Edge-Cloud Collaborated Object Detection via Bandwidth Adaptive Difficult-Case Discriminator

Zhiqiang Cao, *Student Member, IEEE,* Yun Cheng, Zimu Zhou, Yongrui Chen *Member, IEEE,* Youbing Hu, Anqi Lu, *Student Member, IEEE,* Jie Liu, *Fellow, IEEE,* Zhijun Li, *Member, IEEE*

Abstract—Object detection, a fundamental task in computer vision, is crucial for various intelligent edge computing applications. However, object detection algorithms are usually heavy in computation, hindering their deployments on resource-constrained edge devices. Traditional edge-cloud collaboration schemes, like deep neural network (DNN) partitioning across edge and cloud, are unfit for object detection due to the significant communication costs incurred by the large size of intermediate results. To this end, we propose a Difficult-Case based Small-Big model (DCSB) framework. It employs a difficult-case discriminator on the edge device to control data transfer between the small model on the edge and the large model in the cloud. We also adopt regional sampling to further reduce the bandwidth consumption and create a discriminator zoo to accommodate the varying networking conditions. Additionally, we extend DCSB to video tasks by developing an adaptive sampling rate update algorithm, aiming to minimize computational demands without sacrificing detection accuracy. Extensive experiments show that DCSB can detect 97.26%-97.96% objects while saving 74.37%-82.23% network bandwidth, compared to cloud-only methods. Furthermore, DCSB significantly outperforms the latest DNN partitioning methods, reducing inference time by 92.60%-95.10% given an 8Mbps transmission bandwidth. In video tasks, DCSB matches the detection accuracy of leading video analysis methods while cutting the computational overhead by 40%.

Index Terms—Object detection; edge-cloud collaboration; neural networks; small-big model; difficult-case discriminator

1 Introduction

Deep neural network (DNN) based object detection is extensively utilized in various intelligent applications owning to its remarkable performance [1], [2], [3], [4]. However, the computational demand of DNN-based object detection impedes its adoption on resource-limited edge devices. This limitation is particularly critical in safety-sensitive mobile applications, such as autonomous driving [5] and safety monitoring systems [6], where accurate and low-latency object detection is essential.

Model compression holds potential for efficient object detection on edge devices [7], [8]. However, these compressed models often incur accuracy drop, which is unsuitable for applications requiring high precision. An alternative is to cloud offloading, where data is transmitted to the cloud for inference on powerful DNNs, and the results are returned to edge devices (middle of Fig. 1) [9], [10], [11], [12]. However, cloud offloading incurs significant latency and bandwidth usage due to the necessity of uploading large data volumes over the wide-area network (WAN). As the number of edge devices connected to the network grows explosively, and the data volume transferred also increases dramatically [13], the bandwidth may fail to meet the requirements of individual devices. Moreover, the operational costs and logistical challenges of maintaining network connections for devices in remote locations *e.g.*, surveillance cameras [14], coupled with fluctuating network conditions, underscore the critical need for bandwidth conservation.

A more promising strategy is collaborative inference between the cloud and the edge (left of Fig. 1) [15], [16], [17], [18]. It partitions the DNN inference workload between the cloud and the edge, with the division optimized against metrics such as bandwidth usage, processing delay, and data output size. During DNN inference, the edge device processes first part of the DNN and forwards the intermediate data to the cloud. The cloud continues the execution of the remaining part and returns the result to the edge device.

Despite edge-cloud collaboration frameworks designed for image classification, their applicability to object detection can be challenging. *(i)* Partitioning models for object detection often results in the transmission of large intermediate outputs, leading to increased communication overhead and inference delays [19]. For instance, studies [20] show that models like ResNet152 [21] generate outputs 19 to 4500 times larger than the input video itself. *(ii)* Many schemes are dedicated to image classification, *e.g.*, instance-wise adaptive networks [22], which are not directly applicable to objective detection. This is because object detection involves analyzing images with multiple objects of diverse difficulty levels, which complicates the metrics to assess whether an input instance is difficult or not [22], [23]. *(iii)* The system's

- *Z. Cao, Y. Hu, and A. Lu are with the School of Computer Science and Technology, Harbin Institute of Technology, 150006, Harbin, China. (E-mail: {zhiqiang_cao,youbing,luanqi}@stu.hit.edu.cn)*
- *Y. Cheng is with Swiss Data Science Center, 8000 Zurich, Switzerland. (E-mail: yun.cheng@sdsc.ethz.ch)*
- *Y. Chen is with the Department of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing, China. (E-mail: chenyr@ucas.ac.cn)*
- *Z. Zhou is with the School of Data Science, City University of Hong Kong, Hong Kong, China. (E-mail: zimuzhou@cityu.edu.hk)*
- *J. Liu and Z. Li are with the Faculty of Computing, Harbin Institute of Technology, 150006, Harbin, China. (E-mail:{lizhijun_os,jieliu}@hit.edu.cn)*

This work is partly supposed by the National Key R&D Program of China under Grant NO.2023YFB4503100. This work is also partly supported by NSFC 62072137. Zimu Zhou's research is supported by CityU APRC grant (No. 9610633). (Corresponding author: Zhijun Li,Yun Cheng)

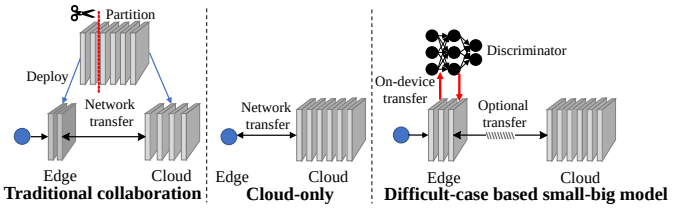


Fig. 1. Comparison between our DCSB and existing methods.

robustness is questionable since abrupt changes in networking conditions can easily cause system failures [24].

To this end, we propose a novel Difficult-Case based Small-Big Model framework (DCSB) for object detection (right of Fig. 1). It involves multiple innovations. *(i) Lightweight Difficult-Case Discriminator*: Deployed at the edge, this module efficiently categorizes incoming images as either "easy cases" or "difficult cases". Only difficult cases are forwarded to the cloud for processing by a more capable, heavyweight model to ensure high detection accuracy, while easy cases are handled locally by a less resource-intensive model, minimizing communication overhead. *(ii) Regional Sampling Algorithm*: It selectively down-samples portions of difficult-case images based on preliminary analysis by the edge-deployed lightweight model, thus reducing the volume of data that must be transmitted to the cloud without significantly compromising the information necessary for accurate object detection. *(iii) Discriminator Zoo*: Recognizing the variability of networking conditions, we propose a mechanism to dynamically select the most appropriate discriminator from a zoo of options. This adaptability ensures optimal performance under fluctuating network conditions. *(iv) Handling Video Input*: We extend DCSB to video tasks, with an algorithm that dynamically adjusts sampling rates. This adaptation balances computational demands with the need for maintaining detection accuracy across video frames. Our evaluation demonstrate that DCSB can save 74.37%-82.23% bandwidth consumption while detecting 97.26%-97.96% of objects. Our main contributions are summarized as follows.

- We introduce a novel framework for edge-cloud collaborative DNN inference. Conventional solutions often distribute a DNN as a computation graph between the device and cloud. In contrast, we strategically place a compact model at the edge and a more powerful model in the cloud, utilizing a novel difficult-case discriminator to selectively transmit data. It enables more bandwidth-efficient data transfers between edge devices and the cloud.
- We systematically address multiple critical challenges to implement DCSB, including identification of difficult cases, design of a lightweight difficult-case discriminator that leverages specific semantics extracted by the small model at the edge, and reduction of data transfer volumes without compromising object detection accuracy.
- We evaluate DCSB across three object detection algorithms, four datasets and two edge devices. The results are compelling. Data transmission is reduced by 74.37%-82.23% compared to cloud-only methods, while still detecting 97.26%-97.96% of objects. Mean Average Precision (mAP) improves by 15.97%-17.61% over edge-only methods. When compared to the state-of-the-art model partitioning method, CAS [25], our framework demonstrates a significant reduction in inference time by 92.60%-95.10% at given a fixed 8Mbps transmission bandwidth.

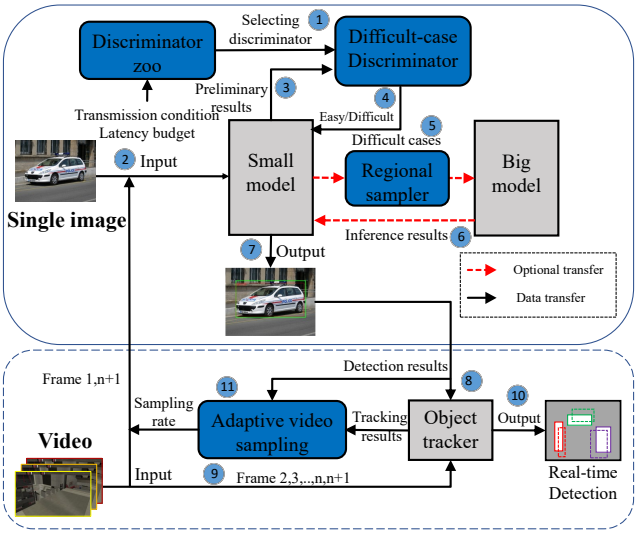


Fig. 2. DCSB architecture and workflow. Blocks in blue are the main functional modules. Dashed boxes are extra modules for video input.

A preliminary version of this work is presented in [26]. This paper makes the following additional contributions.

- We replace its simplistic threshold-based discriminator design by a neural network of fully connected layers enhanced with an attention module. This redesign offers improved accuracy and robustness in identifying difficult cases.
- We add a novel discriminator zoo module, which dynamically selects an optimal difficult-case discriminator based on the networking conditions and latency budget. This module ensures optimal performance across varying networking environments.
- We extend DCSB to video tasks with an algorithm to adaptively adjust sampling rates to minimize computational and communication demands without sacrificing detection accuracy. When benchmarked against EdgeDuet [27], the state-of-the-art in video analysis, we achieve comparable detection accuracy while notably reducing the overhead by 40%.

2 System Overview

Architecture. The basic idea of DCSB is to offload images that the small, edge-based model cannot recognize accurately, to a big model in the cloud for processing. It works with both single images and videos as input. Note that DCSB is agnostic to the architectures of the small and big model. Fig. 2 illustrates the main functional modules.

- *Difficult-case Discriminator* (Sec. 3). This module determines whether an image can be processed locally (an *easy* case) or must be sent to the cloud (a *difficult*

case). We replace the static threshold based design in our preliminary version [26] with an attention based mechanism to improve the decision accuracy and robustness to varying bandwidth.
- *Regional Sampler* (Sec. 4). This module adaptively down-samples regions of an image based on preliminary analysis by the small model, reducing the amount of data to be sent without notably compromising detection accuracy.
- *Discriminator Zoo* (Sec. 5). This module dynamically selects the most suitable difficult-case discriminator based on networking conditions and the latency budget. It houses various discriminators, each tuned to specific scenarios.
- *Object Tracking* (Sec. 6.1). For video inputs, this module continuously tracks objects across frames, leveraging the latest object detection results.
- *Adaptation Video Sampling* (Sec. 6.2). This module adjusts the video frame sampling rate based on object tracking and detection outcomes, optimizing computational and communication efficiency without compromising on detection quality.

Workflow. DCSB is designed to efficiently handle both single images and video streams. We outline the distinct steps for processing single images and videos below.

- **Single Image Processing**. ① DCSB selects the most suitable difficult-case discriminator from the discriminator zoo based on current networking conditions and latency constraints. This selection is updated periodically *e.g.*, hourly to adapt to changing networking conditions. ② The edge device captures an image and forwards it to the small model for preliminary recognition. ③ The small model's initial detection results are fed into the difficult-case discriminator. ④ The difficult-case discriminator evaluates the complexity of the case (easy or difficult), and returns it to the small model. ⑤ For a difficult case, the edge device compresses the image by the regional sampler, and uploads the image to the cloud, where the big model performs more accurate object detection. ⑥ The big model's results are sent back to the edge device for final processing. ⑦ DCSB aggregates the results from both models for a difficult case or solely relies on the small model's output for an easy case as the final output. In summary, the workflow for an easy case follows steps ②-③-④-⑦, and a difficult case proceeds through steps ②-⑦.
- **Video Processing**. For video inputs, DCSB incorporates two additional modules (dashed box in Fig. 2). The workflow for processing video frames mirrors that of single images, treating each sampled frame as an individual image, *i.e.*, following ①-⑦. ⑧ The detection results are fed into the object tracker. ⑨ All frames except the initial one are also sent to the object tracker. ⑩ The object tracker tracks the detected objects based on the latest detection results and outputs the object positions and identities. ⑪ DCSB determines the next sampling rate based on the detection and tracking results of the current sampled frame. Note that the processes of detection and tracking operate in parallel. The tracker detects objects in each frame in real-time, while the object detection models run on a small subset of sampled frames to update the tracking objects.

3 Difficult-Case Discriminator

The difficult-case discriminator is a core module in DCSB, which regulates the communication between the small model on edge devices and the big model in the cloud. It decides the need for engaging the big model's processing capabilities for accurate object detection.

3.1 Defining Difficult Cases

A prerequisite is to define the difficult cases, which is more challenging in object detection than image classification. In image classification, difficult cases can be assessed via the entropy of the classifier output [22]. In object detection, the difficulty relates to the *undetected* objects, which cannot be easily measured from the output of a single model. Therefore, we define the difficult cases by comparing the output between the big and small models. For a given image i, let the number of objects detected by the small model be n_{i}, and by the big model be N_{i}. We classify the image as a difficult case if the difference in the number of detected objects $D_{\mathrm{i}} = N_{\mathrm{i}} - n_{\mathrm{i}}$ surpasses a predefined threshold $\alpha > 0$. For instance, $\alpha = 1$ implies an image is a difficult case if at least one object detected by the big model is missed by the small model. This criterion allows easy labeling of training data for the difficult-case discriminator by comparing the detection results between the small and big models. Only detection boxes with a confidence score above 0.5 are considered as valid detection, as in standard object detection benchmarks [28].

3.2 Indicators for Difficult Cases

Since it is infeasible to compare the outputs between two models to determine whether an image is difficult, we need a discriminator design without reliance on the big model. Directly training a binary classifier on the raw images is feasible, but may result in a discriminator too large for frequent execution on edge devices. Thus, we resort to lightweight indicators as inputs for the difficult-case discriminator. Our analysis of the small model's detection outcomes shows that its performance degrades when handling images with multiple objects or small objects, attributing to the reduced complexity of its convolutional network and the corresponding feature maps. Hence, we focus on two indicators to classify difficult cases: *number* of objects in an image and the *area ratio* of objects (*i.e.*, an object's area relative to the total image area).

We validate the effectiveness of these two indicators as follows. We set α to 1, *i.e.*, an image is difficult if the small model misses at least one object. Then the object with the smallest area ratio is the most likely to be missed. Hence we can use the number of objects and the proportion of the smallest object area as two indicators to assess whether an image is difficult when $\alpha = 1$. Then we train a small model (MobileNet v1-YOLOv4 [29]) and a big model (YOLOv4 [30]) on VOC07 [28], and calculate the two metrics for each

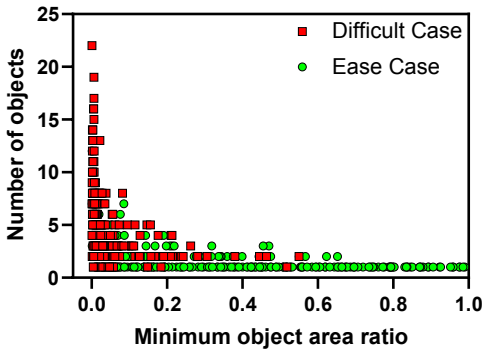


Fig. 3. Distribution of easy and difficult cases identified by two features-the number of objects and the minimum object area ratio, for $\alpha = 1$.

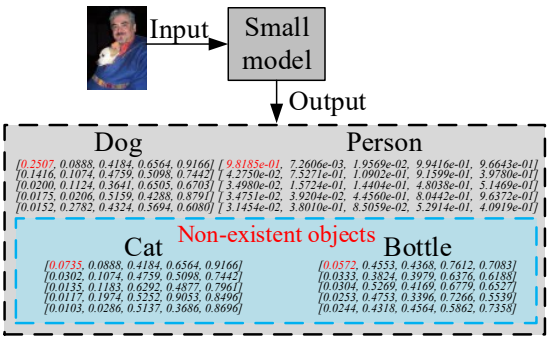


Fig. 4. Confidence scores (in red) of detected and non-existent objects.

image. Finally, we label each image as either difficult or easy as the difference between the number of objects detected by the two models, *i.e.*, the definition in Sec. 3.1. As shown in Fig. 3, difficult cases are concentrated in areas with a large number of objects or a small minimum object area ratio *i.e.*, the left and upper parts in Fig. 3), which validates the effectiveness of the two indicators, *i.e.*, the number of objects and the area ratio of objects.

3.3 Discriminator Design

The difficult-case discriminator takes an image's the detection results by the small model as input, and categorizes the image as either difficult or easy.

3.3.1 Feature Extraction

Although the number of objects and the area ratio of objects are effective in classifying difficult cases (Sec. 3.2), we *calibrate* the bounding boxes before using them to extract the corresponding features. This is because the detection results of the small model often have confidence lower than 0.5, and would be considered invalid. However, this can lead to limited number of features for the discriminator.

To provide more features to the discriminator, we use a data-driven threshold T_{c} rather than the fixed 0.5 to filter invalid bounding boxes. This is because although the confidence of objects detected by the small model might be lower than 0.5, it is still higher than an non-existent object. As shown in Fig. 4, a bounding box contains 5 elements: confidence score, $x_{\min}$, $y_{\min}$, $x_{\max}$, $y_{\max}$. The image contains two objects. If we filter invalid bounding boxes via a confidence threshold of 0.5, the dog (confidence of 0.2507) will be ignored. However, the confidence is still remarkably higher than any object class that does not exist in the image, like 'cat' (0.0735) or 'bottle' (0.0572). Therefore, the fixed threshold of 0.5 is too conservative. Instead, we set a threshold T_{c} for filtering invalid boxes through linear regression with the following loss function.

$$L = N^{\mathrm{predict}} - N^{\mathrm{truth}} \tag{1}$$

where N^{predict} is the total number of objects detected (*i.e.*, with confidence no lower than T_{c}) by the small model, and N^{truth} is the total number of objects from the ground truth.

After filtering invalid bounding boxes based on T_{c}, we can get the number of objects and corresponding area ratios for each detected object in the image.

3.3.2 Discriminator Architecture

Given the number of detected objects N and the corresponding object area ratios $\{S_{\mathrm{k}}\}$ from the small model, the discriminator decides whether the image is a difficult case with a lightweight full connected network (FCN) (see Fig. 5). Since the number of objects varies across images, we standardize the input dimensions via zero-padding, *i.e.*, $[N, S_1, \ldots, S_{\mathrm{N}}, 0, \ldots, 0]$, with the input dimension equal to the maximum object number in the dataset plus one. We implement the discriminator with a positional attention module, four linear modules (Linear + BatchNorm1d + ReLU), and an output module (Linear + Sigmoid), with a maximum data dimension of 300. We introduce the positional attention to enhance the FCN's feature learning capability [31], since naive FCNs treat each input dimension equally. In contrast, we harness a positional attention module consisting of two-layer bottleneck (dimension reduction and increase) and a Sigmoid layer to dynamically assign weights to input dimensions. We empirically set the reduction ratio to 3. This design allows the discriminator to prioritize the connection between features and results, without extra efforts to determine the importance of input features.

4 Regional Sampler

Before uploading a difficult image to the cloud, the regional sampler compresses it by down-sampling non-key regions, thus reducing the bandwidth costs.

4.1 Identifying Key and Non-Key Regions

A prerequisite for the regional sampler is to identify *key regions*, which needed to be processed by the big model in the cloud. In contract, there are *non-key regions* that are either irrelevant to object detection *e.g.*, background, or can already be correctly detected by the small model, *e.g.*, large objects. We can safely down-sample these regions in the image without compromising the overall detection accuracy.

In DCSB, we identify key regions based on the inference results of the small model, which can be classified into three types. Set A: regions that can be clearly analyzed on the

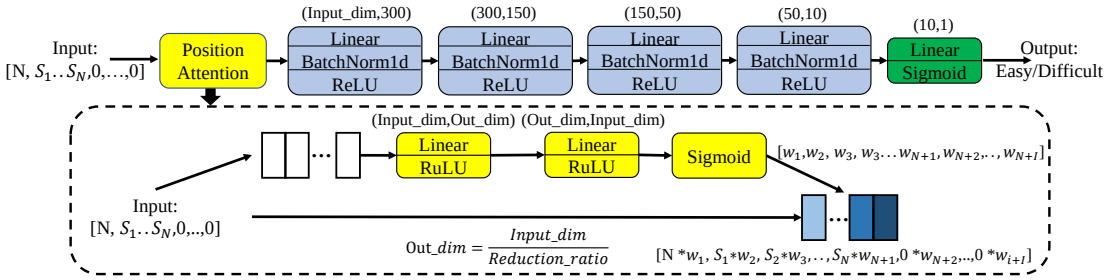
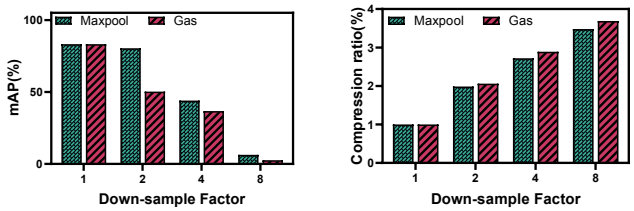


Fig. 5. Illustration of the discriminator architecture.

Fig. 6. An example of regional down-sampling.



(a) mAP (b) Compression ratio

Fig. 7. The mAP and compression ratio of two down-sampling methods. Down-sampling factor of 1 means the original image.

small model (*e.g.*, large objects); Set B: regions where objects may exist but not for sure (*e.g.*, small objects); and Set C: irrelevant regions (*e.g.*, background). The key regions are subset of Set B excluding those in Set A, *i.e.*, $B - A$.

Recall that the inference results of the small model are a list of elements (labeled bounding boxes in object detection), each associated with its coordinates and confidence. We classify a region to Set A if the objects are detected with high confidence. Aligned with the conventions [28], we consider 0.5 as the threshold for a high confidence score. For regions with objects detected with a reasonable confidence, *i.e.*, an object detected but the object class might be inaccurate, we consider these regions as Set B. We can reuse the confidence threshold T_c in Sec. 3.3.1, which is trained to accurately estimate the number of objects. All other regions are classified to Set C. Finally, the key regions are those in $B - A$.

4.2 Non-Key Regions Down-sampling

Given the key and non-key regions in a difficult-case image, we *down-sample* the non-key regions rather than removing them because the objects may overlap, and the semantic relationship between objects may affect the detection accuracy [19]. Fig. 6 shows an example of down-sampled image.

- **Down-Sampling Method**. We empirically evaluate the data compression ratio and the mAP of two widely adopted down-sampling methods, *i.e.*, Maxpool [32] and Gaussian Pyramid [33]. As shown in Fig. 7, given the same down-sampling factor, Maxpool achieves higher accuracy (in mAP) than Gaussian Pyramid. Therefore, we adopt Maxpool in DCSB.
- **Down-Sampling Factor**. We set the down-sampling factors[1] dynamically based on the confidence scores of the non-key regions (the down-sampling factors for key regions are 1). We empirically set the down-sampling factor for non-key regions with a confidence higher than 0.95 to 8, a confidence between 0.85 and 0.95 to 4, and lower confidences to 2. For regions that overlap, the lowest down-sampling factor is utilized for all regions.

After down-sampling different regions within the image, it is then uploaded to the cloud for further processing. When the output of the big model is returned to the edge device, we remove duplicate objects between the inference results of the big model and the small model using Non-Maximum Suppression (NMS) [34], and generate the final results.

5 Discriminator Zoo

The discriminator zoo is designed to dynamically select the most suitable discriminator based on the available latency budget and transmission conditions, optimizing both efficiency and accuracy.

5.1 Discriminator Generation

Generating a variety of discriminators is a one-off effort that is conducted offline. It involves annotating multiple datasets under varying definitions of difficult cases and using these

1. The down-sampling factor is a positive integer. A down-sampling factor of K means we down-sample both the length and width of the region to $1/K$.

datasets to train various discriminators. Specifically, we first input the images in the training dataset into both the large and small models and collect their inference results. We then adjust the threshold α to set varying levels of difficult-case criteria. For different difficult-case criteria, we compare the inference results of the big and small models and annotate the results of the small model. For instance, if the small model fails to detect two or more objects compared to the large model when $\alpha = 2$, the image is marked as a difficult case; and it is an easy case otherwise. This method allows us to create diverse training sets, which are then used to train multiple difficult-case discriminators, denoted as $DC_1, \ldots, DC_{\mathrm{n}}$, with with their corresponding data upload ratios (the fraction of data sent relative to the total dataset) as $U_1, \ldots, U_{\mathrm{n}}$. These discriminators share the same architecture, but differ in model parameters. We also record metrics such as the average inference times for both the small and large models (T_{s} and T_{b}), the average amount of data transmitted post-regional sampling (D), the average data size of inference results returned by the big model (D_{r}), and the inference time of the discriminator (T_{d}).

5.2 Discriminator Selection

During the online phase, we choose the optimal discriminator from those trained in the offline phase based on the transmission conditions and latency budget. Specifically, we monitor the network bandwidth B at regular intervals (*e.g.*, once an hour or every half an hour) and calculate the estimated average inference latency $L_1, \ldots, L_{\mathrm{n}}$ using different discriminators $DC_1, \ldots, DC_{\mathrm{n}}$ under the current transmission conditions. The average inference latency L_{n} can be estimated as

$$L_{\mathrm{n}} = T_{\mathrm{s}} + T_{\mathrm{d}} + (T_{\mathrm{b}} + (D + D_{\mathrm{r}})/B) * U_{\mathrm{n}} \tag{2}$$

Subsequently, we calculate the difference $Loss_{\mathrm{n}}$ between the user-defined latency budget l and the estimated average inference latency L_{n}. The different $Loss_{\mathrm{n}}$ is calculated by:

$$Loss_{\mathrm{n}} = |L_{\mathrm{n}} - l| \tag{3}$$

Finally, discriminator DC_{i} with minimum different $Loss_{\mathrm{i}}$ is chosen as the optimal discriminator.

6 Extensions to Video Input

We extend DCSB to video inputs on top of the prevailing "tracking + detection" mode for video analysis [23], [35], [36]. Specifically, lightweight object trackers are applied to infer the current position of the object from the previous frame and object detection is only triggered at certain sampling rate on key frames to update the tracked object. Accordingly, we add an object tracker and an adaptive video sampling module into DCSB (see Fig. 8). The former tracks objects between adjacent frames, while the latter reduces the triggering of object detection by adaptively adjusting the sampling rate. These two modules operate in parallel with the original DCSB and are deployed on the edge device.

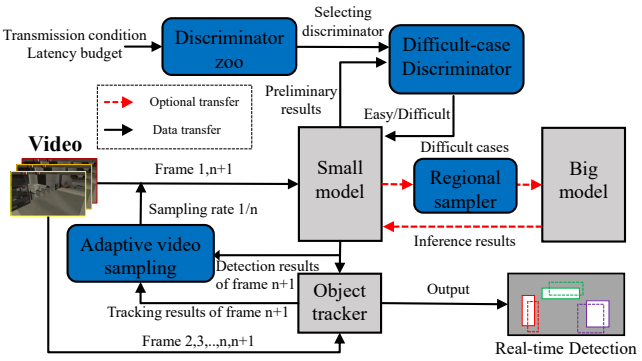


Fig. 8. Workflow to handle video input.

Algorithm 1: Adaptive Sampling Rate Update

Input: R_{t}: Tracking results for the current sample frame, R_{d}: Detection results of the current sampled frame, S^{p}: The previous sampling rate status, f: The current sampling rate, Δ: The speed of sampling rate changes, f_{MAX}: The maximum sampling rate
Output: f_{next}: Sample rate of next time
ALGORITHM:
Step 1: Calculate I
$I \leftarrow \mathrm{mIoU}(R_{\mathrm{t}}, R_{\mathrm{d}})$
Step 2: Determine current sample rate status S^{c} using I
if $I \geq T_{\mathrm{c}}$ **then**
$S^{\mathrm{c}} \leftarrow 1$
else
$S^{\mathrm{c}} \leftarrow 0$
end if
Step 3: Determine the speed Δ of sampling rate changes combining with S^{c} and previous sample rate status S^{p}
if it is the initial sampling **then**
$\Delta \leftarrow \Delta_0$
else if $S^{\mathrm{c}} = S^{\mathrm{p}}$ **then**
$\Delta \leftarrow \Delta \times \alpha$
else
$\Delta \leftarrow \Delta \times \beta$
end if
Step 4: Generate the sampling rate of next time f_{next}
if $S^{\mathrm{c}} = 1$ **then**
$f_{\mathrm{next}} \leftarrow f - \Delta$
else
$f_{\mathrm{next}} \leftarrow f + \Delta$
end if
if $f_{\mathrm{next}} > f_{\mathrm{MAX}}$ **then**
$f_{\mathrm{next}} \leftarrow f_{\mathrm{MAX}}$
end if

6.1 Object Tracker

We empirically choose the Oriented FAST and Rotated BRIEF (ORB) feature extraction method [37] and the Lucas-Kanade (LK) optical flow method [38] as the object tracker. The ORB enables accurate and real-time feature point extraction at the edge device [35]. The LK optical flow can estimate the positions of feature points in next frame by a local image flow (velocity) vector $(V_{\mathrm{x}}, V_{\mathrm{y}})$ [35], [39].

6.2 Adaptive Frame Sampling

To reduce the frequency to trigger object detection without compromising accuracy, we leverage an adaptive sampling scheme, which adjusts the sampling rate based on the current tracking and detection results. Specifically, a decline in tracking accuracy indicates rapid shifts in video content, and thus necessitates an increased sampling frequency to compensate for the tracking errors. Conversely, stable video content allows for a reduced sampling pace.

As shown in Algorithm 1, we employ the mean Intersection over Union (mIoU) I between tracking and detection results as the metric for tracking accuracy (Step1), and adjust next time the sampling rate f_{next} based on a predefined threshold T_{c} and current sampling rate f (Step2). When the mIoU exceeds T_{c}, we decrease the sampling rate f by Δ ($f_{\text{next}} = f - \Delta$); otherwise, we increase the sampling rate f by Δ ($f_{\text{next}} = f + \Delta$, Step4). We further introduce two state variables S^{c} and S^{p} to refine Δ, increasing or decreasing Δ based on the consistency of S^{c} and S^{p} (Step3). S^{c} represents the current sampling rate state, and S^{p} represents the previous sampling rate state. When the two states are the same, we increase the sampling rate change speed Δ. Conversely, when the two states are different, we decrease Δ. When the sampling rate states are the same for two consecutive instances ($S_{\text{c}} = S_{\text{p}}$), it indicates that the current sampling rate change speed Δ is insufficient to maintain system accuracy. For instance, if the sampling rate has been increasing for two consecutive instances ($S_{\text{c}} = S_{\text{p}} = 0$), it suggests that the system has encountered challenging scenes within the interval period. In such cases, it is necessary to increase the sampling rate change speed Δ to rapidly increase the sampling rate and adapt to the changing environment. Inspired by Hill Climbing, a classic optimization technique in hyper-parameter search [40], we treat the sampling rate as a hyper-parameter. We set the threshold T_{c} to 0.5 as in previous studies [35], [39]. The α and β are the ratio of Δ increase and decrease, respectively. Users can adjust them based on the specific video context. For instance, on the Helmet dataset, we set α and β to 1.2 and 0.5, respectively. In the case study experiment, where the video scenes are more stable, we adjusted α and β to 1.0 and 0.8. To select the optimal hyper-parameters, we considered previous research [40], in conjunction with our experimental results, to determine these parameter settings. Constrained by the inference latency of the object detection, the next time sampling rate $f_{\text{next}} \leq 1/(T_{\text{MAX}} * fps)$, where T_{MAX} and fps are the maximum inference time and the frames per second of the video, respectively.

$$T_{\text{MAX}} = T_{\text{s}} + T_{\text{b}} + T_{\text{d}} + (D + D_{\text{r}})/B \tag{4}$$

7 Evaluation

7.1 Experiment Setup

7.1.1 Hardware and Software

We use Jetson Nano [41] and Jetson TX2 [42] as the edge device. Unless specified, the evaluation result on Jetson Nano is reported. We emulate a cloud with a PC equipped with an AMD Ryzen9 5900HX CPU, an NVIDIA RTX3060 GPU, and 32G RAM. The client and the server are connected via WLAN. The default network bandwidth is 8Mbps. We implement DCSB with PyTorch 1.7 [43] and TensorRT [44], deploying a small model on the edge device and a big model on the server.

7.1.2 Datasets

We test four datasets.

- **Voc2007**: PASCAL VOC2007 [28], train on VOC2007 trainval (5011 images); and test on VOC2007 test (4952 images).
- **Voc2007+2012**: PASCAL VOC2007+VOC2012 [45], train on the union of VOC2007 trainval (5011 images) and VOC2012 trainval (11540 images), test on VOC2007 test (4952 images).
- **COCO**: COCO [46] trainval 135k. It has 80 object classes. We select 98,267 images containing 18 object classes, which are the same 18 classes as the VOC dataset. 5% are randomly selected for test (4914 images), the remaining are for training (93353 images).
- **Helmet**: It comes from the KubeEdge [47] open-source sub-project Sedna, which is derived from the videos collected by the camera on building sites. The videos cover various classes: blur, occlusion, water stains, smoke, insufficient light, etc., which can test the robustness of our method. The training set and test set created by KubeEdge contain 1,600 and 209 video frames, respectively, each containing 4480 and 606 objects.

7.1.3 Big Model & Small Model & Discriminator

We use SSD [48], YOLOv4 [30] and Resnet101-RetinaNet [49] as big models, and Mobile v1 [29], Mobile v2 [50] and ResNet18 [21] as small models. The image sizes for training these three sets of small-big models are 300x300 for SSD, 416x416 for YOLOv4, and 600x600 for RetinaNet. The batch sizes are set to 16 for three sets of small-big models. The initial learning rate are 1×10^{-2}, 2×10^{-3} and 1×10^{-4} for SSD, YOLOv4 and RetinaNet, respectively. We use a cosine schedule for learning rate decay. All other hyperparameters and pre-trained models were configured according to the official code settings.

The training process of the discriminator uses weights initialized with a normal distribution and the Adam optimizer. The initial learning rate is set to 9e-3, and the batch size is 128. We have uploaded the key code of DCSB to GitHub. You can access the repository using the following link: https://github.com/zhiqiangcao1218/DCSB.

7.1.4 Baselines

We compare DCSB with the following methods.

- **Edge-Only**: Only execute the small model on edge devices.
- **Cloud-Only**: All data are uploaded to the cloud.
- **CAS** [25]: The state-of-art adaptive model partitioning framework.
- **Early-exit** [51]: It allows a neural network to terminate its inference process early, thereby reducing unnecessary computation and inference time for certain inputs.

For video inputs, we further compare the adaptive sampling scheme in DCSB with the following methods.

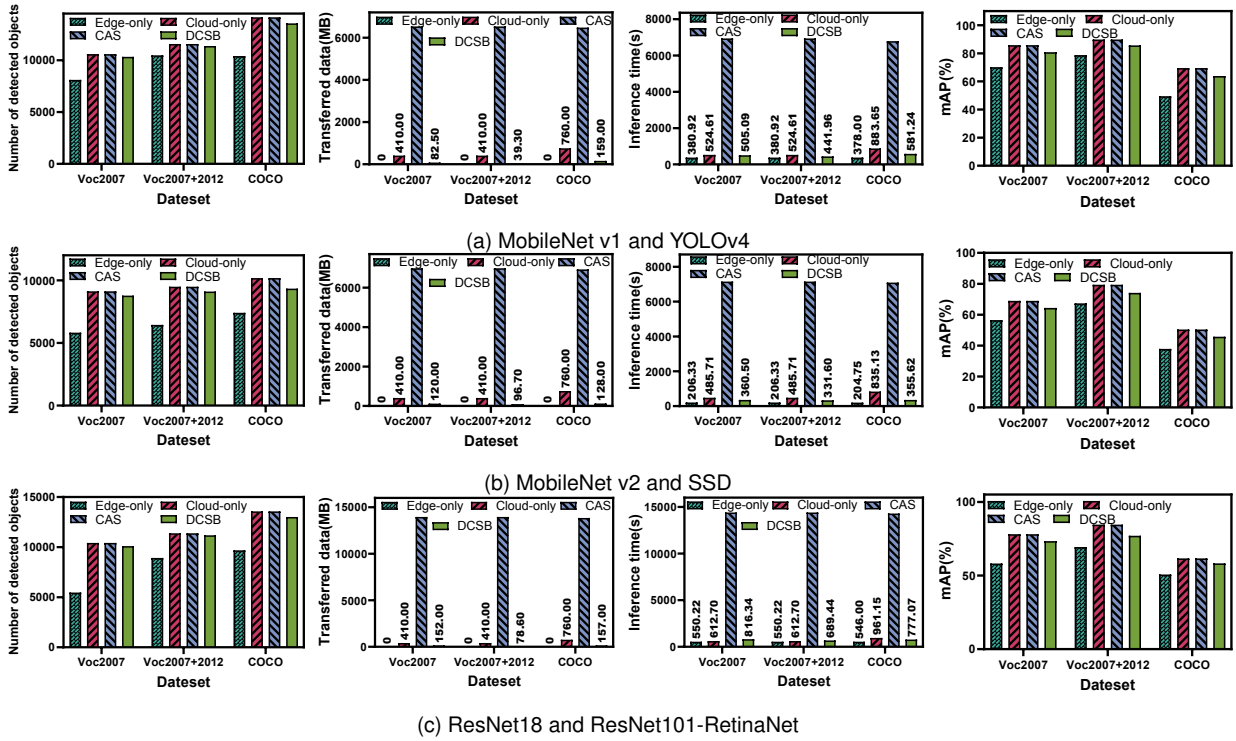


Fig. 9. Performance with different models and datasets when the transmission bandwidth is 8Mbps and difficulty criterion $\alpha = 1$.

- **Fixed Sampling Rate**: The sampling rate remains constant.
- **AMS** [52]: It adjusts the frame sampling rate dynamically based on the extent and speed of scene change in a video.
- **EdgeDuet** [27]: It adjusts the frame sampling rate dynamically based on the object's speed in continuously tracked frames.

7.2 Performances Overview

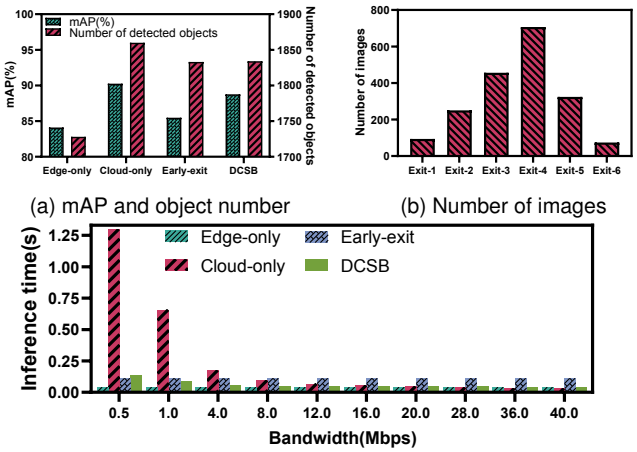
Performance on MobileNet v1 and YOLOv4. In this thread of experiments, we use MobileNet v1-YOLOv4 and YOLOV4 as the small and the big model, respectively. The experimental results are shown in the Fig. 9 (a). Across the three datasets, DCSB demonstrates an impressive ability to detect an average of 97.25% of objects, with only about an average 93.60 MB of data being transferred. It saves 82.23% of the bandwidth resources compared with the cloud-only method, and 98.56% of the bandwidth resources compared with CAS, respectively. The efficacy of DCSB is attributed to its strategy of exclusively uploading difficult cases and proficiently compressing the data associated with such difficult cases.

On the contrary, the edge-only method can only detect about 73.50% of objects. This is because the limited computation resources of the edge devices can not perform accurate DNN inference for complex object detection tasks. Obviously, CAS suffers from a much larger bandwidth consumption and longer inference latency due to the huge amount of intermediate data when applied to object detection. For example, the intermediate data size for one image in CAS is about 1.32MB, while the average data size of a compressed image is only about 0.06 MB. Also, DCSB improves the end-to-end mAP by 17.61% compared with the edge-only method and saves about 92.60% of inference time compared with the model partition method CAS.

Performance on MobileNet v2 and SSD. In this thread of experiments, we use MobileNet v2-SSD and SSD as the small and the big model, respectively. In general, on three datasets, DCSB can detect 96.26% of objects with only about 114.90MB data being transferred on average. It saves 76.77% of the bandwidth resources compared with the cloud-only method, and 98.35% of the bandwidth resources compared with CAS, respectively. Additionally, DCSB enhances the end-to-end mAP by 15.97% when contrasted with the edge-only method, and achieves a noteworthy 95.10% reduction in inference time compared to CAS.

Performance on ResNet18 and ResNet101-RetinaNet. In this thread of experiments, we use ResNet18-RetinaNet and ResNet101-RetinaNet as the small and the big model, respectively. Across the three datasets, DCSB can detect an average of 97.96% of objects and save 74.37% of the bandwidth resources compared with the cloud-only method, and 99.07% of the bandwidth resources compared with CAS, respectively. Moreover, DCSB demonstrates a significant increase in end-to-end mAP, achieving a 17.38% improvement over the edge-only method. Simultaneously, it achieves substantial efficiency gains, reducing inference time by about 94.70% compared to CAS.

The experimental results exhibit stability across all three diverse datasets and three sets of models, affirming the robustness of DCSB under varying application scenarios.



(a) mAP and object number (b) Number of images

(c) Inference time under various bandwidths

Fig. 10. (a) Performance of DCSB and baselines on single object dataset.(b) Number of images at different exit points. (c) The inference time of DCSB and baselines under different bandwidths.

7.3 Compared with Early-exit methods in the Single-Object Detection scenario

In this section, we primarily evaluate the performance of DCSB and early-exit networks for single-object detection. We use MobileNet v2-SSD and SSD as the small and the big model, respectively. SSD extracts six feature layers from the backbone into the detection head for object recognition, so we set six exit points based on the model structure of SSD. We determine the exit points based on the confidence scores of the detection results, setting the confidence threshold at 0.5 according to previous research [28], [51], [53]. We filter the Voc2007+2012 dataset, retaining only single object images. After filtering, the training set consists of 6,821 images, and the test set contains 1,905 images. Subsequently, we train the discriminator, maintaining consistency with the experimental setup used in prior experiments. The accuracy of the small model, big model, and discriminator are 84.10%, 90.25%, and 97.64% on the single object dataset, respectively. As shown in Fig. 10 (a), compared to the early-exit method, DCSB detects the same number of objects but achieves higher inference accuracy (88.77% versus 85.48%). As shown in Fig. 10 (c), our method incurs less time overhead compared to the early-exit method. For instance, we reduced the inference time by 56.05% compared to the early-exit method when the transmission bandwidth is 8Mbps. As shown in Fig. 10 (b), only 18.06% of the images exit from the first two layers, so the computational efficiency gains from the early-exit method are quite limited. When the bandwidth is extremely low, such as at 0.5 Mbps, the time overhead of the early-exit method becomes less than our method.

Overall, in most cases, our method outperforms the early-exit approach in both inference accuracy and time, and it is applicable to a broader range of visual tasks.

7.4 Inference Latency under Various Bandwidths

In practical applications, the network bandwidth is subject to fluctuations. The network bandwidth will affect the inference latency and the discriminator selection. To simulate a real environment, we evaluate the inference latency variation of the four frameworks on the Voc2007 dataset when the network bandwidth changes from 4Mbps to 40Mbps. Using the Mobile v1 and YOLO v4, other conditions are the same as above. Fig. 11 plots the inference latency when the network bandwidth changes from 4Mbps to 40Mbps. The discriminator zoo contains three different discriminators, corresponding to α being 1, 2, and 3. As the bandwidth varies, the system automatically selects the discriminator that is closest to the latency budget. When the bandwidth is less than or equal to 4 Mbps, the DCSB chooses the discriminator associated with α equal to 2. Conversely, when the bandwidth exceeds 4 Mbps, the DCSB opts for the discriminator linked to α equal to 1. For the bandwidth-constrained edge, DCSB outperforms both CAS and cloud-only methods.

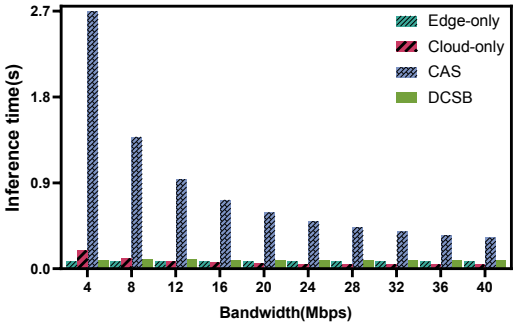
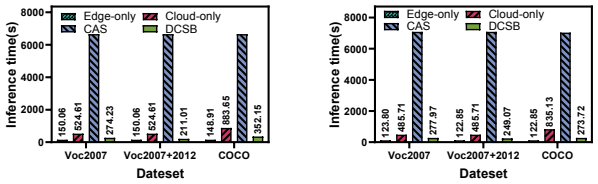


Fig. 11. The inference time of DCSB and baselines under different bandwidths when the latency budget is 0.1s.

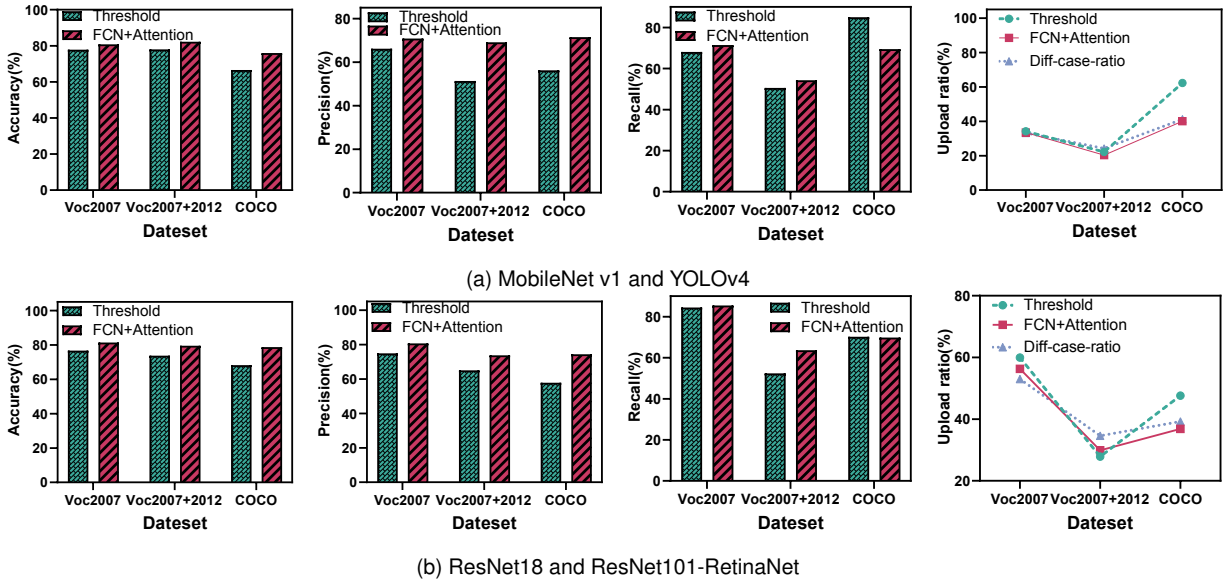


(a) MobileNet v1 and YOLOv4 (b) MobileNet v2 and SSD

Fig. 12. The inference time of DCSB and baselines with different models and datasets on Jetson TX2 when the transmission bandwidth is 8Mbps and difficult criterion $\alpha = 1$.

7.5 Inference Latency on Jetson TX2

In this thread of experiments, we evaluated the inference latency of DCSB and baselines on the Jetson TX2. The experimental results are shown in the Fig. 12. DCSB achieves a noteworthy 95.81% and 96.22% reduction in inference time compared to CAS in two sets of models, respectively. In comparison to the experimental results under equivalent conditions on the Jetson Nano, there was an improvement of 3.2% and 1.12% respectively. This is attributed to the more powerful computational capabilities of the Jetson TX2, leading to a reduction in the inference latency of the small model. Consequently, our DCSB can yield greater gains.



(a) MobileNet v1 and YOLOv4

(b) ResNet18 and ResNet101-RetinaNet

Fig. 13. Performance of current version (FCN+Attention) and preliminary discriminator (Threshold) with different models and datasets when the difficult criterion $\alpha = 1$. Diff-case-ratio: the proportion of difficult cases in the dataset.

7.6 Performances of Difficult-case Discriminator

7.6.1 Comparison with the Preliminary Version

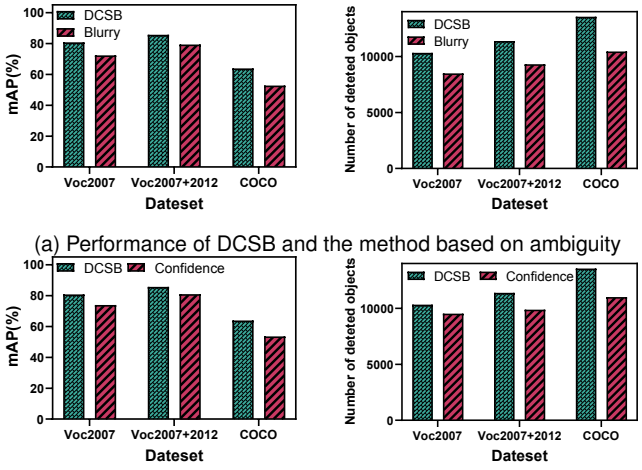
We first evaluate the performance of our new discriminator (FCN + Attention) and the preliminary discriminator (threshold-based) [26]. The average inference time for the new and preliminary versions of the discriminator is 2ms and 4ms, respectively, on Jetson Nano. The FCN+Attention method, which employs a 4-layer fully connected network, does not impose significant computational overhead on edge devices.

As shown in Fig. 13, across the three datasets and two sets of models, the FCN+Attention method achieves an average accuracy improvement of 6.26% compared to the threshold-based method. Moreover, in terms of precision and recall, the FCN+Attention method maker outperforms the threshold-based method. Compared to the preliminary discriminator, the new version discriminator reduces the upload ratio by 4.77% across various datasets, approaching more closely the inherent difficult-case ratio of the datasets. In summary, the new version of the discriminator surpasses the preliminary version in terms of both accuracy and upload ratio, among other aspects.

7.6.2 Comparison with Other Difficult-case Discriminators

We compare DCSB with two other basic difficult-case discriminating strategies: blurred images uploading, and uploading according to the top-1 confidence score. We adopt the Mobile v1 and YOLO v4 as the small-big model. For the sake of fairness, we set the same upload ratio for DCSB and baselines in three datasets.

Upload Blurred Images to the Cloud. There are many ways to define the ambiguity of an image. We choose the Brenner gradient function to define ambiguity. The Brenner gradient function is one of the simplest gradient evaluation functions.



(a) Performance of DCSB and the method based on ambiguity

(b) Performance of DCSB and the method based on top-1 confidence score.

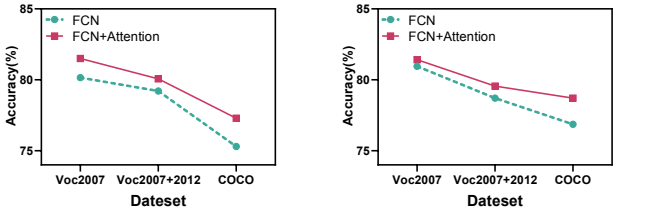
Fig. 14. Performance of DCSB and two basic difficult-case discriminators

It calculates the square of gray level differences between two pixels. The function is defined as follows:

$$\sum_{y}\sum_{x}|f(x+2,y)-f(x,y)|^2 \tag{5}$$

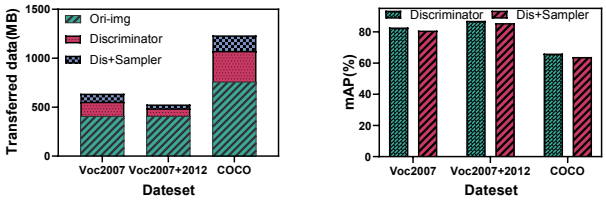
where f(x, y) is the gray value of the pixel (x, y). The larger the value of the function, the clearer the image. The results are shown in Fig. 14 (a). For the end-to-end mAP, DCSB significantly outperforms this method by up to 13.55%; in terms of the number of detected objects, DCSB is 24.56% higher.

Upload Method Based on the Top-1 Confidence Score. The confidence scores of the bounding boxes represent how confident the object detection algorithm is in the classification result. The higher the confidence score, the more confident



(a) MobileNet v2 and SSD (b) ResNet18 and ResNet101-RetinaNet

Fig. 15. Impact of position attention. FCN: full connected network. FCN+Attention: full connected network with positional attention.



(a) Transferred data (b) mAP

Fig. 16. Impact of regional sampler module. Ori-img: the total sizes of all original images in the test set. Discriminator: the transferred date of DCSB with discriminator. Dis+Sampler: the transferred date of DCSB with discriminator and regional sampler.

the model is in the detection result. Therefore, it can be used to evaluate whether the image is a difficult case or an easy case. Take the top-1 of the recognition boxes of each type of object in a single image, and then add a total of 20 confidence scores for 20 categories (VOC dataset) and then take the average value. According to this value, we upload data to the cloud for processing while maintaining the same cloud upload ratio as DCSB in three datasets. The results are shown in Fig. 14(b). For the end-to-end mAP, DCSB has an average improvement of 10.51%; regarding the number of detected objects, DCSB is 11.61% higher than that strategy. This method is far better than the other baseline, but it is still much worse than our semantic-based uploading strategy.

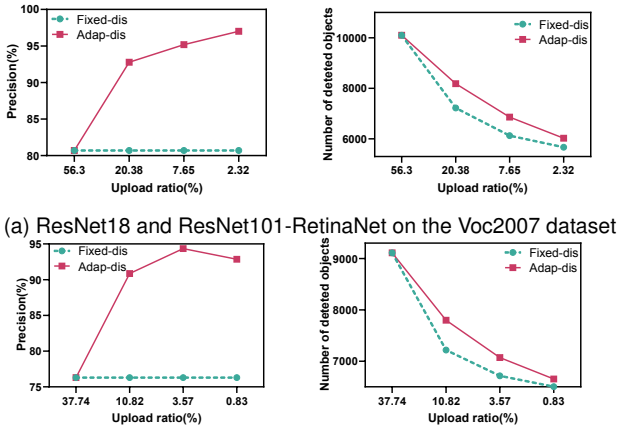
7.7 Ablation Studies

7.7.1 Impact of Position Attention Module

In this section, we primarily examine whether position attention has an impact on the accuracy of discriminator. As shown in Fig. 15, the FCN with added positional attention exhibits significant improvement in accuracy across all three datasets. Taking the example of Mobile v2 and SSD as the small-big model, the FCN+Attention method achieves an average accuracy improvement of 1.8% compared to the FCN method. In two sets of models and three datasets, Compared to the FCN method, the FCN+attention approach shows an increase in accuracy by 1.79%. Overall, the position attention assists FCN in better extracting data features, leading to an improvement in accuracy.

7.7.2 Impact of Regional Sampler Module

In this section, our primary emphasis is to evaluate the contribution of the regional sampler module to the reduction



(a) ResNet18 and ResNet101-RetinaNet on the Voc2007 dataset

(b) MobileNet v2 and SSD on the Voc2007+2012 dataset

Fig. 17. Impact of discriminator zoo module. Fixed-dis: the discriminator ($\alpha = 1$) is fixed. Adap-dis: adaptively selecting the optimal discriminator from the discriminator zoo based on transmission conditions and latency budget.

of transferred data and its influence on accuracy. As shown in Fig. 16, we adopt the Mobile v1 and YOLO v4 as the small-big model. On the three datasets, the discriminator reduced the average uploaded data by 67.98% compared to the cloud-only method (ori-img in Fig. 16). On top of this, the regional sampler module further decreased the data by 47.52%. Taking the Voc2007 dataset as an example, the cloud-only method requires uploading 410MB of data, the discriminator needs to upload 146MB of data, and after passing through the region sampler, the data of difficult-case reduced to only 82.50MB. Moreover, the region sampler has a minimal impact on the end-to-end map, as shown in Fig. 16 (b). Across the three datasets, the dis+sampler exhibits an average decrease of only 1.88% compared to the discriminator method. This is attributed to the fact that the region sampler compresses only non-critical regions, preserving the original resolution for key regions.

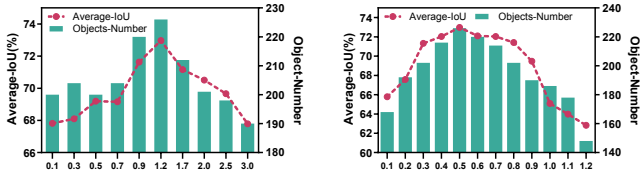
7.7.3 Impact of Discriminator Zoo

In this section, we primarily compare the performance of the fixed discriminator (Fixed-dis) approach with our proposed adaptive bandwidth selection discriminator (Adap-dis) across different datasets and models. As shown in Fig. 17, we employ two sets of models, and for each model group, we train four difficult-case discriminators. These discriminators corresponded to different difficult-case criteria (α =1,2,3,4). The fixed-dis utilizes the discriminator trained when α was set to 1.

As shown in Fig. 17, with the reduction in cloud upload ratio, the Adap-dis method exhibits significantly higher precision compared to the Fixed-dis method. This is because raising the difficult-case criteria improves the precision of identifying difficult cases. When bandwidth decreases, reducing the amount of data uploaded without improving the precision of difficult case identification would lead to a decline in recognition performance. In different upload ratios and models, the Adap-dis method detects 7.86% more objects compared to the Fixed-dis method. This is due to the Adpa-dis method selecting a discriminator ($\alpha > 1$)

TABLE 1
Performance of DCSB and baselines on the Helmet dataset.

Method	Average-IoU(%)↑	Objects-Number↑	Dete-Number↓
Fixed-10	80.93	226	24
Fixed-20	72.60	188	12
Fixed-30	70.46	206	8
Fixed-40	68.58	192	6
DCSB	**72.97**	**226**	**6**
AMS	73.70	188	14
EdgeDuet	73.08	228	10



(a) α: the ratio of Δ increase (b) β: the ratio of Δ decrease

Fig. 18. (a) Fixing the value of β at 0.5, we evaluate the performance of our method under different values of α. (b) Fixing the value of α at 1.2, we evaluate the performance of our method under different values of β.

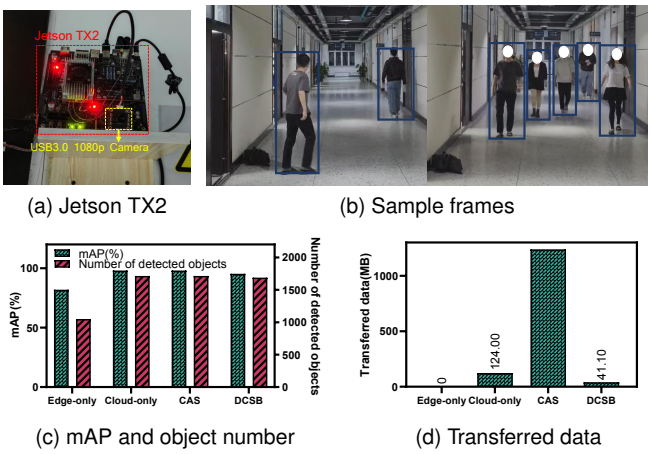
that uploads images with more missed objects to the cloud. Therefore, compared to a fixed discriminator ($\alpha = 1$), it can detect more objects.

7.8 Performance of DCSB in Video Task

In this thread of experiments, we use MobileNet v1-YOLOv4 and YOLOv4 as the small and the big model, respectively. The accuracy of the small model, big model, and discriminator are 44.67%, 68.80%, and 78.47% on the Helmet dataset, respectively. As shown in Table 1, we primarily compare our proposed adaptive dynamic sampling rate method with the baselines in terms of accuracy (Average-IoU and Objects-Number) and efficiency (Dete-Number: the number of times object detection is triggered). "Fixed-xx" denotes the method with a fixed sampling rate, where video frames are sampled every xx frames. Compared to the fixed sampling rate method, DCSB detects the same number of objects but reduces the frequency of object detection triggers by 75%. This significantly reduces the computational overhead on resource-constrained edge devices. Compared to AMS and EdgeDuet, DCSB achieves similar detection accuracy but reduces the number of object detection triggers by 57.14% and 40%, respectively. EdgeDute divides objects into several intervals based on their velocity and assigns different sampling rates accordingly. In contrast, our method adjusts the sampling frames more finely based on the detection results of the previous sampling frame.

7.9 Performance Evaluation Under Various Hyper-parameters

In this section, we primarily evaluate how different hyper-parameters (α, β in Algorithm 1) values impact the overall performance of our method. We use MobileNet v1-YOLOv4 and YOLOv4 as the small and the big model, respectively. As shown in Fig. 18 (a), fixing the value of β, Setting α to 1.2 achieves the best detection accuracy. When the value of α is too small, DCSB lacks sufficient momentum to escape its current state, affecting detection accuracy. Conversely, if α is set too high, the sampling intervals may become excessively large, diminishing the detection accuracy. The other hyper-parameter β controls the reduction rate of the sampling frequency change Δ. When the sampling states of two consecutive instances are inconsistent, it indicates that the system has escaped from its predicament. Consequently, Δ should be reduced to stabilize the system's state. As shown in Fig. 18 (b), setting β to 0.5 achieves optimal recognition accuracy.



(a) Jetson TX2 (b) Sample frames

(c) mAP and object number (d) Transferred data

Fig. 19. (a) Sample frames are collected from the lab corridor. (b) The Jetson TX2 and a 1080p camera are fixed at the end of the corridor. Performance of DCSB and baselines on real-world data (c-d).

8 Case Study

We implement DCSB on a real-world video monitoring system deployed in the lab corridor. The system is designed to detect pedestrians in the corridor. We obtain permission for student volunteers, and the process is approved by our institution's IRB. As shown in Fig. 19 (a), a 1080p camera captures video, and the Jetson TX2 edge device is used to detect pedestrians. The lab server and other experimental setups are detailed in Sec. 7.1. In this thread of experiments, we use MobileNet v1-YOLOv4 and YOLOv4 as the small and the big model, respectively. We manually annotate the ground truth for the video frames. As shown in Fig. 19 (b), we evaluate the performance of DCSB and baselines under varying levels of pedestrian density. The experimental results are shown in the Fig. 19 (c-d). DCSB demonstrates an impressive ability to detect 98.54% of objects, with only 41.10 MB of data being transferred. It saves 66.85% of the bandwidth resources compared with the cloud-only method, and 96.68% of the bandwidth resources compared with CAS, respectively. Additionally, DCSB enhances the end-to-end mAP by 16.46% when contrasted with the edge-only method.

9 Related Work

Our work is related to the following categories of research.

9.1 Object Detection

Object detection can be broadly classified into two approaches based on the detection process. Two-stage algorithms, like R-CNN [54] and Faster R-CNN [55], operate in a sequential manner. They first generate candidate regions, known as region proposals, and then classify objects within these regions and refine their locations. In contrast, one-stage algorithms, such as SSD [48], RetinaNet [49], and YOLOv4 [30], simultaneously predict object classes and locations in a single pass. More recently, transformer-based object detection has gained increasing popularity [56], [57], [58]. DETR treats object detection as a set prediction problem and employs a novel transformer-based architecture [59]. Similarly, ViT-FRCNN [60] replaces the convolutional backbone with a transformer. Despite their efficacy, these advanced object detection models entail considerable computational demands [39], making edge-cloud collaboration a more viable deployment strategy for resource-constrained edge devices.

9.2 Model Compression

Model compression aims to reduce the computational demand of DNNs, enabling their deployment on devices with limited capabilities without notably sacrificing performance [61], [62]. For example, network pruning [61] and lightweight networks (MobileNet v1 [29] and MobileNet v2 [50]) can reduce the DNN inference workload, but often incur noticeable accuracy loss. However, object detection algorithms usually must achieve high accuracy and low latency, as they are frequently applied in safety-critical applications, *e.g.*, autonomous driving [63] and assistance for the visually impaired [64]. Current compression techniques struggle to deliver satisfactory accuracy and latency for object detection.

9.3 Cloud Offloading

By offloading certain computational tasks to the cloud, edge devices can perform intensive operations beyond the limitations of their hardware. For example, CloneCloud [65] seamlessly offloads parts of a mobile application to device clones operating in the cloud. Microsoft Brainwave [9] accelerates DNN inference in major services like Bing's intelligent search and Azure. Cloud offloading often consumes substantial bandwidth, which can be problematic for bandwidth-restricted and time-sensitive mobile applications.

9.4 Device-Cloud Collaboration

To balance inference accuracy and computational overhead, some works have already proposed offloading part of the computational tasks from edge devices to the cloud, introducing methods such as model partitioning and early-exit methods.

Model partitioning is a promising alternative for efficient DNN inference leveraging the resources on both the cloud and edge device [25], [66], [67], [68]. For example, Neurosurgeon [15] partitions a CNN between the device and cloud and determines the best split point based on workload and networking conditions. CAS [25] presents a context-aware adaptive surgery framework that perceives changing processing contexts and dynamically finds suitable partition solutions in real time. Yet, these solutions are primarily designed for image classification and struggle with object detection. This is because partitioning object detection models requires transferring high volume of intermediate data, leading to significant communication delays as in traditional cloud offloading [19]. In contrast, we introduce a novel framework that employs a difficult-case based small-big model architecture. It effectively reduces communication costs and overall inference latency while maintaining high accuracy dedicated for object detection.

Early-exit methods [17], [69], [70] allow a neural network to terminate its inference process early, thereby reducing unnecessary computation for certain inputs and consequently lowering the overall inference time and computational overhead. BranchyNet [53] enables local devices to avoid sending feature maps to the cloud server by adding exit branches to the DNN, allowing inference to be completed locally once a certain accuracy level is achieved. ATHEENA [71] proposes a tool-flow for hardware early-exit network automation to reduce area while maintaining accuracy. EINet [72] simultaneously considers inference accuracy and time to determine the optimal exit point. MAMO [73] proposes a novel model for inference acceleration with exit selection and model partitioning. Both early-exit networks and our method share the same high-level idea, *i.e.*, simple samples can meet accuracy requirements without complex computations. Accordingly, inference overhead can be reduced without compromising accuracy. But most studies on early-exit networks are designed for image classification (where there is only one object in an image) and cannot be applied to object detection (where multiple objects may be present in an image). This is because the exit mechanisms for image classification are based on the difficulty of the entire image. They would fail in object detection because it requires finer-grained processing at regional or even object level. For example, when an image contains two objects, one is easy and the other is difficult to recognize. Then the easy object demands a shallow exit whereas the difficult object requires a deep exit, which cannot be easily implemented with existing early-exit networks. There are a few works that apply early-exit networks to object detection. TEEM [74] uses an early-exit structure to eliminate redundant frames in video object detection. Moos *et al.* [51] deploy early-exit networks to the single-object detection task to reduce overall inference time. However, these proposals are dedicated for certain scenarios and do not apply to generic object detection. In contrast, our DCSB is a versatile solution applicable to both image classification and object detection.

In the mobile computing community, there are also other types of device-cloud collaboration works. For instance, MARIA *et al.* [75] design an incentive mechanism to address the issue of computational task offloading from users to edge servers and the allocation of uplink transmission power, with the aim of minimizing network-wide end-to-end energy consumption and enhancing resource utilization efficiency. WAVE [2] achieves high-accuracy real-time object detection under bandwidth-constrained cellular networks through strategies like deep RoI encoding and

prioritized parallel offloading. Trine [36] enables low-cost real-time video analysis through the collaborative efforts of cloud, edge, and terminal devices. LEAF+AIO [1] balances the energy consumption of mobile augmented reality (MAR) devices with tracking accuracy loss by optimizing offloading frequency and local computing configurations. The aforementioned works are orthogonal to our research direction; their techniques and methods can be effectively applied to our study, with the aim of further enhancing the performance and outcomes of our research.

10 Future work

In the future, we aspire to enhance DCSB in the following areas: *(i)* The current design focuses on a single-client scenario, but we plan to extend the DCSB principle to multi-client scenarios. By employing methods such as client clustering and adding adaptive modules, we aim to reduce overall computational resource consumption while maintaining decision accuracy. *(ii)* We plan to extend the principles of DCSB to other complex visual tasks, such as semantic segmentation, instance segmentation *et al. (iii)* The discriminator currently classifies samples based on the results from a small model, leading to unnecessary computational resource usage. We plan to optimize this process so that samples can be classified without passing through the small model.

11 Conclusion

In this paper, we present DCSB, an edge-cloud collaboration framework for object detection based on a difficult-case discriminator. In this framework, a difficult-case discriminator is designed to quickly decide whether an image should be processed locally by the small model at the edge, or be uploaded to the cloud to be further processed by the big model. Moreover, we design a regional sampling algorithm that adaptively down-samples the regions of already-detected objects to save the communication bandwidth when uploading difficult cases. To adapt to changes in transmission conditions, we propose the dynamic selection of the optimal discriminator through the discriminator zoo module. Finally, we extend DCSB to video tasks and introduce an algorithm for adaptive adjustment of sampling rates, aiming to reduce computational overhead without compromising detection accuracy. Extensive experiments on the real edge device demonstrate the effectiveness of the DCSB on various datasets and object detection algorithms. DCSB can detect 96.26%-97.96% of objects but save 74.37%-82.23% of network bandwidth compared with the cloud-only method and 98.35%-99.07% of the bandwidth resources compared with CAS, respectively. Also, DCSB improves the end-to-end mAP by 15.97%-17.61% compared with the edge-only method. In addition, compared with the state-of-the-art model partition method - CAS, DCSB saves 92.60%-95.10% of the inference time when the transmission bandwidth is 8Mbps. In video tasks, compared to the state-of-the-art video analysis method - EdgeDute, DCSB achieves the same detection accuracy while reducing computational overhead by 40%.

References

[1] H. Wang, B. Kim, J. L. Xie, and Z. Han, "Leaf+ aio: Edge-assisted energy-aware object detection for mobile augmented reality," *IEEE Transactions on Mobile Computing*, 2022.

[2] L. Dong, Z. Yang, X. Cai, Y. Zhao, Q. Ma, and X. Miao, "Wave: Edge-device cooperated real-time object detection for open-air applications," *IEEE Transactions on Mobile Computing*, 2022.

[3] V. Vukotić, C. Raymond, and G. Gravier, "Multimodal and cross-modal representation learning from textual and visual features with bidirectional deep neural networks for video hyperlinking," in *Proceedings of the 2016 ACM workshop on Vision and Language Integration Meets Multimedia Fusion*, 2016, pp. 37–44.

[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[5] S. Milz, G. Arbeiter, C. Witt, B. Abdallah, and S. Yogamani, "Visual slam for automated driving: Exploring the applications of deep learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 247–257.

[6] Y. Lin, L. Deng, Z. Chen, X. Wu, J. Zhang, and B. Yang, "A real-time atc safety monitoring framework using a deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 11, pp. 4572–4581, 2019.

[7] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing dma engine: Leveraging activation sparsity for training deep neural networks," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 78–91.

[8] J. Kim, S. Park, and N. Kwak, "Paraphrasing complex network: Network compression via factor transfer," *Advances in neural information processing systems*, vol. 31, 2018.

[9] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *iEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[10] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro *et al.*, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 620–629.

[11] J. Ouyang, S. Lin, W. Qi, Y. Wang, B. Yu, and S. Jiang, "Sda: Software-defined accelerator for large-scale dnn systems," in *2014 IEEE Hot Chips 26 Symposium (HCS)*. IEEE, 2014, pp. 1–23.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[13] Security, "Data generated by new surveillance cameras to increase exponentially in the coming years." news, Oct. 2023. [Online]. Available: http://www.securityinfowatch.com/news/12160483/

[14] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 557–570.

[15] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[16] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1423–1431.

[17] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[18] C. Ding, Z. Lu, F. Juefei-Xu, V. N. Boddeti, Y. Li, and J. Cao, "Towards transmission-friendly and robust cnn models over cloud and device," *IEEE Transactions on Mobile Computing*, 2022.

[19] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, and Y. Zhang, "Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *Proceedings of*

the 27th Annual International Conference on Mobile Computing and Networking, 2021, pp. 201–214.

[20] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary," in *Proceedings of the 2019 workshop on hot topics in video analytics and intelligent edges*, 2019, pp. 27–32.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2015, pp. 124–132.

[23] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7436–7456, 2021.

[24] M. Liu, X. Ding, and W. Du, "Continuous, real-time object detection on mobile devices without offloading," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 976–986.

[25] H. Wang, B. Guo, J. Liu, S. Liu, Y. Wu, and Z. Yu, "Context-aware adaptive surgery: A fast and effective framework for adaptative model partition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 3, pp. 1–22, 2021.

[26] Z. Cao, Z. Li, Y. Chen, H. Pan, Y. Hu, and J. Liu, "Edge-cloud collaborated object detection via difficult-case discriminator," in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2023, pp. 259–270.

[27] Z. Yang, X. Wang, J. Wu, Y. Zhao, Q. Ma, X. Miao, L. Zhang, and Z. Zhou, "Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision," *IEEE/ACM Transactions on Networking*, vol. 31, no. 4, pp. 1765–1778, 2023.

[28] M. Everingham and J. Winn, "The pascal visual object classes challenge 2007 (voc2007) development kit," *Int. J. Comput. Vis*, vol. 88, no. 2, pp. 303–338, 2010.

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[30] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[31] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[32] R. Hesse, S. Schaub-Meyer, and S. Roth, "Content-adaptive downsampling in convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4543–4552.

[33] Y. Zhao, Z. Chen, and S. Luo, "Micro-expression recognition based on pixel residual sum and cropped gaussian pyramid," *Frontiers in Neurorobotics*, vol. 15, p. 746985, 2021.

[34] M. Gong, D. Wang, X. Zhao, H. Guo, D. Luo, and M. Song, "A review of non-maximum suppression algorithms for deep learning target detection," in *Seventh Symposium on Novel Photoelectronic Detection Technology and Applications*, vol. 11763. SPIE, 2021, pp. 821–828.

[35] K. Apicharttrisorn, X. Ran, J. Chen, S. V. Krishnamurthy, and A. K. Roy-Chowdhury, "Frugal following: Power thrifty object detection and tracking for mobile augmented reality," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, 2019, pp. 96–109.

[36] Y. Zhao, Z. Yang, X. He, X. Cai, X. Miao, and Q. Ma, "Trine: Cloud-edge-device cooperated real-time video analysis for household applications," *IEEE Transactions on Mobile Computing*, 2022.

[37] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[38] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, vol. 2, 1981, pp. 674–679.

[39] K. Yang, J. Yi, K. Lee, and Y. Lee, "Flexpatch: Fast and accurate object detection for on-device high-resolution live video analytics," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1898–1907.

[40] A. K. Pradhan, D. Mishra, K. Das, M. S. Obaidat, and M. Kumar, "A covid-19 x-ray image classification model based on an enhanced convolutional neural network and hill climbing algorithms," *Multimedia Tools and Applications*, vol. 82, no. 9, pp. 14 219–14 237, 2023.

[41] NVIDIA, "Nvidia jetson nano," official document, Oct. 2023. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[42] Nvidia, "Nvidia jetson tx2," Product Introduction, Oct. 2023. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/.

[43] pytorch, "pytorch," official document, Oct. 2023. [Online]. Available: https://pytorch.org/

[44] Nvidia, "tensorrt," official document, Oct. 2023. [Online]. Available: https://developer.nvidia.com/tensorrt

[45] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[46] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[47] S. Wang, Y. Hu, and J. Wu, "Kubeedge. ai: Ai platform for edge devices," *arXiv preprint arXiv:2007.09227*, 2020.

[48] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[49] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[50] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[51] A. Moos, "Efficient single object detection on image patches with early exit enhanced high-precision cnns," *arXiv preprint arXiv:2309.03530*, 2023.

[52] M. Khani, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh, "Real-time video inference on edge devices via adaptive model streaming," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4572–4582.

[53] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[54] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[55] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[56] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video swin transformer," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 3202–3211.

[57] Z. Zhang, X. Lu, G. Cao, Y. Yang, L. Jiao, and F. Liu, "Vit-yolo: Transformer-based yolo for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 2799–2808.

[58] M. Lin, M. Chen, Y. Zhang, C. Shen, R. Ji, and L. Cao, "Super vision transformer," *International Journal of Computer Vision*, pp. 1–16, 2023.

[59] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

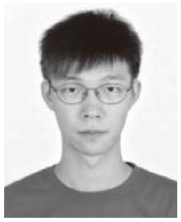
[60] J. Beal, E. Kim, E. Tzeng, D. H. Park, A. Zhai, and D. Kislyuk, "Toward transformer-based object detection," *arXiv preprint arXiv:2012.09958*, 2020.

[61] K. Xu, Z. Wang, X. Geng, M. Wu, X. Li, and W. Lin, "Efficient joint optimization of layer-adaptive weight pruning in deep neural

networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 447–17 457.
[62] L. Huyan, Y. Li, D. Jiang, Y. Zhang, Q. Zhou, B. Li, J. Wei, J. Liu, Y. Zhang, P. Wang *et al.*, "Remote sensing imagery object detection model compression via tucker decomposition," *Mathematics*, vol. 11, no. 4, pp. 1–26, 2023.
[63] J. Li, R. Xu, J. Ma, Q. Zou, J. Ma, and H. Yu, "Domain adaptive object detection for autonomous driving under foggy weather," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 612–622.
[64] J. K. Mahendran, D. T. Barry, A. K. Nivedha, and S. M. Bhandarkar, "Computer vision-based assistance system for the visually impaired using mobile edge artificial intelligence," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2418–2427.
[65] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.
[66] M. Yu, Z. Jiang, H. C. Ng, W. Wang, R. Chen, and B. Li, "Gillis: Serving large neural networks in serverless functions with automatic model partitioning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 138–148.
[67] A. Mukherjee and S. Dey, "Automated deep learning model partitioning for heterogeneous edge devices," in *Proceedings of the Second International Conference on AI-ML Systems*, 2022, pp. 1–8.
[68] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, 2018, pp. 31–36.
[69] Y. Qiu, M. Chen, W. Liang, D. Niyato, Y. Wang, Y. Li, V. C. Leung, Y. Hao, L. Hu, and Y. Zhang, "Reliable or green? continual individualized inference provisioning in fabric metaverse via multi-exit acceleration," *IEEE Transactions on Mobile Computing*, 2024.
[70] S. Laskaridis, A. Kouris, and N. D. Lane, "Adaptive inference through early-exit networks: Design, challenges and directions," in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, 2021, pp. 1–6.
[71] B. Biggs, C.-S. Bouganis, and G. Constantinides, "Atheena: A toolflow for hardware early-exit network automation," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2023, pp. 121–132.
[72] J. Huang, Y. Gao, and W. Dong, "Elastic dnn inference with unpredictable exit in edge computing," in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2023, pp. 293–304.
[73] F. Dong, H. Wang, D. Shen, Z. Huang, Q. He, J. Zhang, L. Wen, and T. Zhang, "Multi-exit dnn inference acceleration based on multi-dimensional optimization for edge intelligence," *IEEE Transactions on Mobile Computing*, vol. 22, no. 9, pp. 5389–5405, 2022.
[74] A. Sabet, J. Hare, B. Al-Hashimi, and G. V. Merrett, "Temporal early exits for efficient video object detection," *arXiv preprint arXiv:2106.11208*, 2021.
[75] M. Diamanti, P. Charatsaris, E. E. Tsiropoulou, and S. Papavassiliou, "Incentive mechanism and resource allocation for edge-fog networks driven by multi-dimensional contract and game theories," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 435–452, 2022.

Zhiqiang Cao received the master's degree in Electromagnetic Field and Microwave Technology from China radio propagation Research Institute, Qingdao, China, in 2020. He is currently pursuing the Ph.D. degree in computer science and technology with the Harbin Institute of Technology, Harbin, China. His research interests include computer vision, edge-cloud collaborative intelligence.

Yun Cheng received the B.Sc. and M.Sc. degrees in computer science and technology from Harbin Institute of Technology, Harbin, China, in 2012 and 2015, respectively.He received his Ph.D. degree from ETH Zürich in 2022. His research interests include machine intelligence and efficient, applied machine learning.



Zimu Zhou is currently a tenure-track assistant professor at the School of Data Science, City University of Hong Kong. He received his Ph.D. degree from Hong Kong University of Science and Technology in 2015. His research interests include model compression, federated machine learning, applied machine learning.

Yongrui Chen Yongrui Chen is currently an Associate Professor with the Department of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing, China. He has received his M.S degree from Tsinghua University, China in 2007, and Ph.D. degree from University of Chinese Academy of Sciences, China in 2011. His research interests include Internet of Things, wireless communications and networks.

Youbing Hu received the master's degree in computer technology from Xidian University, Xi'an, China, in 2020. He is currently pursuing the Ph.D. degree in computer science and technology with the Harbin Institute of Technology, Harbin, China. His research interests include computer vision, edge-cloud collaborative intelligence, and continuous learning.

Anqi Lu received the master's degree in computer science and technology from Heilongjiang University, Harbin, China, in 2021. She is currently pursuing the Ph.D. degree in computer science and technology with the Harbin Institute of Technology, Harbin, China. Her research interests include computer vision and satellite-terrestrial collaborative intelligence.

Jie Liu (Fellow, IEEE) is a Chair Professor at Harbin Institute of Technology Shenzhen (HIT Shenzhen), China and the Dean of its AI Research Institute. Before joining HIT, he spent 18 years at Xerox PARC and Microsoft. He was a Principal Research Manager at Microsoft Research, Redmond and a partner of the company. His research interests are Cyber-Physical Systems, AI for IoT, and energy-efficient computing.

Zhijun Li received the M.S. and Ph.D. degrees in computer science and technology from the Harbin Institute of Technology in 2001 and 2006, respectively. He is currently a Professor with the School of Computer Science and Technology, Harbin Institute of Technology. His research focuses on wireless networks, mobile computing, and artificial Internet of Things. He was a recipient of the MobiCom 2017 Best Paper Award.