\Lambda$;
9      $l'_s \leftarrow$ make partition for the region $l'$ based on $\Lambda$;
10     for each borderline $bst$ in $l'_s$ do
11         if $bst$ not exists in $bsts$ then
12             Add the $bst$ to the $bsts$;
13         end
14     end
15     $bsts \leftarrow$ unify the region in the $bsts$;
16 end
17 for each $bst$ in $bsts$ do
18     $bst \leftarrow$ make partition according to $\Lambda$;
19     for each point $pt$ in $locs$ do
20         if $pt$ exists in $bst$ then
21             Add $bst$ to $\Theta$;
22         end
23     end
24 end
25 Return $\Theta$.
```

Crowd distribution predictor: The crowd distribution of the current frame is estimated based on the previous results of the key frame and the optical flow map [45]. First, a sparse optical flow map of two consecutive frames is computed. If the pixel where the face is located has moved much compared to the key frame, the new face position is calculated based on the optical flow map. Otherwise, the current position is equal to the face position in the previous key frame. In this way, the crowd distribution is estimated for non-key frames. When a frame is judged as a non-key frame, its crowd position and count are estimated locally based on historical information.

B. Adaptive Patch Partition

We propose an adaptive patch partition method based on the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [46], and the overall process is shown in Fig. 4. First, in Fig. 4(a), the estimated crowd position is obtained based on the crowd distribution predictor. It is defined that the center point of the face is used to represent the person's location. Then, we select the potential crowd distribution area according to Algorithm 1. First, all locations are clustered using the DBSCAN algorithm. And then all abnormal points are clustered into one cluster, and the result is shown in Fig. 4(b). For the outliers, the DBSCAN algorithm is used to cluster them again, and the outliers are discarded in this round directly. Meanwhile, the clusters are merged where the normal points in the two rounds of DBSCAN are located in Line 4. In this way, the abnormal crowds are clustered into the normal crowd as much as possible. For the remaining small number of outliers, it is believed that much more transmission and computation cost is required but they have less information, and discarding them has little effect on the overall detection accuracy. This conclusion is validated in the Section VI-B. Next, the boundary lines Γ of the area where the points in each cluster are located are calculated. Then, in Lines 7–16, the first round of partition is performed on these regions. In Line 8, the surrounding area is filled based on the fixed patch size. The filling rule is to divide the image size by the patch size, and the boundary area with the larger remaining region is filled. Afterward, we merge the cropped patches and remove and unify the boundary lines of the patches with inclusion and overlapping regions. After completing the first round of region mergence, in Lines 17–24, the second round of partition is performed. In Line 18, the patches are divided in an overlapping manner again based on the estimated potential crowd distribution area. If there may exist crowds in the patch, the boundary line of the patch is added to the final patch set. Finally, the adaptive patch partition of the RoI is completed for the key frame with the minimum number of patches.

C. Patch Encoder and Decoder

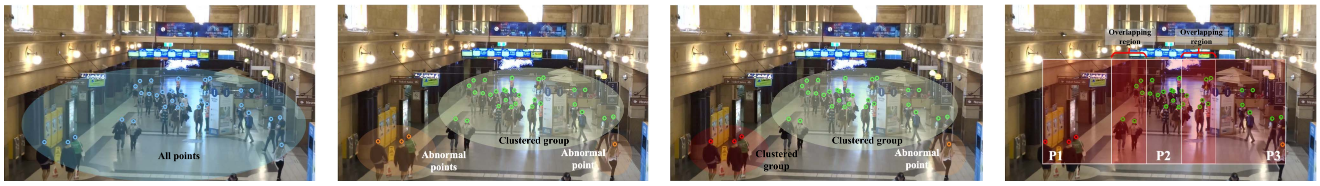
To further reduce the transmission cost, JPEG is used to encode each patch [47]. JPEG compresses the high-frequency information of the image and preserves the color information. Although the quality of the image decreases, compressing the image reduces the transmission cost without affecting the computation of the deep model. The encoder module is realized on the local, and the decoder module is executed on the edge and cloud.

V. Computation Offloading Decision for Multi-Scene Video Streams Based on RTMAAC

In this section, the patch computation problem is modeled as the computation offloading via cloud-edge-device collaboration. We present the system model and the RTMAAC algorithm based on MARL to realize online computation offloading decisions. Table I shows the symbols.

A. System Model

In the RT3C, the local device receives video streams from N scenes. The patches of key frames are transmitted to the edge or cloud to be computed with the deep model. For non-key frames, they are estimated on the local. All results are aggregated locally



(a) Original frame (b) DBSCAN for all points (c) DBSCAN for abnormal points (d) Adaptive partition

Fig. 4. Pipeline of adaptive patch partition. The original frame is processed by DBSCAN and the abnormal points are processed by DBSCAN again. And then the adaptive partition is conducted on the RoI to obtain the patches such as P_1, P_2 and P_3.

TABLE I
LIST OF NOTATIONS

Symbol	Description
N	Number of all scenes
K	Number of patches of each frame
t	Time slot in one episode of multi-frames decision
π	Offloading strategy in $\{0, 1\}$
d_1	Distance from device to edge cloud
d_2	Distance from device to cloud
$\mathcal{N}$	Number of frames in each video clip
W	Transmission bandwidth from local to edge
p	Transmission power for each scene
Λ	Patch resolution size to make partition
Ψ^{Emodel}, Ψ^{Cmodel}	The deep model deployed on edge and cloud
$T_{i,j}^{Etrans}$, $T_{i,j}^{Ctrans}$	Transmission time of the j_{th} frame of the i_{th} scene from local device to edge and cloud
$T_{i,j}^{Lcom}$	Computation time of the j_{th} frame of the i_{th} scene on device
$T_{i,j,k}^{Ecom}$, $T_{i,j,k}^{Ccom}$	Computation time of k_{th} patch of the j_{th} frame of i_{th} scene on edge and cloud
$A_{i,j,k}^{Eap}$, $A_{i,j,k}^{Cap}$	AP of the k_{th} patch of the j_{th} frame of the i_{th} scene on edge and cloud
$A_{i,j,k}^{Emae}$, $A_{i,j,k}^{Cmae}$	MAE of the k_{th} patch of the j_{th} frame of the i_{th} scene on edge and cloud
$A_{i,j}^{Lap}$, $A_{i,j}^{Lmae}$	AP and MAE of the j_{th} frame of the i_{th} scene on device
ϕ, ϕ'	Parameters of critic network and target network
θ, θ'	Parameters of actor network and target network
τ	The soft update parameter for the two target networks
γ	Discount factor in [0,1] to update RTMAAC

and output for the crowd monitoring center. The communication model, inference model and evaluation model are formulated.

Communication model: For a non-key frame, it is processed on the local without an extra transmission. For the patches of key frame, they are transmitted to the edge or cloud through the wireless network and optical fiber network. It is assumed that each video stream occupies a separate transmission channel, and its bandwidth is constant during transmission. The upload rate from the device to the edge is [48]:

$$r_1 = W \log_2 \left(1 + \frac{p}{\sigma^2 \cdot d_1^{\iota}}\right), \tag{3}$$

where σ^2 is the Gaussian noise, and ι is the distance loss parameter of the power. The upload rate from the edge to the cloud is r_2. Therefore, for frame j in scene i, the transmission delay of the k_{th} patch from local to edge is:

$$T_{i,j,k}^{Etrans} = \frac{z_{i,j,k}}{r_1}, \tag{4}$$

where $z_{i,j,k}$ is the image size of the k_{th} patch of frame j in scene i. If the patch is offloaded to the cloud, the transmission delay from the local to the cloud is:

$$T_{i,j,k}^{Ctrans} = T_{i,j,k}^{Etrans} + \frac{d_2 - d_1}{r_2}. \tag{5}$$

DNN inference model: A lightweight model Ψ^{Emodel} is deployed on the edge and a large model Ψ^{Cmodel} is deployed on the cloud. The computation time is obtained based on the FLOPs of deep models and the performance of the running machine. $T_{i,j,k}^{Ecom} = \mathcal{F}(\delta^e, f^e, z_{i,j,k})$ and $T_{i,j,k}^{Ccom} = \mathcal{F}(\delta^c, f^c, z_{i,j,k})$ are two computation delays. $\mathcal{F}$ represents the computation delay function in the real system, the δ^c and δ^e are the FLOPs of the deep models, and f^c and f^e represent the computation performance of the edge and cloud. $T_{i,j,k}^{Ecom}$ and $T_{i,j,k}^{Ccom}$ are calculated by averaging the total inference time by performing inference on publicly available datasets with the deep model. Therefore, the delay is the sum of the communication delay and the computation delay for the k_{th} patch of frame j of scene i:

$$T_{i,j}(\pi) = \sum_{k=0}^{K_{i,j}} T_{i,j,k}(\pi) = \sum_{k=0}^{K_{i,j}} ((1 - \pi_{i,j,k}) \cdot (T_{i,j,k}^{Etrans} + T_{i,j,k}^{Ecom}) + \pi_{i,j,k} \cdot (T_{i,j,k}^{Ctrans} + T_{i,j,k}^{Ccom})), \tag{6}$$

where $\pi \in \{0, 1\}$ is the offloading decision. When $\pi = 0$, the k_{th} patch of frame j of scene i is offloaded to the edge. When $\pi = 1$, it is offloaded to the cloud. The total latency of frame j in scene i is the sum of communication delays and the computation delays of $K_{i,j}$ patches.

Accuracy evaluation model: Average precision (AP) is used to evaluate the detection accuracy of the face position in the patch. After all the patches in the frame are computed by the DNN model Ψ^{Emodel} and Ψ^{Cmodel} deployed on the edge and the cloud, the detection results are sent back to the local to make aggregation. Non-maximum suppression is used to deal with the problem of multiple estimated boxes for the same object in overlapping regions to get the only detection result for a target [49]. The network structure and the number of parameters decide the inference ability of Ψ^{Emodel} and Ψ^{Cmodel}. We adopt the AP and MAE to measure the inference performance of these two models. The AP of k_{th} patch of the frame j in the scene i is:

$$A_{i,j,k}^{ap} = \int_0^1 p_{i,j,k}(r) dr, \tag{7}$$

where $p_{i,j,k}(r)$ is the detection precision and recall curve of the patch. The $A_{i,j,k}^{Eap}$ and $A_{i,j,k}^{Cap}$ are APs of k_{th} patch of the frame j in the scene i, which are calculated by lightweight model on the edge and large deep model on the cloud. Therefore, the AP for the k_{th} patch of the frame j in the scene i based on offloading decision π is:

$$\begin{aligned} A_{i,j}^{ap}(\pi) &= \frac{1}{K_{i,j}} \sum_{k=0}^{K_{i,j}} A_{i,j,k}^{ap}(\pi) \\ &= \frac{1}{K_{i,j}} \sum_{k=0}^{K_{i,j}} ((1-\pi_{i,j,k}) \cdot A_{i,j,k}^{Eap} + \pi_{i,j,k} \cdot A_{i,j,k}^{Cap}). \end{aligned} \tag{8}$$

Besides, the number of people in the scene through the detection result is estimated, so the Mean Absolute Error (MAE) showing the count accuracy of k_{th} patches:

$$A_{i,j,k}^{mae} = \mid \hat{y}_{i,j,k} - y_{i,j,k} \mid, \tag{9}$$

where $\hat{y}_{i,j,k}$ and $y_{i,j,k}$ are the predicted value and true value of the crowd count in the patch. The $A_{i,j,k}^{Emae}$ and $A_{i,j,k}^{Cmae}$ are MAEs of k_{th} patch of the frame j in the scene i, which are calculated by lightweight model and large model. Therefore, the MAE based on offloading decision π is:

$$\begin{aligned} A_{i,j}^{mae}(\pi) &= \sum_{k=0}^{K_{i,j}} A_{i,j,k}^{mae}(\pi) \\ &= \sum_{k=0}^{K_{i,j}} ((1-\pi_{i,j,k}) \cdot A_{i,j,k}^{Emae} + \pi_{i,j,k} \cdot A_{i,j,k}^{Cmae}). \end{aligned} \tag{10}$$

By combining the $A_{i,j,k}^{ap}$ and $A_{i,j,k}^{mae}$, the comprehensive accuracy evaluation for the the DNN model Ψ^{Emodel} and Ψ^{Cmodel} is obtained.

B. Problem Formulation

Based on the above models, we formulate the overall optimization problem, aiming to maximize detection and count accuracy and crowd estimation in multiple scenarios while minimizing system delay in the RT3C.

The data generated by scenes at a time slot is transmitted to the intelligent control center locally to perform computation. The optimization goal is to give an appropriate offloading strategy π for patches of key frames in N scenes while minimizing the delay and maximizing the accuracy:

$$\begin{aligned} &\min_{\pi} \sum_{i=1}^{N} (\lambda_1 T_{i,j}(\pi) + \lambda_2 A_{i,j}^{ap}(\pi) + \lambda_3 A_{i,j}^{mae}(\pi)), \\ &\text{s.t. } \pi \in \{0,1\}, \\ &\sum_{i=1}^{N} \sum_{k=1}^{K_{i,j}} (1-\pi_{i,j,k}) \le \mathcal{M}_{\max}^{edge}, \\ &\sum_{i=1}^{N} \sum_{k=1}^{K_{i,j}} \pi_{i,j,k} \le \mathcal{M}_{\max}^{cloud}, \end{aligned} \tag{11}$$

where $\mathcal{M}_{\max}^{edge}$ and $\mathcal{M}_{\max}^{cloud}$ are the maximum number of patches that can be offloaded to edge and cloud. λ_1, λ_2 and λ_3 are weight parameters. There are some challenges to solving the joint optimization problem: (1) Due to the differences in crowd distribution and background in multiple scene frames, it is difficult to make a common offloading decision without considering the frame information. (2) The offloading location of each patch affects patch offloading in other scenarios and subsequent patches due to limited computation resources. (3) The real-time monitoring system requires a balance between delay and accuracy to achieve optimization with quality of experience.

C. MDP Formulation

Deep reinforcement learning is a useful method to solve optimization problems in dynamic environments. The problem is defined as a Markov decision process, expressed as $(\mathcal{S}; \mathcal{A}; P; R)$. $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is a state-transition function expressed as $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \leftarrow R$, R is the reward function. The RT3C is regarded as the interaction environment, and the video streams are the target subjects monitored by the agents. The multi-agents make the offloading decisions for the patches of the multi-scene frames based on interaction and collaboration in the RT3C. Action triggers the state transition, meanwhile affecting the subsequent offloading decision. During the process, agents receive a reward to evaluate actions. According to the optimization goal of RT3C, it can be transformed as the Markov decision process and the state, action and reward can be defined as follows.

State: For the state s_t in the $\mathcal{S}$, t represents the time slot, whose maximum value is the number of patches in these scenes. s_t contains ten values: whether the frame to which the patch belongs is a key frame, expressed as $\kappa_t \in \{0,1\}$, if it is a key frame, then $\kappa_t = 1$ otherwise $\kappa_t = 0$; whether all patches in the frame have been offloaded is expressed as $l_t \in \{0,1\}$; the number of heads in the region in the previous frame is h_t; the index of the current patch in all patches of the frame is k; the number of patches offloaded to the edge and cloud in the frame before t is n_t^E and n_t^C; the computation delays for a patch on edge and cloud are T_t^{Ecom} and T_t^{Ccom}; the transmission delays for a patch from local to edge and cloud are T_t^{Etrans} and T_t^{Ctrans}. Therefore, the state of the agent at time slot t is:

$$s_t = \{\kappa_t, l_t, h_t, k, n_t^E, n_t^C, T_t^{Etrans}, T_t^{Ecom}, T_t^{Ctrans}, T_t^{Ccom}\}. \tag{12}$$

After the system makes an offloading decision for all patches at t, the system enters the next state. The state is not terminated until the patches of all the frames at the same time are completed. When the next frames arrive, the system state is initialized and a new episode state transition begins.

Action: The number of patches in each frame is different due to the differences of crowd distribution. At time slot t, the agent makes an computation offloading decision $a_{i,j}(t)$ of the patch of frame j in the i_{th} scene. And $a_{i,j} \in \{0,1\}$ means the edge and cloud. The offloading strategy in the current state is noted as the π.

Reward: In a multi-scene crowd counting system, the optimization goal is to obtain the accurate location of faces and the number of crowds in the scene with less latency. In MARL, the reward is strongly related to the convergence of the model and the optimization target. The optimization-related reward function is described in detail.

- *Delay-constrained reward:* If the frame is a non-key frame, there is only computation delay. If it is a key frame, its delay includes computation delay and transmission delay. To meet the delay-sensitive requirements, for each agent i, the reward of the k_{th} patch in the j_{th} frame is:

$$r_{i,j,k}^{delay} = (1-\kappa_{i,j}) \cdot (-T_{i,j,k}^{Lcom}) + \kappa_{i,j} \cdot (-T_{i,j,k}(\pi)) \tag{13}$$

At t, if the $t \leq K_{i,j}$, $r_{i,j,t}^{delay} = r_{i,j,k}^{delay}$, otherwise the sum of transmission delay and computation delay is 0 and $r_{i,j,t}^{delay} = 0$. For non-key frames, the number of patches $K_{i,j}$ is set to 1, and the frame is computed on the local side. The local computation delay $T_{i,j,k}^{Lcom}$ is measured on the public dataset. Therefore, the total delay reward at t in the scene is:

$$R_{j,t}^{delay} = \sum_{i=0}^{N} r_{i,j,t}^{delay}. \tag{14}$$

- *AP-constrained reward:* For detection accuracy, after receiving the detection results of all frames, the AP reward of the k_{th} patch and the total reward at t in the scene is:

$$r_{i,j,k}^{ap} = (1-\kappa_{i,j}) \cdot A_{i,j,k}^{Lap} + \kappa_{i,j} \cdot A_{i,j,k}^{ap}(\pi), \tag{15}$$

$$R_{j,t}^{ap} = \sum_{i=0}^{N} (r_{i,j,t}^{ap} - 1), \tag{16}$$

where $A_{i,j,k}^{Lap}$ is AP of patch processed on local. At time slot t, if the $t \leq K_{i,j}$, $A_{i,j,t}^{ap} = A_{i,j,k}^{ap}$, otherwise $A_{i,j,t}^{ap} = 1$ to eliminate the interference of the complete frame. Since the larger AP represents better performance, it is transformed from a maximization problem to a minimization problem to realize joint optimization.

- *MAE-constrained reward:* For counting accuracy, the MAE reward of the k_{th} patch and the total reward at t in the scene is:

$$r_{i,j,k}^{mae} = (1-\kappa_{i,j}) \cdot (-A_{i,j,k}^{Lmae}) + \kappa_{i,j} \cdot (-A_{i,j,k}^{mae}(\pi)), \tag{17}$$

$$R_{j,t}^{mae} = \sum_{i=0}^{N} r_{i,j,t}^{mae}, \tag{18}$$

where the $A_{i,j,k}^{Lmae}$ is the MAE of the patch processed on local side. If the $t \leq K_{i,j}$, $A_{i,j,t}^{mae} = A_{i,j,k}^{mae}$, otherwise $A_{i,j,t}^{mae} = 0$. Therefore, to make a computation offloading decision by considering delay, detection and counting accuracy, the total reward is:

$$R_{j,t}^{total} = \lambda_1 R_{j,t}^{delay} + \lambda_2 R_{j,t}^{ap} + \lambda_3 R_{j,t}^{mae}, \tag{19}$$

where λ_1, λ_2 and λ_3 are weight parameters, λ_1 and λ_3 are negative, and λ_2 are positive. For the j_{th} frame, the cumulative reward is $R_j^{total} = \sum_{t=0}^{\max(K_{i,j})} R_{j,t}^{total}$. We adopt an actor-critic MARL method to solve the problem. According to the Soft Policy Iteration in [51], the repeated application of soft policy evaluation and soft policy improvement to any π converges to the optimal policy π^* for all π and $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, assuming $|\mathcal{A}| < \infty$. Therefore, this problem can be solved with an actor-critic MARL algorithm to maximize the cumulative reward $R_{j,t}^{total}$.

D. RTMAAC Design

We propose an RTMAAC algorithm [50] based on MARL to perform computation offloading, as shown in Fig. 5. The actor networks θ make offloading decisions and the critic network ϕ evaluates the action. First, the agent of each scene perceives the state of the current patch and inputs them into the multi-agent network, and each agent obtains the offloading location of the corresponding patch at the current moment. These actions are executed in the real-time monitoring system to obtain a reward. Then the system steps into the next state. The current state, action, reward and next state are sent to the experience replay buffer. Meanwhile, the state-action is input to the critic network for evaluation, which uses a multi-head attention mechanism to realize the encoding of the state and action. The Q value is obtained to optimize the critic network with Mean Square Error (MSE). The RTMAAC algorithm is iteratively trained with a gradient-based updating manner with action value and Q value in the multi-scene video streams until the model converges. After completing the training step, it executes online inference with actor networks. The details are elaborated in the next.

Multi-agent network: Each agent makes the computation offloading decision according to the current environment and the patch state. The actor network includes a multi-layer perception (MLP), receiving the state vector with a length of 10 and outputting the offloading location vector with a length of 2. Meanwhile, the computation results are put into the experience replay buffer together with the next state. When the samples are accumulated enough in the initial period, the policy network is trained based on:

$$\nabla_{\theta_i} J(\phi) = \mathbb{E}_{s,\, a\sim D}[\nabla_{\theta_i} \phi(a_i(t)|s_i(t)) \cdot \nabla Q^{\phi}(s_i(t), a_i(t))|_{a_i(t)=\phi(s_i(t))}], \tag{20}$$

where D is the historical information in the experience replay buffer, including (s_t, a_t, r_t, s_{t+1}). The update strategy of the actor network θ_i is:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}}[(Q^{\phi}(s_i(t), a_i(t)) - y)^2], \tag{21}$$

where $y = r_i + \gamma Q^{\phi'}(s_i(t+1), a_i(t+1)|_{a_i(t+1)=\phi(s_i(t))})$ and r_i is the reward value of the i_{th} scene. The actor network is updated with the Q value evaluated by the critic network for the action $a_i(t)$. The target network is to improve the stability in the process of model training. And the soft update is used to improve the target network as $\theta' = \tau\theta + (1-\tau)\theta'$.

Multi-head attention mechanism: In MARL, the state information of each agent is shared for mutual cooperative perception and decision-making. To overcome the high-dimensional state vectors and realize information sharing among agents, the RTMAAC effectively encodes state vectors and learn the relation between agents based on the attention mechanism. The RTMAAC algorithm obtains the fixed length attention vector avoiding the problem of the increase of the critic network with

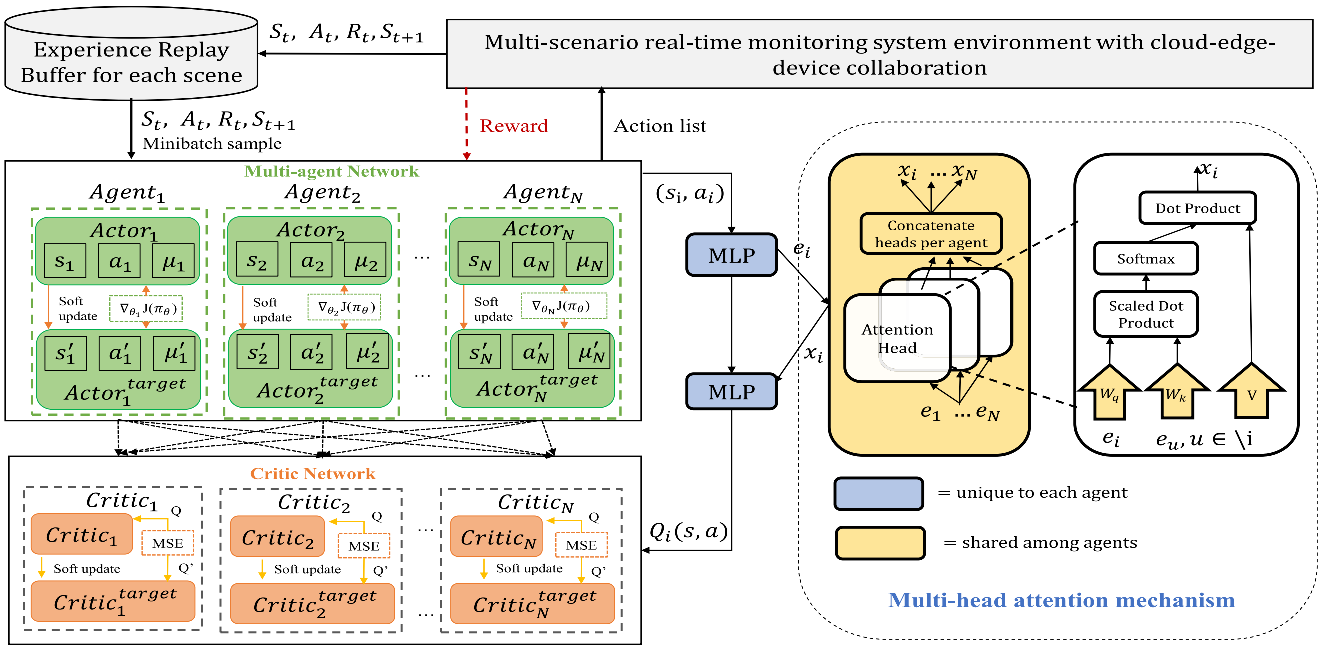


Fig. 5. Overview of the RTMAAC. The left is the multi-agent network and the critic network and the multi-agents interact with the environment to get a reward and the action. The right is the multi-head attention mechanism to get the Q value to evaluate the actor [50].

increasing actors. For agent i, we can obtain Q value by relating with other agents:

$$Q_i^{\phi}(s,a) = f_i(\Phi_i(s_i(t), a_i(t)), w_i), \tag{22}$$

where f_i is a two-layer MLP, and Φ_i is a one-layer MLP embedding function. w_i is the sum of the weights of agent i and other agents, expressed as:

$$w_i = \sum_{\hat{i} \neq i} \alpha_{\hat{i}} \cdot v_{\hat{i}} = \sum_{\hat{i} \neq i} \alpha_{\hat{i}} h(V\Phi_{\hat{i}}(s_{\hat{i}}, a_{\hat{i}})), \tag{23}$$

where $v_{\hat{i}}$ is the embedding equation of the agent, which is encoded by an embedding function and linearly transformed by a shared matrix V, h is an element-wise leaky ReLU function. $\alpha_{\hat{i}}$ is the attention weight mapped by the query-key pair, which is obtained by calculating the similarity of the embedded vector:

$$\alpha_{\hat{i}} \propto \exp(e_u^T W_k^T W_q e_i), \tag{24}$$

where $e_i = \Phi_i(s_i, a_i)$, W_q is the query of e_i, and W_k is the key of e_u. Each head has independent parameters (W_k, W_q, V). By concatenating the contributions of all heads into one vector, the weight combinations of different agents for each head are obtained. In RT3C, multi-scene-oriented agents make more comprehensive decisions by paying attention to the states and actions of agents in other scenes.

Critic network: The update of the critic network is to minimize the regression loss of collaboration, expressed as:

$$\mathcal{L}_Q(\phi) = \sum_{i=1}^{M} \mathbb{E}_{(s,a,r,s')\sim D}[(Q_i^{\phi}(s,a) - y_i)^2], \tag{25}$$

$$y_i = r_i + \gamma \mathbb{E}_{a' \sim \pi_{\theta'}(s')}[Q_i^{\phi'}(s', a') - \beta \log(\pi_{\theta'_i}(a'_i(t) \mid s'_i(t)))], \tag{26}$$

where ϕ' and θ' are the parameters of the target critic network and the target actor network. Q_i^{ϕ} is the estimated action evaluation of agent i, and β is the temperature parameter used to balance entropy and reward. M is the number of samples. The policy network is updated by:

$$\begin{aligned} \nabla_{\theta_i} J(\phi_\theta) &= \mathbb{E}_{s\sim D, a\sim\pi}[\nabla_{\theta_i} \log(\pi_{\theta_i}(a_i|s_i)) \\ &(-\beta \log(\pi_{\theta_i}(a_i \mid s_i)) + Q_i^{\phi}(S,A) - b(S, a_{\hat{i}}))], \end{aligned} \tag{27}$$

where $b(S, a_{\hat{i}})$ is the advantage function, expressed as:

$$b(S, a_{\hat{i}}) = \mathbb{E}_{a_i \sim \pi_i(s_i)}\left[Q_i^{\phi}(s, (a_i, a_{\hat{i}}))\right]. \tag{28}$$

For the update of the target critic, the soft update is $\phi' = \tau\phi + (1-\tau)\phi'$. For the complexity of the RTMAAC algorithm. Let $\omega_a, \omega_{edge}, \omega_{cloud}$ denote the weight counts of the actor network, the lightweight model on edge and the large model on cloud. In the inference stage, the computation complexity of the local, edge and cloud are denoted as $\mathcal{O}(n)$, $\mathcal{O}(\omega_{edge})$ and $\mathcal{O}(\omega_{cloud})$. The $\mathcal{O}(n)$ is the basic linear operation, which is completed in a linear number of operations. Taking the real model used in this paper as an example, the number of parameters and FLOPs of actors are 5.06K and 4.93K. The deep models on the edge and cloud are 5.77M and 13.3G, and 34.89M and 104.1G. It is concluded that the decision complexity and the computation complexity with the counting models during the inference are low to meet the requirements of the real applications.

VI. Experimental Results

In this section, the experimental simulation platform and the parameters and model settings are presented. We compare the proposed pre-processing algorithms and computation offloading algorithms in the RT3C with other baseline methods. And some sensitivity testings are performed to demonstrate the effect of the proposed modules.

A. Experimental Setting

Simulation settings: An server with an NVIDIA Telsa V100 32GB GPU and four Intel(R) Xeon(R) Gold 6252 CPUs of 2.10 GHz is used as simulation platform. The simulation system is Ubuntu 18.04.3 LTS, on which local node, edge node and cloud node are deployed. In the experiments, we use Python 3.6.8 to establish the RT3C and Pytorch 1.7.1 as the framework of the detection model and RTMAAC.

Experimental parameters: The 2, 3, and 4 video scenes with the same number of frames are set to conduct experiments to verify RT3C. The filtering threshold ϵ is 13. d_1 and d_2 are set to 0.3km and 3km. For the communication model, W is $2e6$, p is 0.6w, ι is 3, σ is $1e-9$, and the transmission rate of optical fiber network is $3e8$. When the patch resolution size is 320×320, the computation delays on local, edge and cloud are 0.0005s, 0.0298665s and 0.0064217s. The computation delay of the other patch resolutions is the multiple of the delay of 320×320. For the RTMAAC algorithm, the actor network is a three-layer MLP and the number of hidden layer neurons is 128. The state encoder is a one-layer MLP with 128 neurons and LeakyReLU activation and a four-head attention network and softmax layer to obtain the relation among agents. The critic network is a two-layer MLP with 256 and 128 neurons. In the inference stage, only the actor and state encoder are used to make the computation offloading decision, whose decision cost with low latency can be ignored compared with the computation task based on deep model. The learning rate of both the actor network and the critic network is 0.001, and the soft update weight τ of target networks is 0.01. The discount factor γ to update RTMAAC is 0.95, the buffer length is $1e6$, and the batch size is 256.

Deep model and dataset: For the crowd detection model, we choose two Yolov7 models respectively Yolov7-tiny and Yolov7-face [35], and deploy them on edge and cloud. There are two datasets including CroHD [24] and PANDAS [52] used to validate our RT3C. We use the training set and testing set in CroHD and part of the training set in PANDAS as different video scenes in RT3C. Since the average video resolutions in CroHD and PANDAS are $1,920 \times 1,080$ and $27,722 \times 16,842$, the patch resolutions are set to 640×640 and $3,840 \times 3,840$. In the experiment, the shortest video clip is used as the baseline to realize the synchronization decision for multiple video streams.

B. Pre-Processing Performance

The performance of key frame detection: To demonstrate the effect of key frame detection in the proposed RT3C, we perform some experiments in the four scenes on the CroHD Train dataset, and the results are shown in Fig. 6 and Fig. 7. In Fig. 6, it is assumed that all frames are transmitted to the edge or cloud to execute inference with the deep model, which is noted as only edge and only cloud respectively. The performances of these two methods are worse than the RT3C. Although an accuracy-best model is deployed on the cloud, its reward is the worst in the four scenes because there is a long distance from the local to the cloud to increases the transmission delay. Especially in

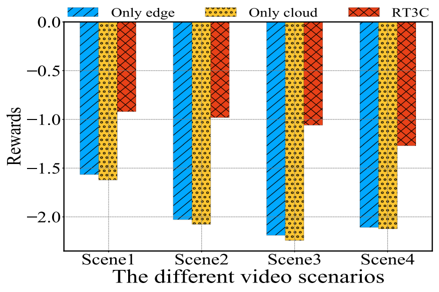


Fig. 6. Rewards when offloading frames on edge, cloud, and RT3C.

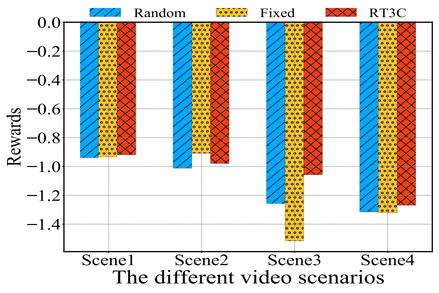


Fig. 7. Rewards when selecting frame randomly, with a fixed interval, and RT3C.

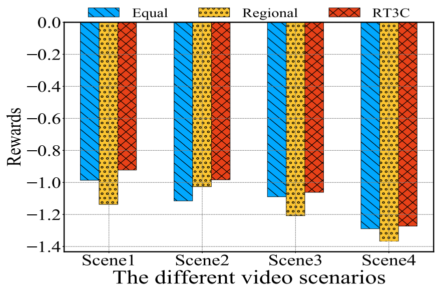


Fig. 8. Rewards when dividing the frame with equal partition, regional partition and RT3C.

Scene1, Scene2 and Scene3, the rewards of RT3C are -0.919, -0.980 and -1.058, which are even better than half of the other two methods. In Fig. 7, if the frames are randomly or with a fixed interval determined as the key frame, the RT3C performs the best performance overall. In addition, although the fixed-interval strategy presents the best performance in Scene2, its performance changes in oscillation are worse than RT3C in the rest of the scenes. Therefore, the key frame detection module is effective in the RT3C.

The performance of adaptive patch partition: To demonstrate the effect of adaptive patch partition in the RT3C, we perform experiments according to the above settings and the result is shown in Fig 8. The equal partition is to divide the whole key frame with patch resolution size to cover all regions. The regional partition is to divide the potential region with patch resolution based on the previous frame. This means that the predicted areas where all people appear in the frame are calculated without discarding abnormal crowd areas. It is observed that the proposed adaptive patch partition algorithm shows the best performance of the three methods. And the regional partition algorithm is worse than the equal partition, which demonstrates the effect

TABLE II
COMPUTATION OFFLOADING PERFORMANCE EVALUATION ON FOUR SPLIT DATASETS IN THE TWO PUBLIC DATASETS WITH OUR RTMAAC AND OTHER BASELINE METHODS

Dataset	Method	The number of scenes = 2				The number of scenes = 3				The number of scenes = 4			
		Delay ↓	MAE ↓	AP ↑	Reward ↑	Delay ↓	MAE ↓	AP ↑	Reward ↑	Delay ↓	MAE ↓	AP ↑	Reward ↑
CroHD Train	Random	0.068	30.09	0.393	-1.278	0.050	27.65	0.360	-1.345	0.048	29.01	0.425	-1.160
	DDPG [53]	0.047	29.13	0.421	-1.223	0.055	25.37	0.387	-1.343	0.053	25.66	0.421	-1.219
	MADDPG [54]	0.055	32.61	0.398	-1.275	0.040	27.54	0.355	-1.331	0.064	25.04	0.431	-1.259
	MAPPO [55]	0.035	30.52	0.419	-1.139	0.031	30.65	0.407	-1.120	0.042	23.55	0.417	-1.219
	RTMAAC	**0.017**	**24.21**	**0.427**	**-1.115**	**0.019**	**24.50**	**0.486**	**-1.033**	**0.031**	**23.12**	**0.476**	**-0.925**
CroHD Test	Random	0.059	65.79	0.441	-1.359	0.039	35.93	0.393	-1.349	0.045	72.05	0.329	-1.567
	DDPG [53]	0.042	51.89	0.465	-1.247	0.045	34.90	0.396	-1.336	0.050	**67.57**	0.338	-1.571
	MADDPG [54]	0.053	61.40	0.445	-1.352	0.046	34.94	0.397	-1.342	0.048	71.94	0.324	-1.603
	MAPPO [55]	0.030	53.579	0.459	-1.212	**0.017**	34.80	0.407	-1.167	0.031	73.87	0.327	-1.434
	RTMAAC	**0.022**	**44.25**	**0.476**	**-1.141**	**0.017**	**33.88**	**0.413**	**-1.156**	**0.028**	71.13	**0.350**	**-1.346**
PANDAS Train1	Random	0.311	40.90	0.568	-1.985	0.264	35.91	0.465	-2.202	0.289	33.04	0.476	-2.161
	DDPG [53]	0.256	33.35	0.565	-2.025	0.224	32.79	0.480	-1.996	0.274	30.07	0.465	-2.143
	MADDPG [54]	0.344	44.80	0.554	-2.001	0.220	33.61	0.479	-2.005	0.317	34.23	0.467	-2.295
	MAPPO [55]	0.293	38.86	0.565	-1.967	0.262	35.74	0.470	-2.233	0.278	32.37	0.476	-2.142
	RTMAAC	**0.205**	**29.85**	**0.581**	**-1.589**	**0.189**	**31.04**	**0.484**	**-1.773**	**0.206**	**27.53**	**0.497**	**-1.725**
PANDAS Train2	Random	0.908	344.69	**0.353**	-2.180	0.780	291.66	0.293	-2.248	0.698	225.33	0.327	-2.217
	DDPG [53]	0.957	342.32	0.331	-2.382	0.745	271.39	0.310	-2.328	0.623	202.58	0.327	-2.237
	MADDPG [54]	0.763	298.81	0.338	-2.211	0.802	288.68	0.298	-2.334	0.694	226.63	0.331	-2.130
	MAPPO [55]	0.826	327.66	0.341	-2.112	0.727	273.35	0.302	-2.198	0.665	217.06	0.329	-2.245
	RTMAAC	**0.549**	**271.97**	**0.338**	**-1.744**	**0.410**	**195.68**	**0.339**	**-1.818**	**0.393**	**163.02**	**0.352**	**-1.871**

The number of scenes is set as 2, 3 and 4 and delay, MAE, AP and reward are used to evaluate the algorithms.

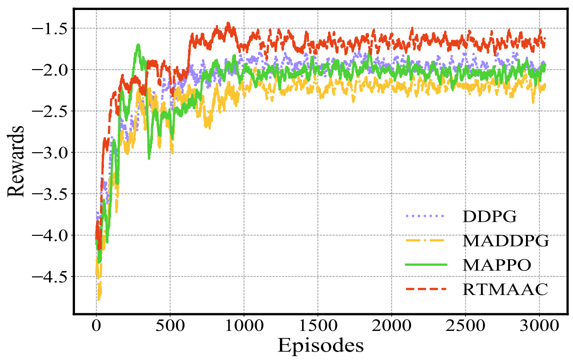


Fig. 9. Convergence curve of computation offloading algorithms based on four deep reinforcement learning algorithms.

of combining crowd distribution and estimation and discarding outliers in the RT3C.

C. Computation Offloading Performance

To verify the feasibility of RT3C, we perform experiments with multiple scenes of 2, 3, and 4 on the two datasets. Table II shows the overall performance of RT3C and the other four algorithms. The Random algorithm refers to randomly generating the computation offloading strategy. The Deep Deterministic Policy Gradient (DDPG) algorithm refers that there are multiple independent agents participating in the decisions of multi-scene frames [53]. MADDPG [54] and Multi-Agent Proximal Policy Optimization (MAPPO) [55] are classic multi-agent reinforcement learning algorithms.

The convergence of the computation offloading algorithms: The Fig. 9 shows the convergence curve for the MARL computation offloading algorithms on PANDAS Train1. Since the Random algorithms cannot reach the stable convergence, it is not shown in the figure. It is observed that the optimization problem can be solved by MARL algorithms, which get convergence after about 1000 episodes. Meanwhile, the proposed RTMAAC algorithm presents a fast convergence speed and the best rewards.

The performances of computation offloading decisions: From the Table II, it is observed that our algorithm RTMAAC achieves the best reward performance in the two datasets with a variety of scenarios. As the number of scenarios increases, resource allocation and optimization decisions become more complex. When the number of scenes is 4, compared with 2 scenes and 3 scenes, the rewards of RTMAAC are much better than these of other algorithms. For example, in the CroHD Train dataset, when there are two video streams required to be computed, the reward of RTMAAC is -1.115, while the reward of Random is -1.278, with a difference of 0.163. Due to the crowd distribution and scene differences in the dataset, there are differences in delay, MAE and AP among these four selected video clips. When the number of scenes is 2, the delay is 0.205 on PANDAS Train1 but 0.549 on PANDAS Train2. And the MAE and AP are 29.85 and 0.581 while the detection performance is weaker, with MAE and AP of 271.97 and 0.338. In addition, since the resolution of the frame in the PANDAS dataset is bigger than that on the CroHD dataset, its latency is higher than on the CroHD dataset. If the entire frame is transmitted to the edge or cloud for detection, the sum of transmission delay and computation delay is even greater. It is concluded that when the resolution is larger, it brings a larger consumption of bandwidth resources and computing resources. In RT3C, RTMAAC greatly eliminates redundant information to reduce resource consumption.

The Cumulative Distribution Function (CDF) of computation offloading decisions: The CDF of delay, MAE, AP and Reward of the five algorithms are shown in Fig. 10 when the number of scenes is 2, 3 and 4 on the PANDAS Train1 dataset. RTMAAC algorithm has significant advantages over other algorithms, which is far ahead of other algorithms in the Reward. The RTMAAC performs the best performance on the CDF of Delay, MAE and AP compared to other algorithms.

D. Sensitivity to System Settings

The effect of threshold to select key frame: We set different $\overline{\epsilon}$ to judge key frames to verify the threshold sensitivity,

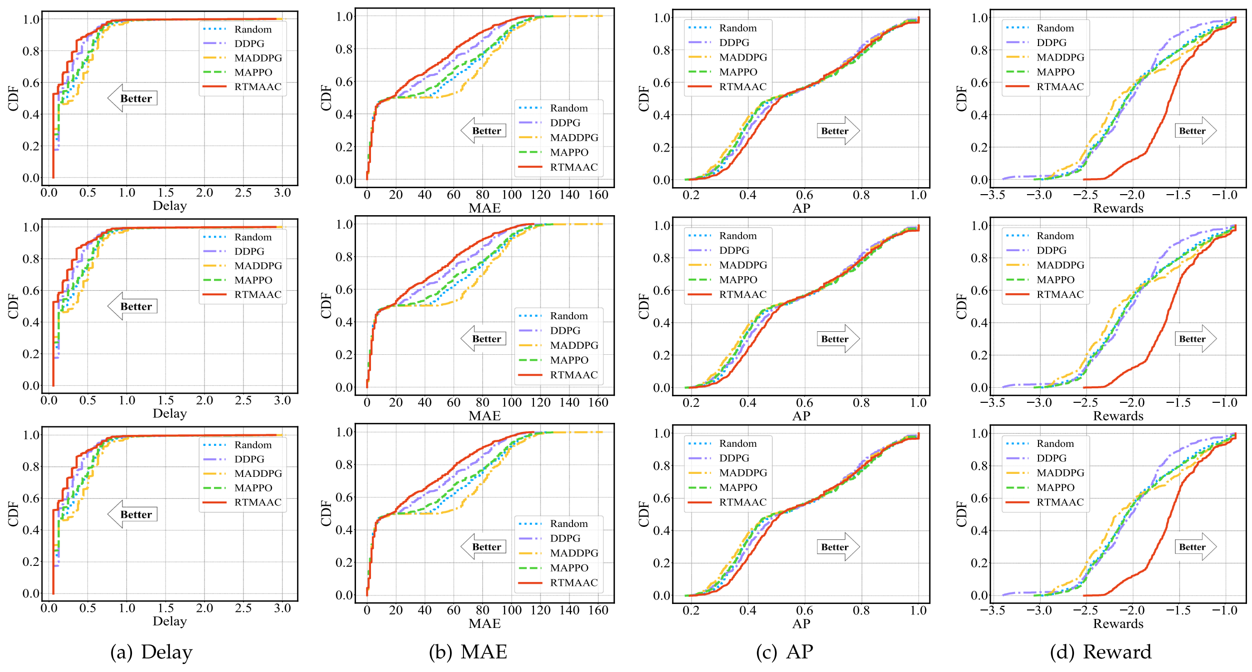


Fig. 10. CDF of delay, MAE, AP and reward for crowd counting task on PANDAS Train1 dataset. In the three rows, there are 2, 3 and 4 video streams respectively required to decide the offloading location based on the computation offloading decision algorithm.

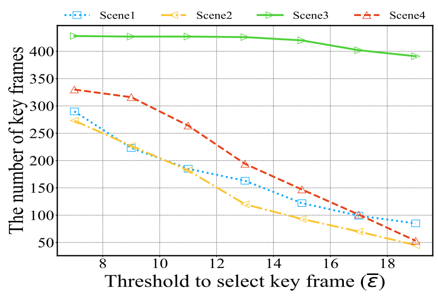


Fig. 11. Number of key frame with different threshold $\bar{\epsilon}$ to select key frame.

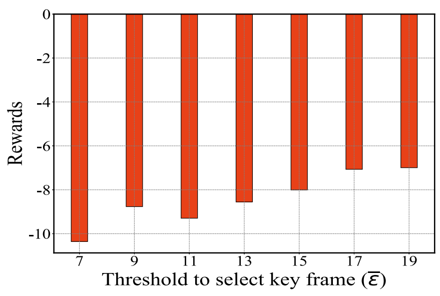


Fig. 12. Rewards when applying different thresholds $\bar{\epsilon}$ to select key frame.

as shown in Fig. 11 and Fig. 12. There are 429 frames in these scenes. By conducting experiments in four scenes on the CroHD Train dataset, the number of key frames obtained in different scenes are very different. In Fig. 11, the number of key frames in Scene3 keeps stable with the increase of $\bar{\epsilon}$, so it is a dense scene and the number of people varies greatly, which requires more frames to be computed with the deep model. As the threshold increases the number of key frames is reduced.

TABLE III
EFFECT OF THE ADAPTIVE PARTITION SIZE

Partition resolution (Λ)	Delay	MAE	AP	Reward
320×320	0.016	42.40	0.041	-1.54
540×540	0.026	26.85	0.281	-1.25
640×640	0.031	23.12	0.476	**-0.93**
860×860	0.032	34.40	0.594	-1.11
$1,080 \times 1,080$	0.046	45.40	0.636	-1.22

There are four video streams required to decide the offloading location.

In Fig. 12, as the threshold increases, the rewards function increases gradually, because the reduction of the number of key frames reduces the transmission delay and computation delay. In RT3C, the requirement for the delay is stricter than MAE and AP, so the rewards increase as the number of key frames decreases.

The effect of the adaptive partition size: We explore the effect of the adaptive partition size of RT3C on the CroHD Train dataset in the four scenes. The frame resolution of CroHD Train is $1,920 \times 1,080$ and five partition sizes are set as shown in Table III . It is observed that as the partition size increases, the AP increases but the delay also increases. When the partition patch resolution is 640×640, the reward shows the best value of -0.93. Therefore, it is important to select the proper partition size according to the frame resolution to balance the transmission and computation.

The effect of transmission bandwidth: We demonstrate the sensitivity of offloading decision algorithms to the transmission bandwidth as shown in Fig. 13. The reward is larger as the bandwidth increases since the larger bandwidth greatly reduces the transmission delay. MADDPG achieves the optimal reward of -1.232 and our RTMAAC algorithm is -1.253 with a

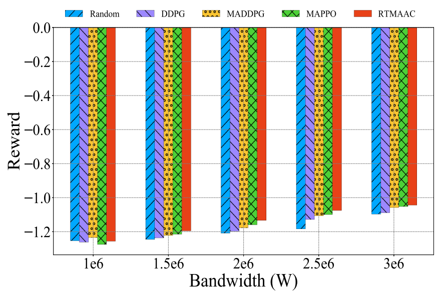


Fig. 13. Rewards when applying different transmission bandwidth W.

bandwidth of $1e6$. Our RTMAAC algorithm achieves the best performance with the other bandwidths especially when the bandwidth is $3e6$ and the reward is -1.042. It is found that our algorithm can adapt to complex network environments and maintain stable performance.

VII. Conclusions and Future Work

This paper presents RT3C, a novel framework to achieve a real-time multi-scene crowd counting system based on cloud-edge-device collaboration. Key frame detection, adaptive partition and patch encoding are established to reduce the amount of data uploaded to the server. Then, we propose an RTMAAC based on MARL to decide the computation offloading location of the patches of key frames. Extensive experiments are performed on the real crowd counting dataset to validate that the RT3C effectively reduces latency and improves counting accuracy by considering the specific video contents, especially in high-resolution multi-scene video streams with limited bandwidth resources.

There still exists more comprehensive application backgrounds which have not been considered in the RT3C. There is more than one edge and cloud and not only one type of computation-intensive task in the real application. In future work, we will target multi-edge clouds and specific multi-task scenarios to conduct research on computing offloading issues based on real system interactions. It is believed that RT3C provides an initial inspiration for video-related applications and promotes more practical research based on edge computing with multiple video streams.

References

[1] T. Kulshrestha, D. Saxena, R. Niyogi, and J. Cao, “Real-time crowd monitoring using seamless indoor-outdoor localization,” *IEEE Trans. Mobile Comput.*, vol. 19, no. 3, pp. 664–679, Mar. 2020.

[2] A. H. Fitwi, Y. Chen, and S. Zhu, “Enforcing privacy preservation on edge cameras using lightweight video frame scrambling,” *IEEE Trans. Services Comput.*, vol. 16, no. 1, pp. 276–287, Jan./Feb. 2023.

[3] C. Zheng, S. Liu, Y. Huang, and L. Yang, “Hybrid policy learning for energy-latency tradeoff in MEC-assisted VR video service,” *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9006–9021, Sep. 2021.

[4] A. C. B. Monção, S. L. Correa, A. C. Viana, and K. V. Cardoso, “Combining resource-aware recommendation and caching in the era of MEC for improving the experience of video streaming users,” *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 1698–1712, May/Jun. 2023.

[5] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, “Reducto: On-camera filtering for resource-efficient real-time video analytics,” in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl., Technol., Architectures, Protoc. Comput. Commun.*, 2020, pp. 359–376.

[6] L. Dong, Z. Yang, X. Cai, Y. Zhao, Q. Ma, and X. Miao, “WAVE: Edge-device cooperated real-time object detection for open-air applications,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4347–4357, Jul. 2023.

[7] 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.

[8] S. Jiang, Z. Lin, Y. Li, Y. Shu, and Y. Liu, “Flexible high-resolution object detection on edge devices with tunable latency,” in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 559–572.

[9] K. Du, Q. Zhang, A. Arapin, H. Wang, Z. Xia, and J. Jiang, “AccMPEG: Optimizing video encoding for video analytics,” 2022, *arXiv:2204.12534*.

[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. 2022.

[11] Y. Wu, H. Guo, C. Chakraborty, M. Khosravi, S. Berretti, and S. Wan, “Edge computing driven low-light image dynamic enhancement for object detection,” *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 5, pp. 3086–3098, Sep./Oct. 2023.

[12] H. Zhou, M. Li, N. Wang, G. Min, and J. Wu, “Accelerating deep learning inference via model parallelism and partial computation offloading,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 2, pp. 475–488, Feb. 2023.

[13] H. Zhou, T. Wu, X. Chen, S. He, D. Guo, and J. Wu, “Reverse auction-based computation offloading and resource allocation in mobile cloud-edge computing,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 6144–6159, Oct. 2023.

[14] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, “Computation offloading in multi-access edge computing: A multi-task learning approach,” *IEEE Trans. Mobile Comput.*, vol. 20, no. 9, pp. 2745–2762, Sep. 2021.

[15] J. Tan, R. Khalili, H. Karl, and A. Hecker, “Multi-agent distributed reinforcement learning for making decentralized offloading decisions,” in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 2098–2107.

[16] R. Wang, Y. Hao, L. Hu, J. Chen, M. Chen, and D. Wu, “Self-supervised learning with data-efficient supervised fine-tuning for crowd counting,” *IEEE Trans. Multimedia*, vol. 25, pp. 1538–1546, 2023.

[17] Z. Wu, X. Zhang, G. Tian, Y. Wang, and Q. Huang, “Spatial-temporal graph network for video crowd counting,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 1, pp. 228–241, 2023.

[18] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, “Chameleon: Scalable adaptation of video analytics,” in *Proc. Conf. ACM Special Int. Group Data Commun.*, New York, NY, USA, 2018, pp. 253–266. [Online]. Available: https://doi.org/10.1145/3230543.3230574

[19] X. Xiao, J. Zhang, W. Wang, J. He, and Q. Zhang, “DNN-driven compressive offloading for edge-assisted semantic video segmentation,” in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1888–1897.

[20] K. Yang, J. Yi, K. Lee, and Y. Lee, “FlexPatch: Fast and accurate object detection for on-device high-resolution live video analytics,” in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1898–1907.

[21] 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.

[22] D. Lee, Y. Kim, and M. Song, “Cost-effective, quality-oriented transcoding of live-streamed video on edge-servers,” *IEEE Trans. Services Comput.*, vol. 16, no. 4, pp. 2503–2516, Jul./Aug. 2023.

[23] R. Bhardwaj et al., “Ekya: Continuous learning of video analytics models on edge compute servers,” in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 119–135.

[24] R. Sundararaman, C. De Almeida Braga, E. Marchand, and J. Pettre, “Tracking pedestrian heads in dense crowd,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 3865–3875.

[25] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[26] M. Tan and Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 6105–6114.

[27] R. Wang et al., “Efficient crowd counting via dual knowledge distillation,” *IEEE Trans. Image Process.*, vol. 33, pp. 569–583, 2024.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[29] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7263–7271.

[30] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” 2020, *arXiv: 2004.10934*.

[31] T. Han, L. Bai, J. Gao, Q. Wang, and W. Ouyang, “DR VIC: Decomposition and reasoning for video individual counting,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 3083–3092.

[32] Z. Zhao, T. Han, J. Gao, Q. Wang, and X. Li, “A flow base bi-path network for cross-scene video crowd understanding in aerial view,” in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2020, pp. 574–587.

[33] F. Xiong, X. Shi, and D.-Y. Yeung, “Spatiotemporal modeling for crowd counting in videos,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 5151–5159.

[34] Y.-J. Ma, H.-H. Shuai, and W.-H. Cheng, “Spatiotemporal dilated convolution with uncertain matching for video-based crowd estimation,” *IEEE Trans. Multimedia*, vol. 24, pp. 261–273, 2021.

[35] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” 2022, *arXiv:2207.02696*.

[36] P. Qin, Y. Fu, G. Tang, X. Zhao, and S. Geng, “Learning based energy efficient task offloading for vehicular collaborative edge computing,” *IEEE Trans. Veh. Technol.*, vol. 71, no. 8, pp. 8398–8413, Aug. 2022.

[37] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, “Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing,” *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.

[38] Y. Chen, F. Zhao, X. Chen, and Y. Wu, “Efficient multi-vehicle task offloading for mobile edge computing in 6G networks,” *IEEE Trans. Veh. Technol.*, vol. 71, no. 5, pp. 4584–4595, May 2022.

[39] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, “Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing,” *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, Jun. 2019.

[40] A. Naouri, H. Wu, N. A. Nouri, S. Dhelim, and H. Ning, “A novel framework for mobile-edge computing by optimizing task offloading,” *IEEE Internet Things J.*, vol. 8, no. 16, pp. 13065–13076, Aug. 2021.

[41] R. Wang, J. Wang, Y. Hao, L. Hu, S. A. Alqahtani, and M. Chen, “C3Meta: A context-aware cloud-edge-end collaboration framework toward green metaverse,” *IEEE Wireless Commun.*, vol. 30, no. 5, pp. 144–150, Oct. 2023.

[42] M. Tang and V. W. Wong, “Deep reinforcement learning for task offloading in mobile edge computing systems,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[43] H. Xiao, C. Xu, Y. Ma, S. Yang, L. Zhong, and G.-M. Muntean, “Edge intelligence: A computational task offloading scheme for dependent IoT application,” *IEEE Trans. Wireless Commun.*, vol. 21, no. 9, pp. 7222–7237, Sep. 2022.

[44] H. Peng and X. Shen, “Multi-agent reinforcement learning based resource management in MEC-and UAV-assisted vehicular networks,” *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 131–141, Jan. 2021.

[45] S. S. Beauchemin and J. L. Barron, “The computation of optical flow,” *ACM Comput. Surv.*, vol. 27, no. 3, pp. 433–466, 1995.

[46] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, “DBSCAN revisited, revisited: Why and how you should (still) use dbscan,” *ACM Trans. Database Syst.*, vol. 42, no. 3, pp. 1–21, 2017.

[47] A. Skodras, C. Christopoulos, and T. Ebrahimi, “The JPEG 2000 still image compression standard,” *IEEE Signal Process. Mag.*, vol. 18, no. 5, pp. 36–58, Sep. 2001.

[48] T. S. Rappaport, “Wireless communications–principles and practice, (the book end),” *Microw. J.*, vol. 45, no. 12, pp. 128–129, 2002.

[49] A. Neubeck and L. Van Gool, “Efficient non-maximum suppression,” in *Proc. 18th Int. Conf. pattern Recognit.*, 2006, pp. 850–855.

[50] S. Iqbal and F. Sha, “Actor-attention-critic for multi-agent reinforcement learning,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 2961–2970.

[51] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 1861–1870.

[52] X. Wang et al., “PANDA: A gigapixel-level human-centric video dataset,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 3268–3278.

[53] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” 2015, *arXiv:1509.02971*.

[54] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6382–6393.

[55] C. Yu et al., “The surprising effectiveness of PPO in cooperative multi-agent games,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 24611–24624.

Rui Wang received the bachelor’s degree in computer science and technology from Lanzhou University, in 2018, China. She is currently working toward the PhD degree with Embedded and Pervasive Computing (EPIC) Laboratory, School of Computer Science and Technology, Huazhong University of Science and Technology, China. Her research interests include computer vision, emotion recognition and edge computing.

Yixue Hao (Senior Member, IEEE) received the PhD degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2017. He is an associate professor with the School of Computer Science and Technology, Huazhong University of Science and Technology. His current research interests include 5G network, Internet of Things, edge computing, edge caching and cognitive computing.

Yiming Miao (Member, IEEE) received the PhD degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2021. She is currently a research assistant professor with the School of Data Science, The Chinese University of Hong Kong, Shenzhen, China. She is reviewer of *IEEE Wireless Communications*, *IEEE Network*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Intelligent Transportation Systems*, *Future Generation Computer Systems*, *Journal of Circuits, Systems, and Computers*, *ACM Computing Surveys*. She was the symposium chair for IWCMC 2022 & 2021, the web chair for TRIDENTCOM 2017 and the CloudComp 2016. Her research interests include edge computing, 5G mobile communication system, Internet of Things, unmanned aerial vehicle, blockchain, wireless sensor network, etc.

Long Hu is an assistant professor with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), China. He was a visiting student with the Department of Electrical and Computer Engineering, University of British Columbia from 2015 to 2017. His research includes the Internet of Things, software defined networking, caching, 5G, body area networks, body sensor networks, and mobile cloud computing.

Min Chen (Fellow, IEEE) has been a full professor with the School of Computer Science and Engineering, South China University of Technology. He is also the director of Embedded and Pervasive Computing (EPIC) Lab, Huazhong University of Science and Technology. He is the founding chair of IEEE Computer Society Special Technical Communities on Big Data. He was an assistant professor with the School of Computer Science and Engineering at Seoul National University before he joined HUST. He is the chair of IEEE Globecom 2022 eHealth Symposium. His Google Scholar Citations reached 44,000+ with an h-index of 93. His top paper was cited 4,090+ times. He was selected as Highly Cited Researcher from 2018 to 2022. He got IEEE Communications Society Fred W. Ellersick Prize in 2017, the IEEE Jack Neubauer Memorial Award in 2019, and IEEE ComSoc APB Oustanding Paper Award in 2022. He is a fellow of IET.