Computer Networks 280 (2026) 112130

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

DACC: Discerning and adaptive offloading for coarse-grained content-aware video analytics ⋆

Hao Pan [a], Ning Chen [a,b,*], He Huang [a], Yu-E Sun [c,*], Xiaoyu Wang [a], Yanni Xing [c], Sheng Zhang [b], Jie Wu [d]

[a] School of Computer Science and Technology, Soochow University, Suzhou, 215006, China
[b] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[c] School of Rail Transportation, Soochow University, Suzhou, 215006, China
[d] Center for Networked Computing, Temple University, Philadelphia, PA 19122, USA

ARTICLE INFO

Keywords:
Edge video analytics
Discerning and adaptive offloading
Lyapunov optimization

ABSTRACT

Edge Video Analytics (EVA) significantly reduces response time by executing analytical tasks at the edge. However, it inevitably faces accuracy loss when dealing with highly complex analytical scenarios. Cloud-edge collaborative inference serves as an effective approach, yet ensuring low latency while improving accuracy presents a critical challenge. Moreover, determining which and how many frames to offload is challenging due to edge-cloud bandwidth constraints and the dynamic nature of video content. To tackle this problem, we propose offloading the most complex video frames to the cloud while processing other frames at the edge. We propose DACC, a Discerning and Adaptive Cloud-Edge Collaborative framework, which enables coarse-grained content awareness. DACC consists of two key components, Offloading Scheduler(OS) and Accuracy Predictor(AP). The OS determines the optimal proportion of frames offloaded to the cloud and the edge through a Lyapunov-optimization-based algorithm. It adjusts the proportion adaptively in response to time-varying resource conditions. The AP discerns the detection complexity of each frame by predicting its F1-score gain based on multi-dimensional information, performing frame-level offloading based on the proportion prescribed by the OS. Experimental results demonstrate the effectiveness of DACC, showing that our system reduce offloaded data volume by 7.1%–36.3%, decrease latency by 2.6%–19.5%, and improve accuracy by 1.72%–18.79% compared to baseline methods.

1. Introduction

As a cornerstone for latency-sensitive applications, EVA performs immediate data interpretation at the network periphery, thereby enabling swift decision-making [1,2]. This paradigm demonstrably diminishes processing delays relative to conventional cloud-centric approaches, proving especially critical in domains like autonomous driving, industrial automation, and public safety monitoring [3]. Nevertheless, EVA systems grapple with a persistent issue: securing an effective equilibrium among detection precision, latency, and resource efficiency during real-time execution, especially in dynamic settings. TileSR [4] and ResMap[5] attempt to coordinate processing latency through image patch partitioning or CNN model partitioning. Promising avenues like edge intelligence [6] and cloud-edge collaboration [7] are being explored, yet the need for adaptive frameworks that can dynamically handle fluctuating workloads remains an active research area.

The primary constraint of edge devices is their inability to effectively handle highly dynamic and complex scenarios that demand high-precision outcomes. For instance, adverse weather conditions (e.g., rain or snow) degrade visual input quality, leading to reduced detection accuracy. Similarly, small or partially occluded objects, often due to unfavorable lighting or viewing angles, are particularly difficult for edge models to process accurately. Under these circumstances, even sophisticated edge models struggle due to their inherent computational constraints, which limit effective processing of complex inputs. Offloading these frames to cloud servers [8], which employ more powerful inference models, mitigates these constraints and improve detection performance. However, this approach introduces a trade-off between

⋆ This work was supported in part by NSFC (Grant No. 62502331 and No. 62332013), Natural Science Foundation of Jiangsu province (Grant No. BK20250783), China Postdoctoral Science Foundation (Grant No. 2025M771496), State Key Lab. for Novel Software Technology of Nanjing University (Grant No. KFKT2025B24), Jiangsu Funding Program for Excellent Postdoctoral Talent.

* Corresponding authors.
E-mail addresses: hpanpanhao@stu.suda.edu.cn (H. Pan), ningc@suda.edu.cn (N. Chen), huangh@suda.edu.cn (H. Huang), sunye12@suda.edu.cn (Y.-E Sun), xywang21@suda.edu.cn (X. Wang), ynxing02@stu.suda.edu.cn (Y. Xing), sheng@nju.edu.cn (S. Zhang), jiewu@temple.edu (J. Wu).

https://doi.org/10.1016/j.comnet.2026.112130
Received 22 January 2025; Received in revised form 16 January 2026; Accepted 16 February 2026
Available online 9 March 2026

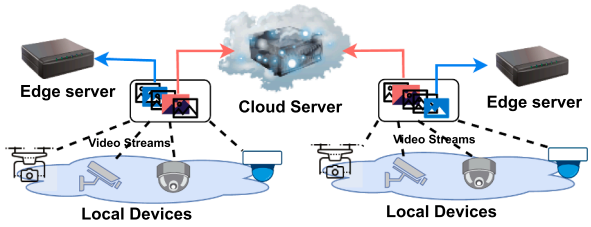


Fig. 1. A collaborative cloud-edge-end analytics scenario.

detection accuracy and latency, owing to transmission delays. The collaborative cloud-edge system provides a viable solution by leveraging joint inference across multiple models to achieve an optimal balance between accuracy and latency.

In this work, we address the aforementioned challenge by considering a real-world scenario as depicted in Fig. 1. Multiple local devices—such as surveillance cameras and drones—are connected to both an edge and a cloud server, enabling collaborative video analytics. The cloud server is equipped with a heavyweight inference model, while the edge nodes deploy relatively lightweight models. Given the resource constraints of the local devices, certain video frames are offloaded to edge nodes for basic object detection. However, the overall accuracy may be insufficient for complex scenarios, requiring more sophisticated frames to be offloaded to the cloud for higher accuracy. To reduce unnecessary transmission overhead, simpler frames are processed locally using a Detection-Based Tracking (DBT) framework [9], which supports efficient inter-frame object tracking. This scenario faces three key challenges, which we explore in detail below.

First, determining the number of frames to offload is difficult. Across different scenarios, the frame sampling rate must be adjusted to accommodate diverse requirements for accuracy and latency. In this paper, we uniformly adopt the term "offloading proportion" to quantify the extent of frame offloading. However, determining the optimal offloading proportion necessitates exploring the system's state space and conducting a comprehensive assessment that encompasses processing capability, latency, and accuracy. These parameters are either unknown or difficult to quantify precisely. Moreover, compared to the previous edge-end architecture, the incorporation of cloud inference renders the problem more intricate and challenging, as it requires the integration of heterogeneous resources and their coordinated management.

Second, adapting to the dynamic system load and bandwidth is arduous. Adaptively adjusting the offloading proportion is particularly challenging when long-term system cost and stability are considered. On the one hand, network bandwidth is highly dynamic. A fixed offloading strategy may lead to the underutilization of available resources when throughput is high, foregoing potential accuracy gains. Conversely, under limited bandwidth, such a static approach can introduce considerable transmission delays, potentially causing numerous frames to miss their execution deadlines and thereby compromising the system's real-time performance. On the other hand, the effectiveness of collaborative computing is highly dependent on the fluctuating workloads. Without dynamic adjustment of the offloading ratio, the system may encounter substantial load pressure, which can degrade overall performance and may even lead to system failure.

Finally, making offloading decisions for each frame is challenging. Beyond determining the overall offloading proportion, we must specify the execution location for every individual frame. To achieve this goal, we must understand the factors of each image such as the number of small objects within the image, whether the scene has changed, and whether there is any light occlusion. These factors are inherently variable and difficult to reflect through a single parameter. Traditional convolution-based image feature extraction can assist in decision-making, but it is computationally expensive. Alternatively, a fixed decision rule (e.g., processing only keyframes) could be adopted; however, this approach lacks the adaptability to respond to dynamic content changes.

To the best of our knowledge, existing methods struggle to achieve effective collaborative cloud-edge-end [10] offloading while meeting accuracy requirements. In previous work, some approaches make a simple offloading decision, such as setting a fixed sampling frame rate. Other systems dynamically determine the sampling frame rate [11] by considering the overall context through reinforcement learning [12]. However, these methods lack the ability to adapt the offloading strategy based on the specific content of each frame. In addition, they incur significant overhead similar to extracting image features through convolutional operations [13].

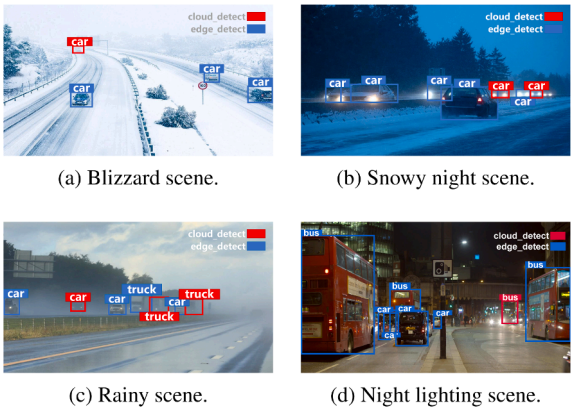
This work proposes DACC, a dynamic framework designed to address the challenges of real-time video analytics. DACC is "Discerning" as it employs a lightweight accuracy prediction model that uses coarse-grained content awareness to discern the detection difficulty of different video frames. Meanwhile, DACC is considered "Adaptive" by virtue of its capability to dynamically adjust the optimal offloading proportion in response to time-varying system load and network bandwidth conditions. The core contributions of this work are summarized as follows:

1. We model the problem with the cloud and edge offloading proportions as decision variables, formulating the objective as accuracy maximization. After transforming this NP-hard problem into a non-convex optimization form, we design a heuristic genetic algorithm to efficiently seek a high-quality solution in each time slot, aiming to maximize the utility function. This approach achieves an effective trade-off among accuracy, load, and latency.
2. To handle dynamic system load and fluctuating network conditions, we leverage a Lyapunov optimization framework to incorporate system constraints into queue stability requirements, thereby minimizing long-term queue backlog and delay. This enables dynamic offloading decisions that are responsive to both internal system state and external network conditions, ensuring efficient resource allocation and preventing excessive latency or system overload.
3. We design a lightweight Accuracy Predictor (AP) to make frame-level offloading decisions. The AP assesses the inference difficulty of each frame based on optical flow, entropy, and the number of edge pixels. The prediction accuracy guides the allocation of each frame to the most suitable location for inference. This content-aware offloading strategy minimizes resource consumption by avoiding unnecessary feature extraction while maintaining high accuracy.
4. Through extensive simulations on the UA-DETRAC and VisDrone2019 datasets, we evaluate DACC against multiple non-adaptive baselines. The results demonstrate that DACC achieves a reduction in offloaded data volume by 7.1%–36.3%, a latency decrease of 2.6%–19.5%, and an accuracy improvement of 1.72%–18.79%. The source code of DACC is open-sourced in https://github.com/twerppan/DEOF.git.

The structure of this paper is outlined as follows. Section 2 presents the motivation behind our work. The design of the proposed framework and system model are detailed in Sections 3 and 4, respectively. Section 5 then presents the problem transformation and our designed heuristic algorithm. The evaluation of our proposal through simulations is discussed in Section 6. Related work is reviewed in Section 7, and finally, Section 8 concludes the paper.

2. Motivation

This section outlines the motivation for introducing cloud detection and implementing frame-level scheduling for this paper through a series of simulation analyses.



(a) Blizzard scene. (b) Snowy night scene.

(c) Rainy scene. (d) Night lighting scene.

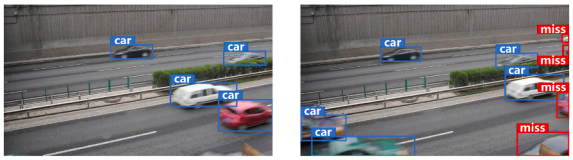
Fig. 2. Differences in cloud detection and edge detection results under various lighting conditions.

2.1. Necessity of cloud inference

The accuracy of object detection models varies significantly across different architectures. For real-time video analytics, lightweight models like YOLO Nano [14] or EdgeYOLO [15] are often deployed on resource-constrained devices. In contrast, achieving higher detection accuracy typically demands more complex and computationally intensive models, such as later members of the YOLO series or Faster R-CNN. However, the resource constraints inherent to local devices and edge servers frequently pose challenges for deploying these heavyweight models. This creates a fundamental trade-off, forcing a choice between computational efficiency and detection performance.

Our observations indicate that lightweight models can generally meet the requirements when lighting conditions are optimal and the targets move slowly in the video. However, in many real-world scenarios, accuracy remains a significant challenge. To validate this, a series of detection experiments were conducted. As shown in Fig. 2a, in snowy weather, white vehicles tend to blend into the snowy background, leading to missed detections. In Fig. 2c, rain-induced haze degrades image quality, blurring vehicle outlines and resulting in further detection failures. In addition, this issue becomes more pronounced under poor lighting conditions at night. As shown in Fig. 2b and d, lightweight models struggle to detect certain small objects, a challenge compounded by complex illumination from vehicle headlights. In these images, blue detection boxes represent the results from a lightweight model, while red boxes highlight additional targets detected by a large model that can be deployed on a cloud server. These findings collectively underscore the significant accuracy gain achievable through cloud-based inference in complex scenarios.

The challenges observed in these complex scenarios underscore the necessity of cloud-based inference. Within the DBT architecture, the high accuracy of cloud detection and the inherent errors from edge processing are propagated to subsequent tracking frames, leading to a cumulative drift in results. This observation validates our strategy of incorporating cloud inference in high-accuracy use cases. Furthermore, as the number of locally tracked frames increases, the performance gap between heavyweight and lightweight models widens, highlighting the growing value of cloud computation. The central challenge, therefore, lies in effectively orchestrating the complementary strengths of edge and cloud servers. Our approach addresses this by offloading complex frames to the cloud. By preserving the resulting high-precision bounding boxes for ongoing tracking, we fully leverage the advantages of cloud-based inference while maintaining system efficiency.



(a) Stable frame. (b) Variable frame.

Fig. 3. Comparison and analysis of detection results for frames with varying difficulty levels.

2.2. Content exploration of frame accuracy

It is well established that using the same resources does not necessarily result in the same average accuracy. Specifically, if the frames selected for offloading have little scene variation, cloud inference will not bring significant benefits. Therefore, a critical subsequent step is to determine both the timing and the specific frames to be offloaded to the cloud server. As shown in Fig. 3a, when the number of objects in the frame is small and their movement is stable, the edge model alone can successfully detect all targets. In contrast, in the scenario depicted in Fig. 3b, newly entering objects at the edge of the field of view are only detectable by the cloud model. This implies a need for detailed image content analysis, which must nevertheless operate under significant resource constraints. Through investigation of such scenarios, we identified that detection accuracy is primarily influenced by two factors: scene complexity and the degree of change.

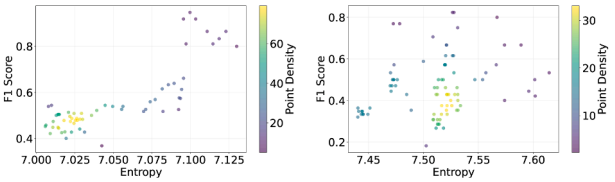
First, regarding complexity, detection difficulty predictably increases when the image background is cluttered or the number of targets is high. To quantify complexity, we define a metric called edge pixels — pixels identified as such if the color depth difference with their neighboring edge pixels exceeds a predefined threshold. Since object-background boundaries are most frequently classified as edge pixels, the number of edge pixels serves as a rough indicator of the number of targets in a fixed scene. Additionally, we extract image entropy as a supplementary measure of scene disorder.

Second, concerning the degree of change, detection becomes more challenging when objects move rapidly or enter/exit the field of view. Given our use of optical flow for local object tracking, we can preemptively assess the degree of scene change using motion information. After multiple experiments, we selected the standard deviation of sparse optical flow as our measure of change. This approach significantly reduces computational overhead while providing a coarse-grained representation of movement dynamics. Therefore, we attempt to use these features to indirectly guide the per-frame offloading strategy, and will elaborate on our method in the following sections.

3. Framework design

Based on the motivations discussed earlier, we propose DACC, a collaborative framework for real-time video analytics. This section outlines the key components of DACC, as illustrated in Fig. 5. The main objective of DACC is to facilitate collaborative video analytics, where each local device plays an integral role. These devices are equipped with an Information Extractor(IE) and an Optical-flow Tracker for target tracking, as well as an Accuracy Predictor for estimating detection accuracy. Additionally, we introduce an Offloading Scheduler (OS) that coordinates the offloading decisions for each frame within the video stream across multiple local devices.

The Accuracy Predictor is designed to estimate the detection accuracy for each frame, reflecting the complexity of the scene. Direct extraction of detailed frame-level information would be resource-intensive for local devices, so we rely on several simpler features, such as the standard deviation of the optical flow, entropy, and the number of edge pixels to predict accuracy. Since the resulting F1-score is interactively influenced by these parameters, establishing a direct functional relationship



(a) Sence1 (edge count=68040.0,std=1.688). (b) Sence2 (edge count=25235,std=0.935).

Fig. 4. F1 scores vs entropy in different scenarios.

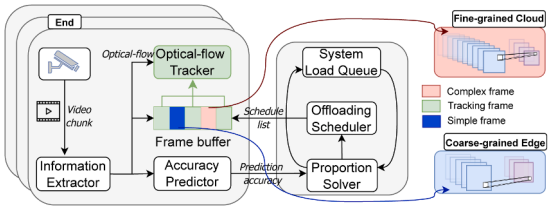


Fig. 5. Video stream processing framework of DACC.

between any single feature and F1 is infeasible. We take the relationship between image entropy and F1 score under two different scenarios as an example for brief explanation. With both the count of edge pixels and std variables fixed, Fig. 4a shows that F1 score increases monotonically with entropy, exhibiting nonlinear characteristics. In Fig. 4a, however, no apparent relationship is observed. To validate whether our proposed metrics effectively reflect detection difficulty, we instead attempted to model the non-linear relationship between the three parameters and the F1-score using regression. If a model can successfully predict F1 on a test set, it would demonstrate that the relationship between these parameters and accuracy can be captured statistically—thus providing empirical evidence that complexity and degree of change are practically linked to performance. Our experiments reveal that Random Forest Regression (RFR) can effectively capture such relationships. We will provide a detailed discussion in Section 6.3.

The Offloading Scheduler manages offloading decisions for each video frame. It incorporates a compact memory unit, the System Load Queue (SLQ), which stores the video stream parameters (e.g., image resolution) of each connected device alongside current system load information. During system initialization, the Proportion Solver (PS) within the scheduling module computes the maximum data capacity supportable by each processing unit, based on the predicted accuracy from the AP and the data in the SLQ. Using the derived offloading proportion and per-frame processing difficulty, the core SC then generates a scheduling list, which is returned to the local device. This list directs how video blocks in the device's buffer queue are transmitted to appropriate processing locations. The entire decision process is governed by algorithms (see Section 5.2) designed to ensure scheduling efficiency and optimal resource utilization across the system.

4. System model and problem formulation

In this section, we model the system in terms of accuracy, load, and latency to quantify the concept of offloading proportion based on the previous framework design. Then we formulate the entire problem and constraints in a unified manner. The notations used in this article are defined in Table 1.

4.1. Edge-cloud offloading model

We suppose that M cameras are deployed locally, denoted by $C = \{C_1, C_2, \ldots, C_M\}$. These cameras are connected to an edge node and a cloud server. Multiple edge nodes can also connect to the same cloud server. To facilitate the discussion of the impact of the proportion of offloaded frames on the entire architecture, we first fixed the frame rate f_0, resolution $r_{i,t}$, and bit rate $b_{i,t}$ of videos on each device. Each local device C_i have a video frame set $F_{i,t} = \{F_{i,1}, F_{i,2}, \ldots, F_{i,K}\}$ during slot t (The length of each slot is defined as l_t). $F_{i,t}$ represents the execution strategy for each frame in the video block. Therefore the proportion of frames offloaded during slot can be denoted by(P^e_i, P^c_i), these proportions of such multiple equipments are form the set $P^e_t = \{P^e_{1,t}, P^e_{2,t}, \ldots, P^e_{M,t}\}$ and $P^c_t = \{P^c_{1,t}, P^c_{2,t}, \ldots, P^c_{M,t}\}$, where:

$$P^e_{i,t} = \frac{\sum_{k=1}^{K} X_{i,t,k}}{f_0 l_t}, \tag{1}$$

$$X_{i,t,k} = \begin{cases} 1, & \text{if } F_{i,k} \text{ is offloaded to the Edge} \\ 0, & \text{else} \end{cases}, \tag{2}$$

$$P^c_{i,t} = \frac{\sum_{k=1}^{K} Y_{i,t,k}}{f_0 l_t}, \tag{3}$$

$$Y_{i,t,k} = \begin{cases} 1, & \text{if } F_{i,k} \text{ is offloaded to the Cloud} \\ 0, & \text{else} \end{cases}. \tag{4}$$

Table 1
Summary of notations used for system model.

Inputs	Description
B	Bandwidth
C	Set of local devices
M	Number of local devices
f_0	Video frame rate
$r_{i,t}$	Video resolution of device i in slot t
ρ	Data volume coefficient
$F_{i,t}$	Frame set of device i in slot t
K	Number of frames in slot t
l_t	Length of each slot
P^e_i	Proportion of frames offloaded to the edge
P^c_i	Proportion of frames offloaded to the cloud
Φ^e_t	Inference accuracy of the edge server in slot t
Φ^c_t	Inference accuracy of the cloud server in slot t
λ	Accuracy attenuation coefficient
$s_{i,t,k}$	Standard deviation of optical flow of frame k
$h_{i,t,k}$	Entropy of frame k
$e_{i,t,k}$	Number of edge pixels of frame k
$\bar{A}_t$	Average frame accuracy of all devices in slot t
$Q_{i,t}$	Amount of offloaded data of device i in slot t
$D_{i,t}$	Transmission delay of device i in slot t
T^e_t	Processing capability of the edge server
T^c_t	Processing capability of the cloud server
$D^{\max}$	Task deadline in slot t
V	Penalty coefficient of Accuracy

4.2. Accuracy model

The primary goal behind the design of DACC is to leverage cloud-based inference to help EVA achieve higher accuracy. Therefore, the definition of accuracy is particularly important. This includes two components: one is the accuracy predicted by the AP, and the other is the data accuracy achievable under real scheduling conditions, which serves as the reference accuracy for performance comparison. Firstly, we define the standard deviation of the optical flow $s_{i,t,k}$ as

$$s_{i,t,k} = \sqrt{\frac{1}{N_s} \sum_{i=1}^{N_s} (v_i - \mu)^2}, \tag{5}$$

where v_i represents the magnitude of the optical flow vector at the i-th point, which corresponds to the motion of the object between two consecutive frames, computed using pixel intensity variations. μ denotes the mean optical flow magnitude, which is the average of all optical-flow vector magnitudes across the selected regions. N_s is the total number of flow vectors, typically corresponding to the number of key points or feature points used to estimate optical flow.

Secondly, the entropy $h_{i,t,k}$ can be denoted as $-\sum_{i=1}^{L} p_i \log_2 p_i$. L is the number of distinct gray levels in the image, and p_i is the probability of the i-th gray level, which reflects the proportion of pixels with intensity i. Higher entropy values indicate more complexity and variation in the image. Finally, the number of edge pixels $e_{i,k}$ of $F_{i,k}$ is defined as

$$e_{i,t,k} = \sum_{i,j} \|\nabla I(i,j)\| > T. \tag{6}$$

Where $\nabla I(i,j)$ is the gradient of the image I at position (i,j), which captures the intensity changes between neighboring pixels. T is the threshold value; pixels with gradient magnitudes larger than T are considered edge pixels. The gradient magnitude $\|\nabla I(i,j)\|$ indicates how much intensity changes at that pixel, helping to identify boundaries within the image. Then the AP can output the *Prediction_accuracy_list*([0.5, 0.7, 0.4, …], etc.) based on these variables extracted through the IE.

Subsequently, we obtain the inference accuracy Φ_t^e and Φ_t^c of the DNN model on the edge server and the cloud server through offline detection, respectively. After the PS returns P_i^e and P_i^c of slot t, the OS can return the list of frame execution locations $F_{i,t}$ based on the sorted *Prediction_accuracy_list*. In this way, the practical accuracy of the k-th frame on the camera C_i during slot t $a_{i,t,k}$ can be expressed as

$$a_{i,t,k}^e = \Phi_t^e(P_e) X_{i,t,k}, \tag{7}$$

$$a_{i,t,k}^c = \Phi_t^c(P_c) Y_{i,t,k}, \tag{8}$$

$$a_{i,t,k}^l = (a_{i,t,k}^e + a_{i,t,k}^c + a_{i,t,k}^l) e^{\frac{-\lambda}{f_0}} Z_{i,t,k}, \tag{9}$$

$$Z_{i,t,k} = \begin{cases} 1, & \text{if } F_{i,k} \text{ is processed on devices} \\ 0, & \text{else} \end{cases}. \tag{10}$$

Where $a_{i,t,k}^e$, $a_{i,t,k}^c$, and $a_{i,t,k}^l$ represent the accuracy of frame $F_{i,k}$ processed on edge servers, cloud servers, and tracked by local target trackers, respectively. The attenuation coefficient *lambda* can be adjusted based on the content state to control the degree of frame accuracy attenuation under optical flow tracking. Thus the average frame accuracy $\bar{A}_t$ of all devices during slot t can be expressed as

$$\bar{A}_t = \frac{1}{MK} \sum_{i=1}^{M} \left[\sum_{k=1}^{K} \left(a_{i,t,k}^e + a_{i,t,k}^c + a_{i,t,k}^l \right) \right]. \tag{11}$$

4.3. Load model

Increasing the proportion can undoubtedly improve the overall accuracy, but it is also necessary to consider the system load under the entire architecture. We denote the load of the system by the ratio of the data volume beyond the system's processing capacity to the processing capacity. For each device C_i, the amount of data relates to the resolution of every frame. It can be expressed as $\rho r_{i,t}^2$ [16,17], where ρ is a constant. Then all the amount of data $Q_{i,t}$ offloaded to the edge and cloud can be denoted by $P_{i,t}^e f_0 l_t \rho r_{i,t}^2 + P_{i,t}^c f_0 l_t \rho r_{i,t}^2$. Similar to the model accuracy, we also offline measured the frame processing capacities of the edge server and the cloud server, which are T_t^e Mbps and T_t^c Mbps. During the whole slot, the data volume $Q'_{i,t}$ edge and cloud server can reduce is denoted by $\frac{l_t}{T_t^e} + \frac{l_t}{T_t^c}$. Therefore, the overload data volume is $Q_{i,t} - Q'_{i,t}$ (We represent the load in terms of data volume rather than frame count because the amount of data per frame can be adjusted according to resolution, and we can represent the load more adaptively when we adjust the scene or the video resolution changes. Even later work can incorporate resolution as a decision variable). The entire system load degree evolution can be represented as

$$Q_i(t+1) = \max\{Q_i(t) + \frac{Q_{i,t} - Q'_{i,t}}{Q'_{i,t}}, 0\}. \tag{12}$$

4.4. Delay model

Meanwhile, latency must be considered. Higher accuracy can be obtained when the frame is offloaded to the server for inference. However, the transmission delay is also more tremendous. We can not ignore this part. The effect of processing delay caused by DNN inference is almost equivalent to the effect of load queue reduction, which is related to the reasoning ability of DNN models. Here we only consider the transmission delay caused by offloading frames. B_1 represents the bandwidth between local and cloud, and B_2 represents the bandwidth between local and edge. The Delay $D_{i,t}$ can be expressed as

$$D_{i,t} = \frac{P_{i,t}^c f_0 l_t \rho r_{i,t}^2}{B_1} + \frac{P_{i,t}^e f_0 l_t \rho r_{i,t}^2}{B_2}. \tag{13}$$

4.5. Problem formulation

With the representation of accuracy $\bar{A}_t$ and other parameters, our optimization problem can be summarized as

$$\text{P1}: \max_{\{P_t^e, P_t^c\}} \bar{A}_t \tag{14}$$

$$s.t.\ C1: \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{M} (Q_{i,t} - Q'_{i,t}) = 0 \tag{15}$$

$$C2: \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{M} D_{i,t} \le D^{max} \tag{16}$$

$$C3: \sum_{i=1}^{M} b_{i,t} \le B \tag{17}$$

$$C4: 0 \le P_{i,t}^e \le 1 \quad \forall P_{i,t}^e \in P_t^e \tag{18}$$

$$C5: 0 \le P_{i,t}^c \le 1 \quad \forall P_{i,t}^c \in P_t^c \tag{19}$$

$$C6: 0 \le P_{i,t}^e + P_{i,t}^c \le 1 \quad \forall P_{i,t}^e \in P_t^e, \forall P_{i,t}^c \in P_t^c. \tag{20}$$

Constraint $C1$ indicates that the amount of data offloaded cannot exceed the processing capabilities of servers. Constraint $C3$ is a constraint on bandwidth between multiple devices and servers. $C4 - C6$ are several range limits for both $P_{i,t}^e$ and $P_{i,t}^c$. Where $D^{\max}$ in $C2$ represents the result cutoff time per frame. In other words, the delay distributed across each slot cannot exceed $D^{\max}$ under a long-term accumulation of time.

From the perspective of accuracy and constraint representation, our problem is a complex mixed-integer nonlinear programming (MINLP) problem that involves frame count and frame proportion. We establish its computational complexity by noting that even a simplified version of our problem, without the nonlinear constraints, is equivalent to a mixed-integer linear program which is NP-hard. Furthermore, the incorporation of nonlinear accuracy models places it in the general class of MINLP problems, for which no known polynomial-time solutions exist. Consequently, our problem is NP-hard and cannot be solved efficiently to optimality for non-trivial instance sizes. As a result, we first decouple the problem into subproblems for individual time slots and solve them separately. After that, we can convert the integer variables into proportions related to accuracy by referring to Formula 1 and Formula 3. Accordingly, the variables in the entire problem will only include the linear $P_{i,t}^e$ and $P_{i,t}^c$.

5. Lyapunov optimization and algorithm design

We know that Lyapunov theory is widely used in dealing with system stability issues under dynamic network conditions [12,18,19], and can be used to solve problems such as bandwidth [20] and resource allocation. Therefore, we apply the Lyapunov framework to our scenario which allows for a more intuitive representation of the latency in cloud processing and edge processing. This can ensure that the system achieves the maximum theoretical accuracy on the basis of stable load and delay.

5.1. Drift-plus-penalty expression

For an effective representation of the constraint, we construct a virtual evolution queue to describe $C2$,

$$D_i(t+1) = \max\{D_i(t) + \frac{D_{i,t} - D^{max}}{D^{max}}, 0\}. \tag{21}$$

Therefore the constraint problem of $C1$ and $C2$ can be transformed into the stability problem of $Q_i(t)$ and $D_i(t)$. During slot t, we defined the Lyapunov function as

$$L(H_i(t)) = \frac{1}{2}(\sum_{i=1}^{M} Q_i^2(t) + \sum_{i=1}^{M} D_i^2(t)). \tag{22}$$

Drift$\Delta H_i(t)$ is given by

$$\Delta H_i(t) = \mathbf{E}[L(H_i(t+1)) - L(H_i(t))|H_i(t)]. \tag{23}$$

By leveraging the form of drift-plus-penalty, we transform the original queue stability problem into an upper bound problem P2, where the queue stability constraint is replaced by drift and the original maximization accuracy objective is replaced by penalty.

$$\text{P2}: \min_{\{P_i^e, P_i^c\}} \Delta H_i(t) + V\mathbf{E}[\frac{1}{\bar{A}_t}|H_i(t)] \tag{24}$$

$$s.t. \quad C3 - C6$$

V is a non-negative constant and a penalty parameter of the target accuracy function in the P2 problem, expressing drift-plus-penalty like this can adjust the contribution of accuracy to the problem. If the problem is feasible, then the virtual queue rate is stable. The P2 problem has the same optimal solution as the P1 problem.

Based on the Lyapunov optimization theory, we prove the existence of an upper bound for the objective function of Problem P2, thereby ensuring the stability of the system queues. By minimizing the upper bound of the drift-plus-penalty term, the original long-term optimization problem is transformed into a distributively solvable per-slot problem (P3). This problem can be efficiently solved by a designed heuristic algorithm, and the obtained solution guarantees the satisfaction of the long-term constraints of the original problem. Detailed proofs and the problem transformation are provided in the appendix.

Algorithm 1 Offloading decision algorithm.

Input: $Prediction\_accuracy\_list$, P_i^e and P_i^c;
Output: $F_{i,t}$;
1: $F_{i,t} \leftarrow [end] * f_0$;
2: $List$ = sorted(enumerate($Prediction\_accuracy\_list$));
3: **for** $k = 1$ to $P_i^c * f_0$ **do**
4: $F_{i,t}[List[k][0]] = cloud$;
5: **end for**
6: **for** $j = P_i^c * f_0$ to $(P_i^e + P_i^c) * f_0$ **do**
7: $F_{i,t}[List[j][0]] = edge$;
8: **end for**
9: **return** $F_{i,t}$;

5.2. Algorithm design

In this part, we describe the algorithms executed within our framework. The main algorithm flow is shown in Algorithm 3. We incorporate several system-related parameters, including bandwidth, queue load values, and the predicted accuracy from the AP, all of which are stored in the OS and serve as input parameters. The objective is to determine the optimal offloading strategy. To solve for the offloading proportion, we design a genetic algorithm tailored to our scenario.

When we invoke Algorithm 3, it initializes the population with the corresponding number of individuals based on the value of pop_size. The number of iterations is determined by $generations$. The main loop, represented by lines 2–8, executes the iterative process for each generation of the population. After completing the iterations, the optimal offloading proportion and the corresponding offloading strategy are obtained (Line 10). Simultaneously, the system's queue is updated for the computation of the next time slot (Line 12).

The core iterative process is explained as follows: As shown in line 3, for each individual in the population, a mutation is performed with a probability of α to explore a larger solution space. For an individual represented by (P_i^e, P_i^c), Algorithm 1 is invoked. The purpose of Algorithm 1 is to determine the execution location of the frame based on the predicted accuracy from the AP. We know that a lower predicted F1 score indicates that the frame is likely to be more challenging during actual inference. Therefore, the frame will require a more complex inference model and should be executed on a higher-capacity server.

Algorithm 2 Average accuracy algorithm.

Input: $F_{i,t}$, Φ_t^e and Φ_t^c;
Output: $\bar{A}_t$;
1: $Accuracy\_list \leftarrow [0] * f_0$;
2: **for** $k = 1$ to f_0 **do**
3: **if** $F_{i,t}[k] == cloud$ **then**
4: $Aaccuracy\_list[k] = \Phi_t^c$;
5: **else if** $F_{i,t}[k] == edge$ **then**
6: $Accuracy\_list[k] = \Phi_t^e$;
7: **else**
8: $Accuracy\_list[k] = Accuracy\_list[k-1] * e^{-\frac{\lambda}{f_0}}$;
9: **end if**
10: **end for**
11: $\bar{A}_t \leftarrow$sum$(Accuracy_list)/f_0$;
12: **return** $\bar{A}_t$;

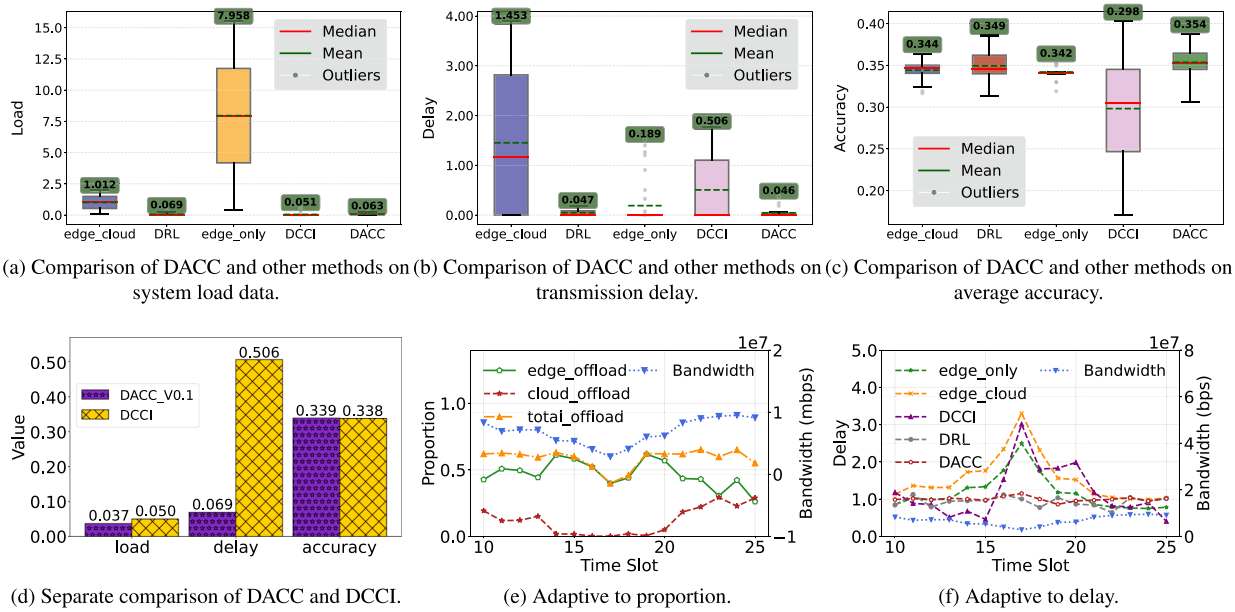
Algorithm 3 Offloading proportion algorithm.

Input: Bandwidth $B1$ and $B2$, Load queue value $Q_i(t)$, Delay queue value $D_i(t)$, $Prediction\_accuracy\_list$;
Output: $F_{i,t}$, P_i^e and P_i^c;
1: Initialize by generating pop_size populations;
2: **for** g = 1 to $generations$ **do**
3: Generate mutated populations with a probability of α;
4: Calculate $F_{i,t}$ for each pair (P_i^e, P_i^c) using Algorithm 1;
5: Calculate $\bar{A}_t$ by Φ_t^e, Φ_t^c and $F_{i,t}$ using Algorithm 2;
6: Calculate the fitness by $\bar{A}_t$, $Q_i(t)$ and $D_i(t)$;
7: Record the best individual (the highest fitness);
8: Crossover to produce offspring based on fitness with a probability of β;
9: **end for**
10: Obtain the optimal $F_{i,t}$ and (P_i^e, P_i^c);
11: Calculate the $Q_{i,t} - Q'_{i,t}$ and $D_{i,t} - D^{\max}$;
12: Update $Q_i(t+1)$ and $D_i(t+1)$;
13: **return** $F_{i,t}$, P_i^e and P_i^c;

Following this logic, we sort the $prediction\_accuracy\_list$ in ascending order. For each frame, we label it as "cloud," "edge," or "end." For example, if $P_i^c = 0.2$, $P_i^e = 0.4$, and the video block contains 30 frames, we offload the 6 frames with the lowest accuracy to the cloud, the next 12 frames to the edge, and the remaining 12 frames locally for object tracking. This strategy allows us to make accurate decisions for each frame based on the system's processing capability.

To evaluate the quality of each pair (P_i^e, P_i^c), we use the objective function in Formula A.1 and take the negative of this term as the fitness to represent the genetic population's quality. Both $Q_i(t)$ and $D_i(t)$ can be directly substituted for calculation. A crucial aspect of this evaluation is the penalty term—average accuracy. To compute this, we invoke Algorithm 2 based on the output from Algorithm 1. Algorithm 2 calculates the actual precision of the video block under each population, using the labels of each frame in the $F_{i,t}$.

For example, if the 5th, 6th, and 7th frames are assigned to the cloud, edge, and local devices respectively, the actual precision of the 5th and 6th frames will be replaced by Φ_t^c and Φ_t^e. The tracked 7th frame's precision will be the precision of the 6th frame, decayed by a coefficient



(a) Comparison of DACC and other methods on system load data.

(b) Comparison of DACC and other methods on transmission delay.

(c) Comparison of DACC and other methods on average accuracy.

(d) Separate comparison of DACC and DCCI.

(e) Adaptive to proportion.

(f) Adaptive to delay.

Fig. 6. Performance of DACC compared to other baselines.

$e^{-\frac{\lambda}{f_0}}$. By calculating the average precision for the entire video block, we can compute the fitness of each individual in line 6.

For each individual in the population, we record the best individual of the current generation before completing the iteration. This allows us to generate the next generation of the population, using a crossover probability of β.

6. Evaluation

In this section, we demonstrate the performance improvement of DACC through various experiments.

6.1. Experimental setup

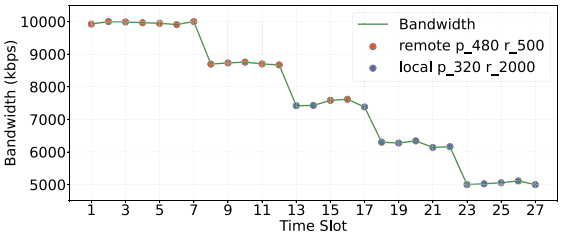
We select several continuous sequences of frames from the UA-DETRAC [21] and VisDrone2019 [22] datasets. The UA-DETRAC benchmark comprises a collection of real-world traffic video sequences, annotated with vehicle positions across frames, for evaluating vehicle detection and tracking algorithms. The VisDrone2019 dataset is a large-scale UAV-captured benchmark for object detection and tracking, providing annotated video sequences of vehicles and pedestrians in real-world aerial scenes. To process these frames, our deployment strategy is structured across three tiers. On the cloud server, equipped with dual 3090 GPUs dedicated to inference, we employ the heavyweight YOLOv7 model. The edge tier, powered by a 1660ti GPU, hosts the computationally efficient Edge-YOLO [15] as a lightweight alternative. For the local device, target tracking is achieved using optical flow.

Since our goal is not to compare models but to demonstrate the superiority of cloud inference and our overall strategy, we directly use the pre-trained weights. Additionally, owing to the dataset's lack of annotations for non-motorized vehicles and the presence of ignored regions, our detection results exhibit high recall but low precision. Consequently, the F1 score is adopted as the primary evaluation metric. Across multiple simulation runs on the UA-DETRAC, the two models achieve average F1 scores of 0.407 and 0.355, respectively. Different theoretical F1 scores only reflect the performance gap between models. For example, when a stronger model is deployed on the cloud, which has a higher theoretical F1 score, DACC will make the most appropriate decisions. It will choose to offload more frames to the cloud to leverage its performance advantage. Furthermore, we compare our proposed DACC method against several benchmarks in the simulation, assessing performance based on load queue, delay queue, accuracy, bandwidth adaptation, and other relevant metrics.

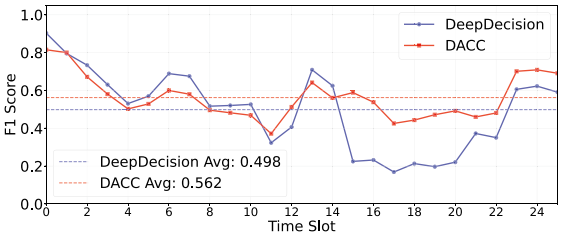
- **Fixed edge offloading proportion** : This scheme offloads frames only to the edge server, and we adjust the offloading proportion to ensure that the results are comparable with other schemes. Furthermore, we strive to ensure an even distribution of these frames, in a manner akin to the fixed frame downsampling approach.
- **Fixed edge and cloud offloading proportion** : This scheme uses fixed offloading proportions for both the edge and the cloud. But this method adopts a frame allocation strategy consistent with DACC, in order to demonstrate DACC's dynamic adaptability compared to static cloud-edge collaborative systems.
- **DRL-based agent**: Building upon prior work that utilized reinforcement learning to optimize video analytics configurations, we train a simplified RL agent. Since the agent cannot make per-frame decisions, we define the cloud and edge offloading ratios as its action space and employ the same AP as in our DACC framework. This approach serves to evaluate DACC's capability in managing long-term system load and latency.
- **DCCI**: DCCI aims to train discriminators to distinguish the image as a hard case or simple case. It chooses different discriminators based on bandwidth and load. We apply the discriminator concept to our scenario. We adopt some settings from the original article. Specifically, we train four discriminators with IoU of 0.2, 0.5, 0.6, and 0.7 respectively, and apply them with different bandwidth intervals. The frames classified as hard cases are offloaded to the cloud, and others are kept in local devices for tracking.

6.2. Baseline comparison

This part demonstrates the performance improvement of DACC over other algorithms. We implement four additional algorithms in the same scenario for comparison. Fig. 6a to Fig. 6c present the box plots of load, delay, and accuracy values across different methods. The results indicate that DACC achieves the best performance in most cases, although it ranks second to DCCI in terms of load. This is because DCCI's strategy involves offloading frames only to the cloud server for processing, which, however, leads to significantly higher latency compared to all other methods.

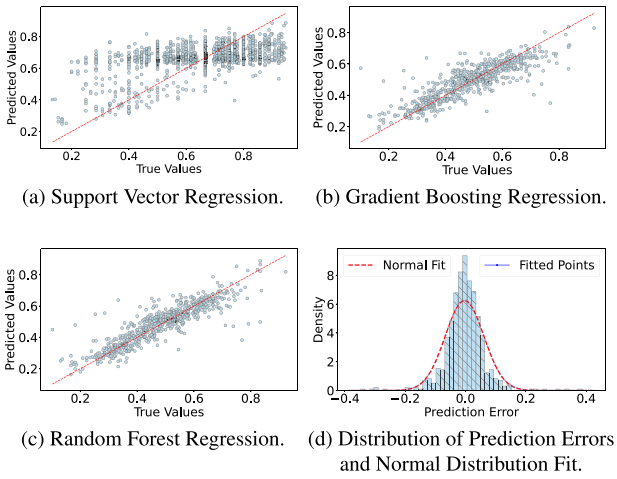


(a) Bandwidth trajectory and th best decision of DeepDecision.



(b) Accuracy of DACC and DeepDecision at different time slots.

Fig. 7. Comparison of DACC and DeepDecision in real scenario.



(a) Support Vector Regression. (b) Gradient Boosting Regression.

(c) Random Forest Regression. (d) Distribution of Prediction Errors and Normal Distribution Fit.

Fig. 8. Prediction distribution of accuracy-predictor.

To demonstrate the superiority of our method over DCCI, we relax the accuracy constraint by adjusting the parameter V, which allow us to minimize the overall load and delay as much as possible. As illustrated in Fig. 6d, when V is set to 0.1, DACC outperforms DCCI across all metrics. Additionally, it can be observed that while the edge-only approach achieves accuracy comparable to DACC, it does so at the cost of system load efficiency by offloading the majority of frames. The DRL-based method attains results closest to those of DACC. However, it is incapable of performing frame-level scheduling and instead employs the AP from our approach to make per-frame decisions. Furthermore, the fixed edge-cloud method fails to dynamically adjust the offloading volume in response to bandwidth and system load variations, resulting in consistently inferior performance across all evaluated aspects.

To further evaluate the bandwidth adaptability of DACC, we simulate a scenario with fluctuating bandwidth conditions. As depicted in Fig. 6e, a significant decrease in bandwidth occurs after the 10th time slot. The system initially maintains a stable offloading proportion, but once the persistent low bandwidth begins to affect the overall delay, DACC adaptively adjusts the offloading proportion. Specifically, given that the bandwidth between local devices and the cloud server is considerably lower than that to the edge server, DACC reduces the proportion of frames offloaded to the cloud while moderately increasing those sent to the edge. This strategy effectively mitigates accuracy loss while ensuring delay constraints are satisfied, as clearly shown in Fig. 6f. In contrast, other methods lack this dynamic adaptation capability, leading to significant delay fluctuations between time slots 10–20 due to their inability to effectively adjust offloading volume effectively in response to bandwidth changes.

Simulation experiments alone are insufficient to demonstrate the high adaptability of our method to bandwidth and load variations. Therefore, we implement a Flask-based application that employs DACC for practical video frame scheduling. By measuring function call times, we observe that our main genetic algorithm requires only approximately 100 milliseconds per time slot to make frame scheduling decisions, which does not compromise the real-time performance of video analytics.

For comparison, we reproduced DeepDecision[23]. As shown in Fig. 7, subfigure 7a illustrates the decision variations of DeepDecision under a new bandwidth curve, while subfigure 7b presents the average accuracy performance of both methods across different time slots in a real-world scenario. Since in the modeling of Section 4.1, we explained that the computational load and transmission data volume of images are entirely represented by image size (resolution), we did not include resolution as one of our adjustable parameters. In contrast, DeepDecision dynamically adjusts both resolution and bitrate, sometimes sacrificing resolution to offset the latency caused by remote offloading, in exchange for the performance compensation from the cloud model. Therefore, it is not possible to directly compare the latency and load of the two approaches on the same dimension. Instead, we demonstrate how both methods adjust their respective offloading strategies to achieve desired accuracy under the same bandwidth conditions. However, particularly under low-bandwidth conditions, DACC's more flexible frame-level scheduling significantly enhances overall accuracy.

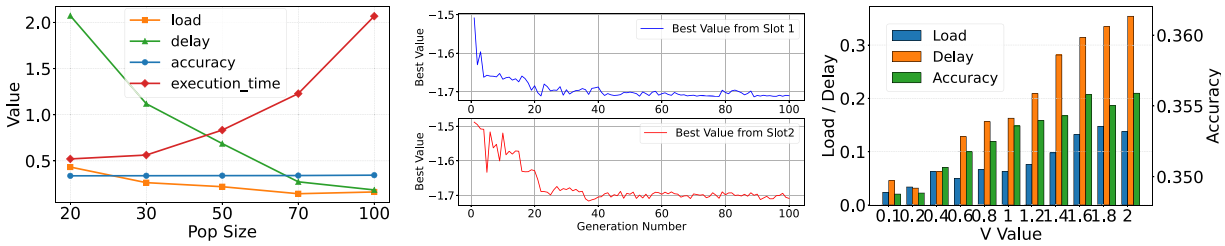
6.3. Accuracy predictor and parameter weight

In Section 3, we introduced the use of a regression model as a predictor to estimate detection accuracy for each frame. In this part, we detail the process of selecting the appropriate model. We sequentially trained three lightweight regression models on the UA-DETRAC and VisDrone datasets: Support Vector Regression (SVR), Gradient Boosting Regression (GBR), and Random Forest Regression (RFR). As shown in Fig. 8a to Fig. 8c, we selected RFR as our Accuracy Predictor (AP) due to its lowest Mean Squared Error (MSE) value of 0.00408. In comparison, the predictions from SVR exhibited significant deviations from the actual values, performing substantially worse than RFR. We also present the prediction error distribution of RFR in Fig. 8d. These results align with our expectations, confirming that the three features are sufficient for evaluating detection difficulty.

As described in the previous section, our main algorithm is a heuristic genetic algorithm whose different parameters significantly influence the execution time and system performance. Therefore, we iteratively adjust and select the optimal parameters through experiments.

Firstly, we investigate the size of the genetic population. Fig. 9a illustrates the behavior of the load queue, delay queue, accuracy, and execution time per slot for population sizes of [20, 30, 50, 70, 100]. As evident from the figure, a larger population size enables the algorithm to explore a larger solution space, potentially leading to better performance. However, this also means the algorithm will incur a greater delay in execution. As a compromise, we chose a population size of 50 as a compromise. With this population size, the execution time per slot is only 0.834 seconds, while still meeting the required performance standards.

Secondly, varying the number of iterations can yield similar outcomes. We simulate the iterative process for two different time slots in this scenario. As shown in Fig. 9b, although the iterations required for convergence differ between the segments, it can be observed that



(a) The effect of different population sizes on algorithm performance.

(b) Iterations required for convergence under two segments.

(c) Effect of different coefficients(V).

Fig. 9. The impact of certain parameters on algorithm executions.

Table 2
Counts of optimal mutation and crossover rate.

parameter	load_count	delay_count	accuracy_count
$\beta=0.01$	0	0	20
$\beta=0.03$	2	1	2
$\beta=0.07$	5	5	0
$\beta=0.1$	17	16	11
$\beta=0.2$	29	26	17
$\alpha=0.5$	14	8	9
$\alpha=0.6$	8	12	13
$\alpha=0.7$	13	12	18
$\alpha=0.8$	11	12	6
$\alpha=0.9$	4	6	4

the algorithm tends to stabilize after 50 iterations. Additionally, increasing the number of iterations also results in longer execution time. Therefore, we select 50 as the final number of iterations for our genetic algorithm.

Finally, for the genetic algorithm, both the crossover rate α and mutation rate β require tuning. Since heuristic algorithms can produce different results, we conduct additional experiments to determine optimal values for these parameters. We test α values from [0.5, 0.6, 0.7, 0.8, 0.9], and β values from [0.01, 0.03, 0.07, 0.1, 0.2]. The maximum value of β was set to 0.2 to encourage broader exploration and prevent convergence to suboptimal solutions. As shown in Table 2, each parameter set was repeat 50 times. The configuration with $\beta = 0.2$ achieve the highest frequency of minimum load and latency, and the second-highest frequency of maximum accuracy. Similarly, yield the most frequent minimum latency and maximum accuracy, and the second-most frequent minimum load. Based on these results, we finalize the value for all algorithm parameters.

When using the genetic algorithm to solve for the final decision variable, we observe that the coefficients of different penalty terms enable different trade-offs between load, latency, and accuracy. This is illustrated in Fig. 9c. A larger value of V indicates a stronger constraint on accuracy, causing the algorithm to favor higher and more stable accuracy at the expense of load and latency constraints. To make the trade-off relationship among the three more intuitive, we choose to display the reciprocal of accuracy to determine the optimal trade-off point. In the end, we use 1 as the penalty coefficient.

Actually, the accuracy and the latency achieved by our algorithm depend on not only unstable bandwidth and varying queue values but also the system processing capacity and frame processing deadline. Different frame processing deadlines represent the system's tolerance to task delay. In previous experiments, we set $D^{\max}$ for each frame to be only $1.5/f0$ sec. Increasing latency tolerance allows more frames to be offloaded, which leads to higher overall accuracy and improves system stability. We evaluate different values of $D^{\max}$ and selected 1.5 seconds as the value under the bandwidth trajectories simulated in this paper to ensure the real-time performance of the analytics.

7. Related work

A significant amount of prior work has focused on video stream processing, addressing various aspects of offloading frameworks, video configuration adjustments, and optimizations in video stream transmission. These studies either propose novel offloading architectures or explore methods to adapt video configurations in response to network conditions, aiming to improve the efficiency of video analytics systems.

7.1. Offloading frameworks

DeepDecision [23] introduces a framework that dynamically selects whether a DNN model should be executed locally or in the cloud, based on the current video configuration parameters. Building on this approach, REACT [24] utilizes cloud resources to execute large-scale DNN models and adjusts the frequency of edge-cloud detection through an edge manager to balance the high accuracy of the cloud with the low latency of the edge. In a similar vein, OSMOTICGATE [25] proposes the HQM architecture based on the dynamic workload of the system, which leverages a two-stage gradient algorithm (PGS-VAO and PGD-VAO algorithms) to optimize the overall minimum latency. DDS [8] initially transmits low-quality video streams, then re-encoding and retransmitting the areas that require precise inference after receiving the feedback region detected for the first time. However, these architectures are either confined to edge-only scenarios or overlook the coexistence of fluctuating load and bandwidth. In the experimental section, we primarily compare two influential methodologies: DeepDecision and REACT. By simultaneously addressing both of the aforementioned challenging conditions, DACC demonstrates better adaptability to varying bandwidth and load conditions, as well as superior accuracy performance.

7.2. Configuration adjustment

As previously noted, the optimal configuration for video offloading is not static and varies depending on the context. Some work [4,26] attempts to explore the optimal solution space. For example, Chameleon [27] reduces the configuration space by learning the top-K configurations and their independence within a slot. AdaDSR [28] adjusts downsampling ratios dynamically. DAO [29] introduces an Accuracy Estimator to predict analytics accuracy, greedily selecting the optimal bitrate and resolution for each chunk to accommodate dynamic video content. [30] and [31] have also designed a similar collaborative cloud-edge architecture to dynamically adjust the pipeline knobs to achieve a better balance. All these architectures share a common denominator: their inability to perform frame-level scheduling as DACC does.

7.3. Integration of reinforcement learning and Lyapunov theory

Several approaches frame video offloading as a single-time-slot optimization problem. For example, JCAB [12] incorporates factors like bandwidth into the state space and uses Markov approximation for

model and frame rate selection, while [19] leverages Markov properties across time slots with an edge-coordinated reinforcement learning algorithm for query and resource management. The rise of Deep Reinforcement Learning (DRL) has enabled various studies to solve the optimal configuration of video offloading [32,33]. CASVA [34] trains a DRL model to select configurations for each segment based on experience, adapting to fine-grained network bandwidth and dynamic video content. Batch Adaptative [35] adapts the optimal transmission batch size using an Actor-Critic architecture. However, in comparison, DACC requires significantly fewer training resources while achieving superior performance over reinforcement learning.

7.4. Content awareness

As demonstrated by [27,29,34,36], the optimization of video analytics systems is inherently tied to the content being analyzed. A significant body of publications have been devoted to the influence of video content. For example, [37] proposes a content-based video popularity prediction caching strategy, which extracts video features and applies clustering and popularity prediction models to effectively optimize the caching efficiency of cellular networks. ELF [38] predicts the region of interest and segments the video frame. On this basis, ELF offloads these regions to multiple edge servers in parallel, to accelerate the application of mobile depth vision. DCCI [13] determines hard case or simple case based on content complexity. Thus, it is decided whether the frame is inferred at the cloud server or locally. But we contend that the exploration of video content should be coarse-grained; otherwise, excessive resource consumption will prove counterproductive.

8. Conclusion

In this paper, we consider the frame offloading problem in a cloud-edge-end collaborative architecture. Our objective is to achieve the theoretical maximum accuracy under the constraint of long-term system load and latency requirements, specifically for scenarios with high precision demands. We use Lyapunov optimization theory to model DACC and solve the optimal offloading proportion through a heuristic genetic algorithm. Additionally, we design a predictor that predicts a target accuracy for each frame to implement the optimal offloading strategy. Through comparison with multiple baseline methods, our approach outperforms them in terms of accuracy and effectively adapts to dynamic changes in both the system and video content.

Our approach still has some shortcomings that require improvement in the future. First, the Accuracy Predictor (AP) in its current form requires dedicated training for each specific scenario, as building a fully general-purpose AP remains theoretically challenging. We plan to design a more versatile AP architecture in the future, which would only need light fine-tuning with minimal data when applied to new scenarios. Second, our current scheduling operates only at the frame level, while video analytics involves other tunable parameters—such as resolution and bitrate—that could be jointly optimized. As the number of such decision variables grows, the algorithm's execution time may increase significantly. We therefore aim to develop more efficient optimization methods to handle this multi-dimensional decision space in real time. Although our framework has some issues, we believe it has practical value and can be further improved in the future.

CRediT authorship contribution statement

Hao Pan: Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Ning Chen:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization; **He Huang:** Supervision, Resources, Project administration, Funding acquisition; **Yu-E Sun:** Supervision, Resources, Funding acquisition; **Xiaoyu Wang:** Supervision, Resources; **Yanni Xing:** Formal analysis, Data curation; **Sheng Zhang:** Supervision; **Jie Wu:** Supervision.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. APPENDIX

A.1. Upper bound proof

In Lyapunov optimization theory, we can prove that the optimal value of the P2 problem is bounded, which can indicate that the rate is stable. Through greedy minimization of the right formula, we can achieve the goal of obtaining the maximum accuracy value. Here we prove $\frac{\mathbf{E}}{2}\sum_{i=1}^{M}\{[Q_i^2(t+1)-Q_i^2(t)][D_i^2(t+1)-D_i^2(t)]\}$ has an upper bound.

$$
\begin{aligned}
&(Q_i(t+1))^2-(Q_i(t))^2\\
=&(\max\{Q_i(t)+\frac{Q_{i,t}-Q'_{i,t}}{Q'_{i,t}},0\})^2-(Q_i(t))^2\\
\le&(\frac{Q_{i,t}-Q'_{i,t}}{Q'_{i,t}})^2+2(\frac{Q_{i,t}-Q'_{i,t}}{Q'_{i,t}})Q_i(t)\\
=&[\frac{f_0 l_t \rho r_{i,t}^2(P_{i,t}^e+P_{i,t}^c)}{Q'_{i,t}}]^2\\
&+2(\frac{f_0\rho r_{i,t}^2(P_{i,t}^e+P_{i,t}^c)}{Q'_{i,t}})(Q_i(t)-1)+1-2Q_i(t)
\end{aligned}
$$

Then we use J instead of $f_0 l_t \rho r_{i,t}^2$ to simplify the equation. The constant $1-2Q_i(t)$ is represented by S_1, then we can get the total load queue drift by summing multiple device loads.

$$
\begin{aligned}
&\sum_{i=1}^{M}[(Q_i(t+1))^2-(Q_i(t))^2]\\
\le&[(\frac{J}{Q'_{i,t}})^2+2\frac{J(Q_i(t)-1)}{Q'_{i,t}}]\sum_{i=1}^{M}(P_{i,t}^e+P_{i,t}^c)+MS_1\\
\le&M[(\frac{J}{Q'_{i,t}})^2+2\frac{J(Q_i(t)-1)}{Q'_{i,t}}]+MS_1.
\end{aligned}
$$

Similar proofs process can be used for delay queue.

$$
\begin{aligned}
&(D_i(t+1))^2-(D_i(t))^2\\
=&(\max\{D_i(t)+\frac{D_{i,t}-D^{max}}{D^{max}},0\})^2-(D_i(t))^2\\
\le&[\frac{J}{D^{max}}(\frac{P_{i,t}^c}{B_1}+\frac{P_{i,t}^e}{B_2})]^2\\
&+2\frac{J}{D^{max}}(\frac{P_{i,t}^c}{B_1}+\frac{P_{i,t}^e}{B_2})(D_i(t)-1)+1-2D_i(t).
\end{aligned}
$$

We use S_2 instead of $1-2D_i(t)$. Through $C4$ and $C5$, the total delay queue drift for multiple devices.

$$
\begin{aligned}
&\sum_{i=1}^{M}[(D_i(t+1))^2-(D_i(t))^2]\\
\le&\sum_{i=1}^{M}[(\frac{J}{D^{max}})^2(\frac{P_{i,t}^c}{B_1}+\frac{P_{i,t}^e}{B_2})^2\\
&+2\frac{J}{D^{max}}(\frac{P_{i,t}^c}{B_1}+\frac{P_{i,t}^e}{B_2})(D_i(t)-1)]+MS_2\\
\le&M[(\frac{J}{D^{max}})^2(\frac{1}{B_1}+\frac{1}{B_2})^2
\end{aligned}
$$

$$+2\frac{J}{D^{max}}(\frac{1}{B_1}+\frac{1}{B_2})(D_i(t)-1)+S_2].$$

For $\bar{A}_t$, it is easy to prove that the reciprocal of penalty is always less than the maximum accuracy A^*. Thus the penalty has an infimum $\frac{1}{A^*}$.

A.2. Problem transformation

After the proof is completed, the goal is to minimize the drift-plus-penalty term which involves a large number of unknown input parameters, $P^e_{i,t}$ and $P^c_{i,t}$. We can treat the minimization of the system's drift-plus-penalty as a distributed problem, where we minimize the upper bound of each distributed device. That is, for device C_i, we only need to minimize the following expression, which can be reformulated as P3 related to $P^e_{i,t}$ and $P^c_{i,t}$.

$$\begin{aligned}
\text{P3}: \min_{\{P^e_t,P^c_t\}} \; & (\frac{J}{\frac{l_t}{T^e_t}+\frac{l_t}{T^c_t}})^2(P^e_{i,t}+P^c_{i,t})^2 \\
& +2\frac{J}{\frac{l_t}{T^e_t}+\frac{l_t}{T^c_t}}(Q_i(t)-1)(P^e_{i,t}+P^c_{i,t}) \\
& +(\frac{J}{D^{max}})^2(\frac{P^c_{i,t}}{B_1}+\frac{P^e_{i,t}}{B_2})^2 \\
& +2\frac{J}{D^{max}}(\frac{P^c_{i,t}}{B_1}+\frac{P^e_{i,t}}{B_2})(D_i(t)-1) \\
& +V\frac{1}{\sum_{k=1}^{K}\left(a^e_{i,t,k}+a^c_{i,t,k}+a^l_{i,t,k}\right)} \qquad \text{(A.1)} \\
s.t. \quad & C3-C6
\end{aligned}$$

For the simplified drift-plus-penalty term, it is obvious that this is a non-convex optimization problem. In previous works, such as JCAB, the reinforcement learning method is adopted to take actions according to complex state environment and obtain the maximum reward. So as to obtain the approximate optimal solution. Although the environment follows the Markov property, the difficulty of sample collection and the high cost of calculation make it not friendly for the solution of this paper. Since our decision variables are only $P^e_{i,t}$ and $P^c_{i,t}$, we decided to design a heuristic genetic algorithm to solve for the approximate optimal value.

We explain the relationship between P3 and the original optimization problem. If the P3 problem has an upper bound χ, meaning that the virtual queue rate is stable, the long-term constraints of the original optimization problem are satisfied. In this case, the minimum value of the P3 problem obtained by the decision variables $P^e_{i,t}$ and $P^c_{i,t}$ will correspond to the maximum value of the original problem.

$$\begin{aligned}
L(H_i(t+1))-L(H_i(t))+V(\frac{1}{\bar{A}_t}) &\le \chi \\
L(H_i(t+1))-L(H_i(t)) &\le \chi \\
L(H_i(T))-L(H_i(0)) &\le T\chi \\
L(H_i(0))\ge 0 \;:\; L(H_i(T)) &\le T\chi \\
0\le Q_i(t), D_i(t) &\le \sqrt{T\chi} \\
\lim_{T\to\infty}\frac{Q_i(t)}{T}=\lim_{T\to\infty}\frac{D_i(t)}{T} &= 0
\end{aligned}$$

References

[1] X. Chen, W. Zhu, J. Chen, T. Zhang, C. Yi, J. Cai, Edge computing enabled real-time video analysis via adaptive spatial-temporal semantic filtering, in: 2023 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), IEEE, 2023, pp. 262–267.

[2] L. Lin, X. Liao, H. Jin, P. Li, Computation offloading toward edge computing, Proc. IEEE 107 (8) (2019) 1584–1607.

[3] I.V. Pustokhina, D.A. Pustokhin, J.J. Rodrigues, D. Gupta, A. Khanna, K. Shankar, C. Seo, G.P. Joshi, Automatic vehicle license plate recognition using optimal K-means with convolutional neural network for intelligent transportation systems, IEEE Access 8 (2020) 92907–92917.

[4] N. Chen, S. Zhang, Y. Liang, J. Wu, Y. Chen, Y. Yan, Z. Qian, S. Lu, TileSR: accelerate on-Device super-Resolution with parallel offloading in tile granularity, in: IEEE INFOCOM 2024 - IEEE Conference on Computer Communications, 2024, pp. 2538–2547. https://doi.org/10.1109/INFOCOM52122.2024.10621208

[5] N. Chen, S. Zhang, S. Zhang, Y. Yan, Y. Chen, S. Lu, Resmap: exploiting sparse residual feature map for accelerating cross-edge video analytics, in: IEEE INFOCOM 2023 - IEEE Conference on Computer Communications, 2023, pp. 1–10. https://doi.org/10.1109/INFOCOM53939.2023.10228990

[6] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, A.Y. Zomaya, Edge intelligence: the confluence of edge computing and artificial intelligence, IEEE Internet Things J. 7 (8) (2020) 7457–7469.

[7] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, D.O. Wu, Edge computing in industrial internet of things: architecture, advances and challenges, IEEE Commun. Surv. Tut. 22 (4) (2020) 2462–2488.

[8] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, J. Jiang, Server-driven video streaming for deep learning inference, in: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, 2020, pp. 557–570.

[9] F. Du, P. Liu, W. Zhao, X. Tang, Correlation-guided attention for corner detection based visual tracking, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 6836–6845.

[10] C. Li, Y. Zhang, X. Gao, Y. Luo, Energy-latency tradeoffs for edge caching and dynamic service migration based on DQN in mobile edge computing, J Parallel Distrib. Comput. 166 (2022) 15–31.

[11] S. Cen, M. Zhang, Y. Zhu, J. Liu, AdaDSR: adaptive configuration optimization for neural enhanced video analytics streaming, IEEE Internet Things J. (2023).

[12] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, M. Xiao, Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 257–266.

[13] Y. Hu, Z. Li, Y. Chen, Y. Cheng, Z. Cao, J. Liu, Content-Aware adaptive device–Cloud collaborative inference for object detection, IEEE Internet Things J. 10 (21) (2023) 19087–19101.

[14] A. Wong, M. Famuori, M.J. Shafiee, F. Li, B. Chwyl, J. Chung, YOLO Nano: a highly compact you only look once convolutional neural network for object detection, in: 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS), IEEE, 2019, pp. 22–25.

[15] S. Liang, H. Wu, L. Zhen, Q. Hua, S. Garg, G. Kaddoum, M.M. Hassan, K. Yu, Edge YOLO: real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles, IEEE Trans. Intell. Transp. Syst. 23 (12) (2022) 25345–25360.

[16] J. Xue, Y. An, Joint task offloading and resource allocation for multi-task multi-server NOMA-MEC networks, IEEE Access 9 (2021) 16152–16163.

[17] W. Chang, Y. Xiao, W. Lou, G. Shou, Offloading decision in edge computing for continuous applications under uncertainty, IEEE Trans. Wireless Commun. 19 (9) (2020) 6196–6209.

[18] Z. Tong, J. Cai, J. Mei, K. Li, K. Li, Dynamic energy-saving offloading strategy guided by lyapunov optimization for IoT devices, IEEE Internet Things J. 9 (20) (2022) 19903–19915.

[19] R. Lin, T. Xie, S. Luo, X. Zhang, Y. Xiao, B. Moran, M. Zukerman, Energy-efficient computation offloading in collaborative edge computing, IEEE Internet Things J. 9 (21) (2022) 21305–21322.

[20] N. Chen, S. Zhang, Z. Ma, Y. Chen, Y. Jin, J. Wu, Z. Qian, Y. Liang, S. Lu, Vichaser: chase your viewpoint for live video streaming with block-Oriented super-Resolution, IEEE/ACM Trans. Netw. 32 (1) (2024) 445–459. https://doi.org/10.1109/TNET.2023.3286108

[21] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, S. Lyu, UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking, CVIU 193 (2020) 102907.

[22] D. Du, P. Zhu, L. Wen, X. Bian, H. Lin, Q. Hu, T. Peng, J. Zheng, X. Wang, Y. Zhang, et al., Visdrone-DET2019: the vision meets drone object detection in image challenge results, in: Proceedings of the IEEE/CVF ICCV Workshops, 2019, pp. 0.

[23] X. Ran, H. Chen, X. Zhu, Z. Liu, J. Chen, Deepdecision: a mobile deep learning framework for edge video analytics, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 1421–1429.

[24] A. Ghosh, S. Iyengar, S. Lee, A. Rathore, V.N. Padmanabhan, React: streaming video analytics on the edge with asynchronous cloud support, in: Proceedings of the 8Th ACM/IEEE Conference on Internet of Things Design and Implementation, 2023, pp. 222–235.

[25] B. Qian, Z. Wen, J. Tang, Y. Yuan, A.Y. Zomaya, R. Ranjan, Osmoticgate: adaptive edge-based real-time video analytics for the internet of things, IEEE Trans. Comput. 72 (4) (2022) 1178–1193.

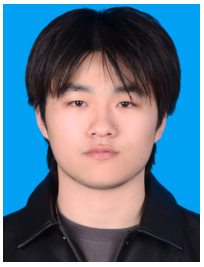
[26] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, M. Philipose, Videoedge: processing camera streams using hierarchical clusters, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2018, pp. 115–131.

[27] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Siddhartha, I. Stoica, Chameleon: scalable adaptation of video analytics, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 253–266.

[28] S. Cen, M. Zhang, Y. Zhu, J. Liu, AdaDSR: adaptive configuration optimization for neural enhanced video analytics streaming, IEEE Internet Things J. 11 (7) (2024) 11919–11929. https://doi.org/10.1109/JIOT.2023.3331699

[29] T. Murad, A. Nguyen, Z. Yan, Dao: dynamic adaptive offloading for video analytics, in: Proceedings of the 30Th ACM International Conference on Multimedia, 2022, pp. 3017–3025.

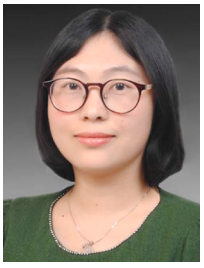
[30] C. Kai, H. Zhou, Y. Yi, W. Huang, Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability, IEEE Trans. Cognit. Commun. Networking 7 (2) (2020) 624–634.
[31] X. Wang, G. Gao, X. Wu, Y. Lyu, W. Wu, Dynamic DNN model selection and inference off loading for video analytics with edge-cloud collaboration, NOSSDAV '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 64–70. https://doi.org/10.1145/3534088.3534352
[32] N. Chen, S. Quan, S. Zhang, Z. Qian, Y. Jin, J. Wu, W. Li, S. Lu, Cuttlefish: neural configuration adaptation for video analysis in live augmented reality, IEEE Trans. Parallel Distrib. Syst. 32 (4) (2020) 830–841.
[33] N. Le, V.S. Rathour, K. Yamazaki, K. Luu, M. Savvides, Deep reinforcement learning in computer vision: a comprehensive survey, Artif. Intell. Rev. (2022) 1–87.
[34] M. Zhang, F. Wang, J. Liu, Casva: configuration-adaptive streaming for live video analytics, in: IEEE INFOCOM 2022-IEEE Conference on Computer Communications, IEEE, 2022, pp. 2168–2177.
[35] L. Zhang, Y. Zhang, X. Wu, F. Wang, L. Cui, Z. Wang, J. Liu, Batch adaptative streaming for video analytics, in: IEEE INFOCOM 2022-IEEE Conference on Computer Communications, IEEE, 2022, pp. 2158–2167.
[36] N. Chen, S. Zhang, S. Quan, Z. Ma, Z. Qian, S. Lu, VCMaker: Content-aware configuration adaptation for video streaming and analysis in live augmented reality, Comput. Netw. 200 (2021) 108513.
[37] K.N. Doan, T. Van Nguyen, T.Q.S. Quek, H. Shin, Content-aware proactive caching for backhaul offloading in cellular network, IEEE Trans. Wireless Commun. 17 (5) (2018) 3128–3140.
[38] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, Y. Zhang, Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading, in: Proceedings of the MobiCom, 2021, pp. 201–214.



Hao Pan received the BS degree from the School of Computer Science and Technology, Soochow University, China, in 2024. He is currently pursuing the M.S. degree in Computer Technology, Soochow University. His research interests include edge Intelligence, edge computing, and video streaming.

Ning Chen received the PhD degree from Nanjing University, and is currently a Lecturer in the School of Computer Science and Technology, Soochow University. His research interests including edge computing, deep reinforcement learning, and video streaming. To date, he has published over 20 papers, including those appeared in INFOCOM, ICDE, TON, TPDS, SECON, Computer Networks, ICPADS, et al.

He Huang(Senior Member, IEEE) received the PhD degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC), China, in 2011. He is currently a professor with the School of Computer Science and Technology, Soochow University, China. From 2019 to 2020, he was a visiting research scholar with Florida University, Gainesville. He has authored more than 100 papers in related international conference proceedings and journals. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a member of the Association for Computing Machinery (ACM). He received the best paper awards from Bigcom 2016, IEEE MSN 2018, and Bigcom 2018. He has served as the Technical Program Committee member of several conferences, including IEEE INFOCOM, IEEE MASS, IEEE ICC, and IEEE Globecom.



Yu-E Sun is a professor of School of Rail Transportation, Soochow University, PR China. She received her PhD degree in Shenyang Institute of Computing Technology from Chinese Academy of Science. From 2019 to 2020, she was a Postdoc Research Scholar with Florida University, Gainesville, USA. She has authored over 80 papers in related international conference proceedings and journals. Her current research interests span traffic measurement, privacy preserving, algorithm design and analysis for wireless networks. She is a Member of both IEEE and ACM. She received the best paper awards from Bigcom 2016, IEEE MSN 2018, and Bigcom 2018. She has served as the Technical Program Committee Co-Chairs of ACM MSCC-MobiHoc 2016.

Xiaoyu Wang received the BS degree from the School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu, China, in 2016, and the PhD degree from the Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu, China, in 2021. Her research interests include wireless charging and data mining. She is an Associate Professor with the School of Computer Science and Technology in Soochow University.

Yanni Xing received the B.S. in Computer Science and Technology from Soochow University (SU), China, in 2024. She is currently pursuing a Master's degree in Transportation Engineering at the School of Rail Transportation, Soochow University. Her current research interests include edge video analytics, objects detection, as well as pedestrian and vehicle tracking.

Sheng Zhang is an associate professor in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. He received the BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. His research interests include cloud computing and edge computing. To date, he has published more than 80 papers, including those appeared in *JSAC*, *TMC*, *TON*, *TPDS*, *TC*, *MobiHoc*, *ICDCS*, and *INFOCOM*. He received the Best Paper Award of IEEE ICCCN 2020 and the Best Paper Runner-Up Award of IEEE MASS 2012. He is the recipient of the 2020 ACM Nanjing Rising Star Award and the 2015 ACM China Doctoral Dissertation Nomination Award. He is a senior member of IEEE and CCF, and a member of ACM.

Jie Wu is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.