DEOF: Discerning and Elastic Offloading for Accuracy-Efficient Video Analytics

Hao Pan[1], Ning Chen[1,3*], Xiaoyu Wang[1], Yanni Xing[2], Sheng Zhang[3], Jie Wu[4]
[1]School of Computer Science and Technology, Soochow University, Suzhou, China
[2]School of Rail Transportation, Soochow University, Suzhou, China
[3]State Key Lab. for Novel Software Technology, Nanjing University, Nanjing, China
[4]Center for Networked Computing, Temple University, Philadelphia, USA
{hpanpanhao, ynxing02}@stu.suda.edu.cn, {ningc, xywang21}@suda.edu.cn, sheng@nju.edu.cn, jiewu@temple.edu

2025 IEEE 31th International Conference on Parallel and Distributed Systems (ICPADS) | 979-8-3315-4901-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICPADS67057.2025.11323028

***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. To overcome this, we propose offloading the most complex video frames to the cloud while processing other frames at the edge. Nevertheless, determining both the quantity and the specific selection of frames for offloading poses challenges due to edge-cloud bandwidth constraints and the dynamic nature of video content. To tackle this problem, we present a Discerning and Elastic Offloading Framework (DEOF), which consists of an Accuracy Predictor and an Offloading Scheduler. The former identifies the detection complexity of each frame by predicting its F1-score gain based on multi-dimensional information, enabling it to discern and select the most complex frames for offloading. The latter determines the optimal proportion of frames to process in the cloud and at the edge by designing a Lyapunov-optimization-based algorithm, which elastically adjusts this proportion in response to time-varying video content and resource conditions, thus ensuring both adaptability and efficiency. We have implemented DEOF fully based on COTS hardware, and the experimental results demonstrate the effectiveness of DEOF, showing that our system can reduce offloaded data volume by 7.1% – 36.3%, decrease latency by 2.6% – 19.5%, and improve accuracy by 2.6% – 3.2% compared to alternative methods.**

***Index Terms*—Edge video analytics, Discerning and elastic offloading, Lyapunov optimization**

I. Introduction

The widespread adoption of edge devices in fields like transportation and medicine [1] has established EVA as a pivotal technology. EVA enables real-time video processing and expedited decision-making by analyzing data directly at the network edge. This significantly reduces latency compared to cloud-based methods, making it valuable for time-sensitive applications like surveillance and industrial monitoring. [2] attempted to solve the optimal configuration of video offloading, [3] [4] [5] focus on optimizing video streaming performance by targeting key metrics like Quality of Experience (QoE).

However, achieving an optimal balance between accuracy, latency, and resource utilization remains a key challenge in complex environments [6]. This challenge is particularly pronounced in real-world scenarios where adverse conditions such as poor weather or object occlusion degrade input quality and detection performance. While collaborative cloud-edge systems that process simpler frames locally and offload complex ones to the cloud offer a promising solution [7], existing frameworks lack the dynamic adaptability needed for varying workloads. In this work, we address these limitations through a realistic collaborative video analytics scenario in Fig.1. Multiple local devices (e.g., surveillance cameras) are connected to edge nodes and a cloud server. To minimize unnecessary transmissions while maintaining accuracy, we employ a Detection-Based Tracking framework [8] for processing simpler frames locally, while offloading more complex frames to the cloud.

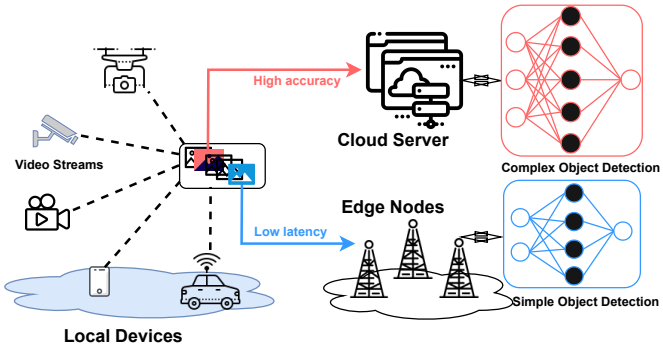


Fig. 1: A collaborative cloud-edge-end video analytics scenario.

Related Work and Challenges. *First of all, determining the optimal offloading proportion is difficult.* Chameleon[9], TileSR[10], and DAO[11] mainly focus on system parameters such as bit rate and resolution, but ignore the real-time impact of frame rate decisions on the system. Determining the optimal offloading proportion means we need to explore the system's state space, including the system's processing capability, delay effects, and accuracy performance. These parameters are mostly unknown or difficult to quantify. REACT [12] achieves a balance between the cloud and the edge by manually adjusting the detection frequency, which obviously lacks adaptability.

Second, identifying the specific frames used for offloading is non-trivial. Since our goal is to develop an execution strategy for each frame, we must understand the key characteristics of each image. Existing methods [13] and [14] propose a

* Corresponding author. This work was supported in part by NSFC (Grant No. 62502331), 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.

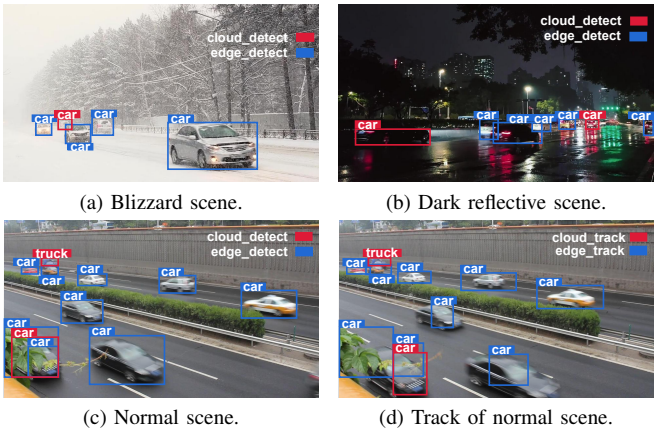
cloud-edge-end collaborative task offloading framework based on different offloading strategies, [15] only detects keyframes while relying on object tracking for the rest. Moreover, [16] dynamically adjusts downsampling ratios. A common shortcoming of these methods is that they overlook the varying detection difficulties across frames and fail to achieve true frame-level scheduling. Convolution-based image feature extraction [17] can assist in decision-making, but it is resource-intensive and consumes too much computing time.

Finally, the load and bandwidth of the system are time-varying. It is highly arduous to adjust strategy adaptively when we consider the long-term cost and stability of the system. [18] and [19] dynamically deploy neural network models on edge devices considering the impact of bandwidth. [20] applies a fixed frame detecting rate, which will inevitably cause significant transmission delays when the bandwidth is limited. This could compromise the system's real-time performance. On the other hand, the efficiency of collaborative computing is influenced by the inconstant workload. CASVA [21] dynamically determines video configuration based on bandwidth and precision requirements, DeepDecision [22] dynamically selects the execution location of a DNN model. Both of them ignore the additional impact brought by the device's workload and load capacity.

Our contribution. To address these challenges, we propose DEOF, a dynamic offloading framework for real-time video analytics that adaptively orchestrates computation across cloud, edge, and end devices. Our solution intelligently leverages real-time system states—including network bandwidth and device workload—in conjunction with fine-grained frame characteristics to determine an optimal execution strategy for each video frame. The evaluation of real-world data shows that DEOF is both elastic and discerning: elastic because it can dynamically adjust the offloading proportions, and discerning because it intelligently identifies the complexity of each frame for optimized processing.

We make the following contributions:

1) A novel problem formulation that models the dynamic offloading challenge as an NP-hard accuracy-maximization task, and the application of the Lyapunov optimization framework to transform long-term constraints into a tractable, real-time optimization problem. We integrate system state parameters and treat the optimal offloading proportions as the decision variables, and solve them via an efficient genetic algorithm;
2) A lightweight Accuracy Predictor (AP) is used to predict the detection accuracy of frames to assess their detection difficulty. AP enables fine-grained, frame-level offloading decisions by quantifying frame complexity through efficient feature extraction (optical flow, entropy, and edge density), thereby minimizing feature extraction overhead and transmission costs;
3) A comprehensive implementation and evaluation on the UA-DETRAC [23] and the VisDrone2019 [24] dataset, demonstrating that DEOF significantly reduces offloaded data volume by 7.1%–36.3%, decreases latency by 2.6%–19.5%, and improves accuracy by 2.6%–3.2% compared to non-adaptive baselines, with the source code publicly released.



(a) Blizzard scene. (b) Dark reflective scene.

(c) Normal scene. (d) Track of normal scene.

Fig. 2: Discrepancies in Cloud vs. Edge Detection Under Diverse Conditions

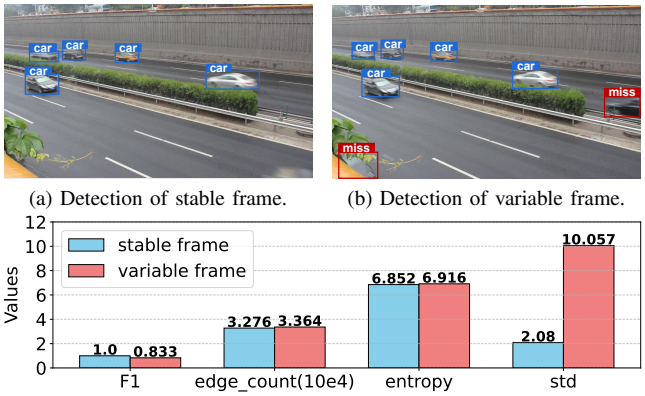
Organization. The remainder of this paper is organized as follows. Section II delineates the motivation for this paper. Sections III describe our framework and system model. We subsequently illustrate the problem transformation and the heuristic algorithm we designed in Section IV. We evaluate our proposed design using simulations in Section V. And the paper is finally concluded in Section VI.The code of DEOF is open-sourced in https://github.com/twerppan/DEOF.git.

II. MOTIVATION

A. Necessity of Cloud Inference

It is well established that different models exhibit varying accuracy in object detection, with lightweight models like EdgeYOLO [25] preferred for real-time video analytics, while heavyweight models (e.g., YOLO series) achieve higher accuracy at greater computational cost. Deployment on edge devices remains challenging due to resource constraints, forcing a trade-off between these model types. Our research shows lightweight models suffice under optimal lighting and slow target movement, but accuracy falters in complex real-world scenarios. Experiments confirm detection failures in snowy weather (vehicles blending into backgrounds, Fig.2a), rainy conditions (reflections causing false detections, Fig.2b), poor nighttime lighting, and cases of small or occluded objects (Fig.2c). Blue detection boxes (lightweight models) contrast with red boxes (cloud-deployed large models), demonstrating cloud inference's significant effect in complex scenarios.

The presence of these complex scenarios reinforces our belief that cloud-based inference is still essential. As shown in Fig.2d, when we implement video analytics using the DBT architecture, the high accuracy of cloud detection and the errors from edge detection are propagated to subsequent tracking frames. This result supports our approach of integrating cloud inference into high-accuracy scenarios. As the number of locally tracked frames increases, the accuracy gap between heavyweight and lightweight models becomes more pronounced, further enhancing the benefits of cloud inference.



(a) Detection of stable frame. (b) Detection of variable frame.

(c) Bar chart presentation of parameter values.

Fig. 3: Comparison of parameters and F1 score of different images.

B. Content Exploration of Frame Accuracy

Another key point in achieving collaborative inference between edge and cloud is to determine the offloading decision for each frame. As shown in Fig.3a, edge models alone can effectively detect all objects. Therefore, to use cloud resources reasonably, we also need to selectively filter out suitable frames for complex inference. We find that frames with more target objects and greater complexity tend to exhibit higher image entropy [26] in Fig.3c, since image entropy serves as a key indicator of information content. Besides, the comparison in Fig.3 reveals that the degree of image change is also an important factor affecting accuracy. Since our object-tracking method uses optical flow, which represents a vector field indicating the displacement of each pixel over time, and the standard deviation of optical flow reflects the degree of change in the image approximately.

An edge pixel is defined as one where the color depth difference with adjacent pixels exceeds a specified threshold. As depicted in Fig.3a and Fig.3b, objects enter and exit the scene at the boundary in the latter case. The resulting pixel changes shown in Fig.3c have a certain impact on the object detection results. We validate that the variations in the number of edge pixels largely capture the appearance and disappearance of objects through multiple experiments in the same scenario. Therefore, we consider using the entropy of the image, the standard deviation of optical flow, and the number of edge pixels as measures of the detection difficulty of the image.

III. FRAMEWORK DESIGN AND SYSTEM MODEL

The key challenge lies in optimizing the trade-off between edge-based and cloud-based inference, specifically determining offloading proportions and which frames to offload. Based on the motivations discussed earlier, we propose DEOF. This section outlines the framework of DEOF, as illustrated in Fig.4. Then we model the system in terms of accuracy, load, and latency based on the framework design. The notations used in this article are defined in TABLE I.

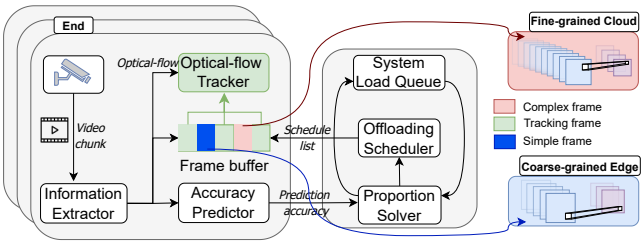


Fig. 4: Video analytics framework of DEOF.

A. Framework Design And Component Introduction

In our design, each device is 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.

The Accuracy Predictor is used to estimate the F1 value of each frame. It relies on several simple features mentioned in II-B. All of these features can be extracted by the IE. We employ Support Vector Regression (SVR) [27] to create a lightweight regression model for this task, as these features do not exhibit strong correlations. The AP is trained on the UA-DETRAC dataset [23] by selecting about 15,000 frames. We compute the values of features as the training dataset. These inputs are fed into the SVR, and the F1 score from the YOLOv7 detections is used to supervise the training process.

The Offloading Scheduler is responsible for managing the offloading decisions for each video frame. It is equipped with a small memory unit called the System Load Queue (SLQ), which records the video stream parameters for each connected device. When the system begins operation, the Proportion Solver (PS) in the scheduling module calculates the maximum data capacity that each processing unit can support based on the predicted accuracy of the AP and the information stored in the SLQ. The core OS then generates a scheduling list and sends it back to the local device. This allows the video blocks stored in the local device's buffer queue to be transmitted to the corresponding processing location according to the scheduling information. The decision-making process is guided by algorithms (outlined in Section IV-C).

B. 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 several edge nodes and a cloud server. We fix 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 $(P_i^e,\ P_i^c)$, which is $(\frac{\sum_{k=1}^{K} X_{i,t,k}}{f_0 l_t},\ \frac{\sum_{k=1}^{K} Y_{i,t,k}}{f_0 l_t})$. Binary numbers $X_{i,t,k}$ and $Y_{i,t,k}$ respectively represent the corresponding frame being offloaded to the edge or the cloud server.

TABLE I: 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_i^e	Proportion of frames offloaded to the edge
P_i^c	Proportion of frames offloaded to the cloud
Φ_t^e	Inference accuracy of the edge server in slot t
Φ_t^c	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_t^e	Processing capability of the edge server
T_t^c	Processing capability of the cloud server
D^{max}	Task deadline in slot t
V	Penalty coefficient of Accuracy

C. Accuracy, Load and Delay Model

The primary goal behind the design of DEOF is to leverage cloud-based inference to help EVA achieve higher accuracy. 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, the AP can output the list of prediction accuracy P_{acc} ([0.5, 0.7, 0.4, ...], etc.) based on the standard deviation of the optical flow $s_{i,t,k}$, the entropy $h_{i,t,k}$ and the number of edge pixels $e_{i,k}$ extracted through the IE. $e_{i,k}$ is defined as $\sum_{i,j} \|\nabla I(i,j)\| > T$, where $\nabla I(i,j)$ is the gradient of the image I at position (i,j). Pixels with gradient magnitudes larger than the threshold value T are considered edge pixels.

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 output the list of frame execution locations $F_{i,t}$. 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 $a_{i,t,k}^e = \Phi_t^e X_{i,t,k}$ or $a_{i,t,k}^c = \Phi_t^c Y_{i,t,k}$. If the frame is not offloaded, then its local tracking accuracy $a_{i,t,k}^l$ is given by

$$a_{i,t,k}^l = (a_{i,t,k}^e + a_{i,t,k}^c + a_{i,t,k}^l) e^{\frac{-\lambda}{f_0}} Z_{i,t,k}, \tag{1}$$

where λ is the attenuation coefficient of accuracy under optical flow tracking. Binary number $Z_{i,t,k}$ indicates whether the frame is processed on devices. Thus, the average frame accuracy $\bar{A}_t$ of all devices during slot t can be expressed as

$$\bar{A}_t = \frac{1}{MK} \sum_{i=1}^{M} \left[\sum_{k=1}^{K} \left(a_{i,t,k}^e + a_{i,t,k}^c + a_{i,t,k}^l \right) \right]. \tag{2}$$

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 $Q_{i,t}$ offloaded to the edge and cloud can be denoted by $P_{i,t}^e f_0 l_t \rho r_{i,t}^2 + P_{i,t}^c f_0 l_t \rho r_{i,t}^2$ [28]. 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 $Q_{i,t} - Q'_{i,t}$. The entire system load degree evolution can be represented as

$$Q_i(t+1) = \max\{Q_i(t) + \frac{Q_{i,t} - Q'_{i,t}}{Q'_{i,t}}, 0\}. \tag{3}$$

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

D. Problem Formulation

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

$$P1: \max_{\{P_t^e, P_t^c\}} \bar{A}_t \tag{4}$$

$$s.t.\ C1: \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{M} (Q_{i,t} - Q'_{i,t}) = 0 \tag{5}$$

$$C2: \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{M} D_{i,t} \le D^{max} \tag{6}$$

$$C3: \sum_{i=1}^{M} b_{i,t} \le B \tag{7}$$

$$C4: 0 \le P_{i,t}^e, P_{i,t}^c, P_{i,t}^e + P_{i,t}^c \le 1 \tag{8}$$

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. $C4$ are several range limits for both $P_{i,t}^e$ and $P_{i,t}^c$. D^{max} in $C2$ represents the result cutoff time per frame. 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. It is NP-hard and cannot be solved in polynomial time. 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 III-B. Accordingly, the variables in the entire problem will only include the linear $P_{i,t}^e$ and $P_{i,t}^c$.

IV. OPTIMIZATION AND ALGORITHM

It is known that Lyapunov theory is widely used in dealing with system stability issues under dynamic network conditions [29]. 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.

A. Drift-Plus-Penalty Expression

To solve the long-term stochastic optimization problem $P1$, we transform the original problem with time-average constraints into a series of tractable, per-time-slot optimization problems. For an effective representation of the constraint, we construct a virtual evolution queue to describe $C2$:

$$D_i(t+1) = \max\{D_i(t) + \frac{D_{i,t} - D^{max}}{D^{max}}, 0\}. \tag{9}$$

Therefore, the constraint problem of $C1$ and $C2$ can be transformed into the stability problem of $Q_i(t)$ and $D_i(t)$. We defined the Lyapunov function $L(H_i(t))$ and Drift$\Delta H_i(t)$ as

$$L(H_i(t)) = \frac{1}{2}(\sum\nolimits_{i=1}^{M} Q_i^2(t) + \sum\nolimits_{i=1}^{M} D_i^2(t)), \tag{10}$$

$$\Delta H_i(t) = \mathbf{E}[L(H_i(t+1)) - L(H_i(t))|H_i(t)]. \tag{11}$$

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. We apply a non-negative constant V to adjust the contribution of accuracy to the problem.

$$P2: \min_{\{P_t^e, P_t^c\}} \Delta H_i(t) + V\mathbf{E}[\frac{1}{\bar{A}_t}|H_i(t)]. \tag{12}$$

B. Optimal Target Transformation

This minimization simultaneously pushes the virtual queues towards stability and drives the utility objective towards optimality, governed by a control parameter V that balances this trade-off. The following Theorem 1 establishes the performance guarantee for this approach, ensuring that the solution to the transformed problem $P2$ is optimal for the primal problem $P1$.

Theorem 1. *In Lyapunov optimization theory, if formula 12 in $P2$ has an upper bound $\chi(\chi > 0)$, the optimal solution A^* of the transformed problem $P2$ is identical to the optimal solution A^* of the primal problem $P1$.*

It is known that the accuracy is always greater than a certain positive number θ (in the case where all frames are only tracked locally). The penalty term $V\mathbf{E}[\frac{1}{\bar{A}_t}|H_i(t)]$ must have an upper bound. The remaining expressions involving $Q_i(t)$ and $D_i(t)$ are also proved to be bounded in the Appendix.

Lemma 1. *Formula 12 in $P2$ has an upper bound means all virtual queues $Q_i(t)$ and $D_i(t)$ are strongly stable. Therefore, long-term constraints of the original problem are satisfied.*

We also prove Lemma 1 in the Appendix. After the constraints were satisfied, we observe that formula 12, which was substituted into the data, only contained two variables $P_{i,t}^e$ and $P_{i,t}^e$. We express this part as:

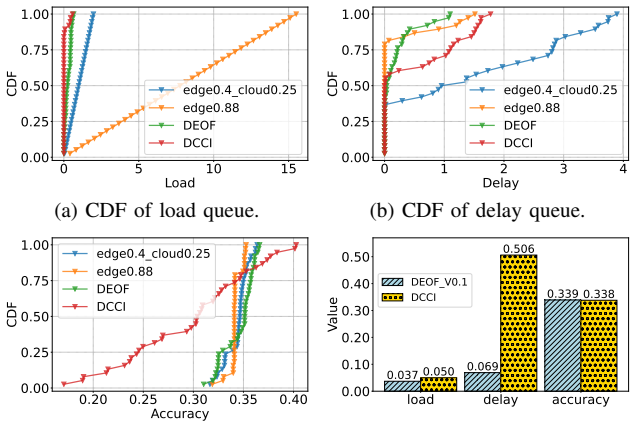
$$\begin{aligned}&(\frac{J}{\frac{l_t}{T_t^e} + \frac{l_t}{T_t^c}})^2(P_{i,t}^e + P_{i,t}^c)^2 + \frac{2J}{\frac{l_t}{T_t^e} + \frac{l_t(Q_i(t)-1)(P_{i,t}^e + P_{i,t}^c)}{T_t^c}} \\ &+ \frac{2J(\frac{P_{i,t}^c}{B_1} + \frac{P_{i,t}^e}{B_2})(D_i(t)-1)}{D^{max}} + (\frac{J}{D^{max}})^2(\frac{P_{i,t}^c}{B_1} + \frac{P_{i,t}^e}{B_2})^2 \\ &+ \frac{V}{\sum_k \left(a_{i,t,k}^e + a_{i,t,k}^c + a_{i,t,k}^l\right)}.\end{aligned} \tag{13}$$

We only need to minimize these parts to obtain the best solution. It is obvious that this is a non-convex optimization problem. In previous works, such as JCAB [30], the reinforcement learning method is adopted to take actions. 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 $P_{i,t}^e$ and $P_{i,t}^c$, we decided to design a heuristic genetic algorithm tailored to our scenario to solve

Algorithm 1 Offloading Proportion Algorithm

Input: Bandwidth $B1$,$B2$, Queue loads $Q_i(t)$, Delays $D_i(t)$, Prediction accuracy list P_{acc}, Edge,Cloud accuracy Φ_t^e,Φ_t^c;
Output: Offloading strategy $F_{i,t}$, Proportions P_i^e, P_i^c;

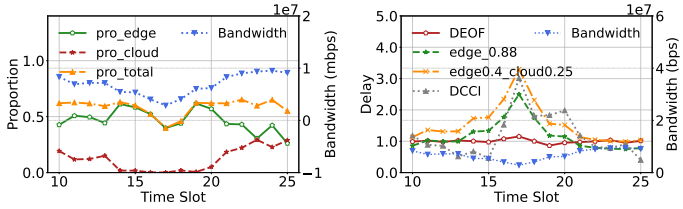
```
Initialize by generating pop_size populations;
for g = 1 to generations do
    Mutate individuals with probability α
    for each individual (P_i^e, P_i^c) do
        F_{i,t} ← [end] × f_0
        P_{acc−sort} ← sorted(enumerate(P_acc))
        for k = 1 to P_i^c × f_0 do
            F_{i,t}[P_{acc−sort}[k]] ← cloud
        end for
        for j = P_i^c × f_0 to (P_i^e + P_i^c) × f_0 do
            F_{i,t}[P_{acc−sort}[j]] ← edge
        end for
        Acc ← [0] × f_0
        for k = 1 to f_0 do
            if F_{i,t}[k] = cloud then
                Acc[k] = Φ_t^c
            else if F_{i,t}[k] = edge then
                Acc[k] = Φ_t^e
            else
                Acc[k] = Acc[k − 1] × e^{−λ/f_0}
            end if
        end for
        Ā_t ← sum(Acc)/f_0
        fitness ← −(formula13)
    end for
    Record best individual
    Crossover with probability β
end for
Obtain optimal F_{i,t}, P_i^e, P_i^c
Update Q_i(t + 1) ← Q_i(t) − Q'_{i,t}
Update D_i(t + 1) ← D_i(t) − D^{max}
return F_{i,t}, P_i^e, P_i^c
```



(a) CDF of load queue. (b) CDF of delay queue.

(c) CDF of accuracy. (d) Comparison of DEOF and DCCI.

Fig. 5: Performance of DEOF compared to other baselines.



(a) Adaptive to proportion. (b) Adaptive to delay.

Fig. 6: The dynamic adaptation of DEOF to bandwidth.

for the approximate optimal value.

C. Algorithm Design

The main algorithm flow is shown in Algorithm 1. Several system-related parameters are stored in the OS and serve as input parameters, including bandwidth, predicted accuracy list, etc. The objective is to obtain the optimal offloading strategy.

Firstly, the population is initialized with the number of individuals specified by pop_size. The number of iterations is determined by $generations$. The main loop proceeds as follows: 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), we sort the P_{acc} in ascending order (a lower element value indicates greater detection difficulty). For each frame, we assign it a sequential label as 'cloud', 'edge', or 'end', in proportion to the frame's actual execution location. Then we calculate the detection accuracy corresponding to each frame in sequence according to its label. In this way, we can obtain $\bar{A}_t$ according to formula 2. We use the objective function in formula 13 and take the negative of this term as the fitness to represent the genetic population's quality.

After evaluating all individuals in the current generation, the best individual is recorded. The next generation is then generated through selection, crossover (applied with probability β), and mutation. This iterative optimization process enables the algorithm to make accurate offloading decisions for each video frame based on the system's processing capability.

V. EVALUATION

A. Experimental setup

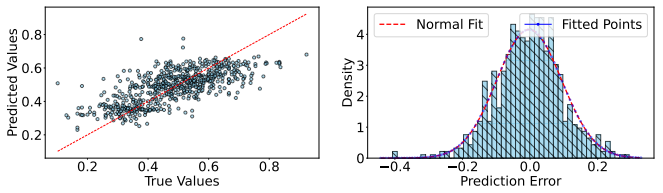
We select continuous sequences of frames from two benchmark datasets: UA-DETRAC [23] and VisDrone2019 [24]. UA-DETRAC provides 100 challenging video sequences from real-world traffic scenes for vehicle detection and tracking evaluation, while VisDrone2019 offers a comprehensive benchmark for object detection in drone-captured aerial imagery. We deploy YOLOv7 as a heavyweight model on the cloud server, which is equipped with two 3090 GPUs dedicated to inference. Simultaneously, the edge server, which is equipped with a 1660ti GPU, hosts the EdgeYOLO model, designed as a computationally efficient and lightweight alternative for deployment. Moreover, to achieve target tracking on the local device, we use optical flow. Ultimately, we developed a Flask-based backend to realize the DEOF algorithm, which dynamically dispatches video frames to the computation locations specified by the optimal scheduling strategies for remote inference.

Since the dataset does not annotate non-motorized vehicles and includes some ignored regions, the detection results show high recall but low precision. Therefore, we use the F1 score as the evaluation metric for detection. Through multiple simulations run, the average F1 scores for the two models in this scenario are 0.407 and 0.355, respectively. Furthermore, we compare DEOF with three other benchmarks in the simulation.

- **Fixed edge offloading proportion**: We set the proportion of offloaded frames to 0.88 to ensure that the final achieved accuracy is comparable. That means about 26 frames are selected and offloaded within the 1-second time slot. Furthermore, these frames are evenly distributed throughout the time slot to maximize accuracy.
- **Fixed edge and cloud offloading proportion**: To demonstrate the dynamic adaptability of our system to various factors such as load and bandwidth, we also implement the method of fixed edge and cloud offloading proportion simultaneously. Two proportions are set to 0.4 and 0.25.
- **DCCI**: DCCI [17] trains discriminators to classify images as hard or simple cases, deploying multiple discriminators for different bandwidths and loads. We adopt settings from the original article: four discriminators with IoUs of 0.2, 0.5, 0.6, and 0.7, applied in different bandwidth intervals. Hard cases are offloaded to the cloud, while simple ones are processed locally for tracking.

B. Baseline comparison

This part shows the advantages of DEOF over other algorithms. We implemented the three other algorithms in the same scenario. Fig.5a - 5c shows the distribution of load queue, delay queue, and accuracy values of DEOF and other algorithms under the same video clip. We can see that DEOF achieves the best performance except for the load queue, which is second only to DCCI. Since DCCI only opts to offload some frames to the cloud, it improves precision but incurs significant latency. To make the comparison more relevant, we conducted the experiment with V=0.1. As shown in Fig.5d, each item of DEOF achieves better values than DCCI because we relax



(a) AP: True vs Predicted. (b) Distribution of Prediction Errors.

Fig. 7: Prediction distribution of Accuracy-Predictor.

TABLE II: Counts of optimal mutation and crossover rate

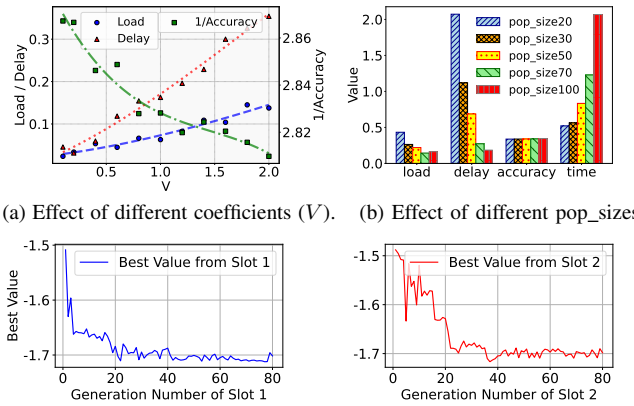
count	β					α				
	0.01	0.03	0.07	0.1	0.2	0.5	0.6	0.7	0.8	0.9
load	0	2	5	17	29	14	8	13	11	4
delay	0	1	5	16	26	8	12	12	12	6
accuracy	20	2	0	11	17	9	13	18	6	4

the accuracy constraint to prioritize minimizing overall load and delay. Besides, we can see that although the edge-only approach achieves similar accuracy to DEOF, it chooses to offload most of the frames, resulting in lower system efficiency. In addition, for the fixed edge-cloud approach, it can be observed that this method is inferior to DEOF across all metrics because it cannot dynamically adjust the offloading amount according to bandwidth and system load. It is worth noting that we selected 33 consecutive time slots (33-second test video) from VisDrone2019, and by monitoring method calls, we measured that the Lyapunov algorithm averages only 0.1008 seconds per time slot, which does not compromise the real-time performance of video analytics.

We also demonstrate the bandwidth adaptability of DEOF by simulating another fluctuating bandwidth trajectory. As shown in Fig.6a, the bandwidth shows a significant downward trend after the 10th time slot. The offloading proportion starts to decrease when the continuous bandwidth decline impacts the overall delay. Specifically, DEOF offloads fewer frames to the cloud because the bandwidth between local devices and the cloud is relatively lower. Meanwhile, it moderately increases the proportion of frames offloaded to the edge server to minimize accuracy loss. This maximizes the delay constraint satisfaction from Fig.6b. Turning to other methods, they are unable to adjust the offloading data volume effectively, which results in significant delay fluctuations.

C. Comparison of parameters

We first evaluate the SVR-based AP model (introduced in Section III-A) using Mean Squared Error, which achieves a satisfactorily low value (0.00922). As shown in Fig.7, the predicted result graph effectively reflects that AP has the ability to accurately predict. The predicted F1 score is designed to reflect the complexity and variation of each image. Distinct predicted F1 scores are employed to quantify the detection difficulty of individual frames in practical deployment, thereby enabling an adaptive execution strategy for each frame. Therefore, the resulting deviations in F1 remain within an acceptable tolerance range. Secondly, we consider the size of the genetic population. Fig.8b illustrates the behavior of the load queue, delay queue, accuracy, and execution time for population sizes



(a) Effect of different coefficients (V). (b) Effect of different pop_sizes.

(c) Iterations required for convergence under two segments.

Fig. 8: The Impact of Certain Parameters on Algorithm Executions.

of [20, 30, 50, 70, 100]. A larger population size enables the algorithm to explore a larger solution space, potentially leading to better performance. However, to avoid the algorithm taking too long to execute, we chose a population size of 50 as an optimal trade-off.

Thirdly, we simulated the iterative process of two different time slots in this scenario. As shown in Fig.8c, although the iterations required for convergence differ between the segments, it can be observed that the algorithm tends to stabilize after 50 iterations. Therefore, we selected 50 as the final number of iterations for our genetic algorithm. We optimized the crossover rate α and mutation rate β via a grid search. The results of the repeated experiments of 50 groups illustrate that the best-performing combination was α=0.7 and β=0.2, which most frequently achieved the minimum latency and load, and the maximum accuracy.

Finally, when solving the final decision variable by genetic algorithm, we found that the penalty coefficient V allows for trade-offs between load, latency, and accuracy. As illustrated in Fig.8a. A larger V indicates a stronger constraint on accuracy, prompting the algorithm to prioritize higher and more stable accuracy at the cost of relaxed constraints on load and latency. In the end, we use 1 as the penalty coefficient. In addition, the frame processing deadline D^{max}, which reflects the system's delay tolerance, also impacts performance. A higher D^{max} allows more frames to be offloaded, improving overall accuracy and system stability. After experimenting with different values, D^{max}=1.5 seconds was selected under the simulated bandwidth trajectories to ensure real-time performance.

VI. CONCLUSIOM

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. We use Lyapunov optimization theory to model DEOF 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.

APPENDIX

Proof of upper bound:

Proof: $(Q_i(t+1))^2 - (Q_i(t))^2$

$$= (\max\{Q_i(t) + \frac{Q_{i,t}-Q'_{i,t}}{Q'_{i,t}}, 0\})^2 - (Q_i(t))^2$$
$$\leq (\frac{Q_{i,t}-Q'_{i,t}}{Q'_{i,t}})^2 + 2(\frac{Q_{i,t}-Q'_{i,t}}{Q'_{i,t}})Q_i(t)$$
$$= (\frac{f_0 l_t \rho r_{i,t}^2 (P_{i,t}^e + P_{i,t}^c)}{Q'_{i,t}})^2$$
$$+2(\frac{f_0 \rho r_{i,t}^2 (P_{i,t}^e + P_{i,t}^c)}{Q'_{i,t}})(Q_i(t)-1) + 1 - 2Q_i(t). \quad \blacksquare$$

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.

Proof: $\sum_{i=1}^{M}[(Q_i(t+1))^2 - (Q_i(t))^2]$
$$\leq [(\frac{J}{Q'_{i,t}})^2 + 2\frac{J(Q_i(t)-1)}{Q'_{i,t}}] \sum_{i=1}^{M}(P_{i,t}^e + P_{i,t}^c) + MS_1$$
$$\leq M[(\frac{J}{Q'_{i,t}})^2 + 2\frac{J(Q_i(t)-1)}{Q'_{i,t}}] + MS_1. \quad \blacksquare$$

Similar proofs process can be used for the delay queue, and we use S_2 instead of $1 - 2D_i(t)$. Through $C4$ and $C5$, the total delay queue drift for multiple devices.

Proof: $\sum_{i=1}^{M}[(D_i(t+1))^2 - (D_i(t))^2]$
$$\leq M[(\frac{J(B_1+B_2)}{D^{max}B_1B_2})^2 + 2\frac{J}{D^{max}}(\frac{B_1+B_2}{B_1B_2})(D_i(t)-1) + S_2]. \quad \blacksquare$$

Proof of Lemma 1:

Proof: $L(H_i(t+1)) - L(H_i(t)) + V(\frac{1}{A_t}) \leq \chi$

$$L(H_i(t+1)) - L(H_i(t)) \leq \chi$$
$$L(H_i(T)) - L(H_i(0)) \leq T\chi$$
$$L(H_i(0)) \geq 0 : L(H_i(T)) \leq T\chi$$
$$0 \leq Q_i(t), D_i(t) \leq \sqrt{T\chi}$$
$$\lim_{T\to\infty} \frac{Q_i(t)}{T} = \lim_{T\to\infty} \frac{D_i(t)}{T} = 0. \quad \blacksquare$$

REFERENCES

[1] L. Yuan, C. Xiong, S. Chen, and W. Gong, "Embracing self-powered wireless wearables for smart healthcare," in *Proceedings of IEEE PerCom*, 2021, pp. 1–7.

[2] N. Chen, S. Quan, S. Zhang, Z. Qian, Y. Jin, J. Wu, W. Li, and S. Lu, "Cuttlefish: Neural configuration adaptation for video analysis in live augmented reality," *IEEE TPDS*, vol. 32, no. 4, pp. 830–841, 2020.

[3] N. Chen, S. Zhang, Z. Ma, Y. Chen, Y. Jin, J. Wu, Z. Qian, Y. Liang, and S. Lu, "Vichaser: Chase your viewpoint for live video streaming with block-oriented super-resolution," *IEEE/ACM TON*, vol. 32, no. 1, pp. 445–459, 2024.

[4] J. Li, Z. Li, R. Lu, K. Xiao, S. Li, J. Chen, J. Yang, C. Zong, A. Chen, Q. Wu *et al.*, "Livenet: a low-latency video transport network for large-scale live streaming," in *Proceedings of ACM SIGCOMM*, 2022, pp. 812–825.

[5] J. Zhao, J. Liu, C. Zhang, Y. Cui, Y. Jiang, and W. Gong, "Mptcp+: Enhancing adaptive http video streaming over multipath," in *Proceedings of IEEE/ACM IWQoS*, 2020, pp. 1–6.

[6] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE IoT-J*, vol. 7, no. 8, pp. 7457–7469, 2020.

[7] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of ACM SIGCOMM*, 2020, pp. 557–570.

[8] F. Du, P. Liu, W. Zhao, and X. Tang, "Correlation-guided attention for corner detection based visual tracking," in *Proceedings of IEEE CVPR*, 2020, pp. 6836–6845.

[9] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of ACM SIGCOMM*, 2018, pp. 253–266.

[10] N. Chen, S. Zhang, Y. Liang, J. Wu, Y. Chen, Y. Yan, Z. Qian, and S. Lu, "Tilesr: Accelerate on-device super-resolution with parallel offloading in tile granularity," in *IEEE INFOCOM*, 2024, pp. 2538–2547.

[11] T. Murad, A. Nguyen, and Z. Yan, "Dao: Dynamic adaptive offloading for video analytics," in *Proceedings of ACM MM*, 2022, pp. 3017–3025.

[12] A. Ghosh, S. Iyengar, S. Lee, A. Rathore, and V. N. Padmanabhan, "React: streaming video analytics on the edge with asynchronous cloud support," in *Proceedings of ACM/IEEE IoTDI*, 2023, pp. 222–235.

[13] C. Kai, H. Zhou, Y. Yi, and W. Huang, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *Proceedings of IEEE TCCN*, vol. 7, no. 2, pp. 624–634, 2020.

[14] X. Wang, G. Gao, X. Wu, Y. Lyu, and W. Wu, "Dynamic dnn model selection and inference offloading for video analytics with edge-cloud collaboration," in *ACM NOSSDAV*, 2022, pp. 64–70.

[15] S. N. Gowda, M. Rohrbach, and L. Sevilla-Lara, "Smart frame selection for action recognition," in *Proceedings of the AAAI*, vol. 35, no. 2, 2021, pp. 1451–1459.

[16] S. Cen, M. Zhang, Y. Zhu, and J. Liu, "Adadsr: Adaptive configuration optimization for neural enhanced video analytics streaming," *IEEE IoT-J*, 2023.

[17] Y. Hu, Z. Li, Y. Chen, Y. Cheng, Z. Cao, and J. Liu, "Content-aware adaptive device–cloud collaborative inference for object detection," *IEEE IoT-J*, vol. 10, no. 21, pp. 19 087–19 101, 2023.

[18] N. Chen, S. Zhang, S. Zhang, Y. Yan, Y. Chen, and S. Lu, "Resmap: Exploiting sparse residual feature map for accelerating cross-edge video analytics," in *IEEE INFOCOM*, 2023, pp. 1–10.

[19] N. Chen, S. Zhang, J. Wu, H. Huang, and S. Lu, "Spliceosome: On-camera video thinning and tuning for timely and accurate analytics," *IEEE/ACM TON*, vol. 33, no. 3, pp. 1252–1264, 2025.

[20] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *arXiv preprint arXiv:1703.02529*, 2017.

[21] M. Zhang, F. Wang, and J. Liu, "Casva: Configuration-adaptive streaming for live video analytics," in *Proceedings of IEEE INFOCOM*, 2022, pp. 2168–2177.

[22] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proceedings of IEEE INFOCOM*, 2018, pp. 1421–1429.

[23] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, and S. Lyu, "Ua-detrac: A new benchmark and protocol for multi-object detection and tracking," *CVIU*, vol. 193, p. 102907, 2020.

[24] 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. 0–0.

[25] S. Liang, H. Wu, L. Zhen, Q. Hua, S. Garg, G. Kaddoum, M. M. Hassan, and K. Yu, "Edge yolo: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles," *IEEE T-ITS*, vol. 23, no. 12, pp. 25 345–25 360, 2022.

[26] A. C. Sparavigna, "Entropy in image analysis," *Entropy*, vol. 21, no. 5, p. 502, 2019.

[27] F. Zhang and L. J. O'Donnell, "Support vector regression," in *Machine learning*. Elsevier, 2020, pp. 123–140.

[28] J. Xue and Y. An, "Joint task offloading and resource allocation for multi-task multi-server noma-mec networks," *IEEE ACCESS*, vol. 9, pp. 16 152–16 163, 2021.

[29] Z. Tong, J. Cai, J. Mei, K. Li, and K. Li, "Dynamic energy-saving offloading strategy guided by lyapunov optimization for iot devices," *IEEE IoT-J*, vol. 9, no. 20, pp. 19 903–19 915, 2022.

[30] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proceedings of IEEE INFOCOM*, 2020, pp. 257–266.