ParaLoupe: Real-time Video Analytics on Edge Cluster via Mini Model Parallelization

Hanling Wang, Qing Li, *Member, IEEE,* Haidong Kang, Dieli Hu, Lianbo Ma, *Member, IEEE,* Gareth Tyson, Zhenhui Yuan, Yong Jiang, *Member, IEEE*

***Abstract*—Real-time video analytics on edge devices has gained increasing attention across a wide range of business areas. However, edge devices usually have limited computing resources. Consequently, conventional approaches to video analytics either deploy simplified models on the edge (resulting in low accuracy) or transmit video content to the cloud (resulting in high latency and network overheads) to enable deep learning inference (*e.g.,* object detection). In this paper, we introduce ParaLoupe, a novel real-time video analytics system that parallelizes deep learning inference in the edge cluster with task-oriented mini models. These mini models do not attain state-of-the-art accuracy individually, but collectively can achieve much better accuracy-latency tradeoff than state-of-the-art models. To achieve this, ParaLoupe crops multiple single-object patches from a given video frame. These single-object patches are then sent to multiple edge devices for parallel inference with specifically designed mini models. A patch-based task scheduling algorithm is further proposed to leverage the computing resources of the edge cluster to meet the service-level objectives. Our experimental results on real-world datasets show that ParaLoupe significantly outperforms baseline methods, achieving up to 14.1× inference speedup with accuracy on par with state-of-the-art models, or improving accuracy up to 45.1% under the same latency constraints.**

***Index Terms*—real-time video analytics, edge computing, distributed computing**

I. Introduction

REAL-TIME video analytics has revolutionized a range of areas, including smart cameras in our homes [1], privacy-preserving video surveillance [2], autonomous driving [3], and intelligent manufacturing [4]. A critical part of these tasks is deep learning model inference, which has become increasingly popular due to its robust performance on vision tasks. Unfortunately, however, edge devices suffer from limited computing resources, making it difficult to complete compute-intensive video analytics tasks locally. On the one hand, streaming video content to cloud servers for inference may incur prohibitively high end-to-end latency [5]. On the other hand, inference with simplified models on edge devices results in significant degradation of accuracy. Therefore, balancing accuracy and resource constraints is a critical issue in real-time video analytics on edge devices.

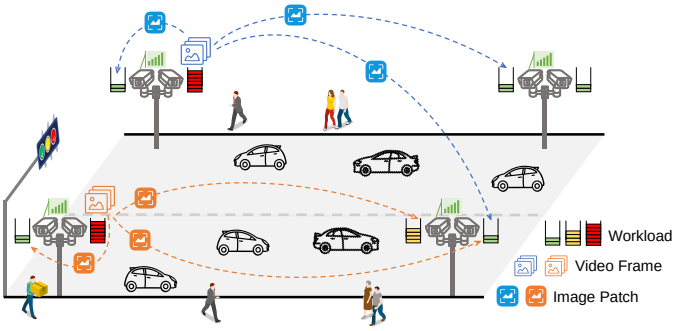

Fig. 1. A usage scenario of ParaLoupe with a cluster of intelligent cameras in the smart city.

Researchers have devoted considerable effort to address challenges in real-time video analytics on edge devices. We classify existing works into three paradigms based on whether the edge and/or cloud participate in inference: (*i*) cloud-only, (*ii*) edge-cloud collaborative, and (*iii*) edge-only. The most straightforward approach, cloud-only, is to offload the entire computation to the cloud. This typically involves optimizing the model inference process [6]–[9] on the cloud. However, transmission overhead and latency can become a bottleneck. The second approach mixes cloud and edge computation. By partially shifting computation to the edge, edge-cloud collaborative video analytics typically sends partial frames to the cloud after preprocessing [10]–[13], or performs part of the inference task via model partitioning [14], [15] at the edge. Despite this, network limitations can still cause additional overhead and processing latency. In contrast, the third approach, edge-only video analytics, eliminates the need for cloud servers by optimizing the model deployed on edge devices to reduce the computational load. This, however, is constrained by the computational resources at the edge. Several techniques have been explored to achieve lightweight inference, *e.g.,* tensor quantization [16], knowledge distillation [17], and neural architecture search [18]. However, performance is typically compromised due to the tradeoff between

Hanling Wang and Yong Jiang are with the Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong 518055, China, and also with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China. E-mail: hl-wang21@mails.tsinghua.edu.cn, jiangy@sz.tsinghua.edu.cn.

Qing Li is with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China. E-mail: liq@pcl.ac.cn.

Haidong Kang and Lianbo Ma are with the Northeastern University, Shenyang, Liaoning 110167, China. E-mail: hdkang@stumail.neu.edu.cn, malb@swc.neu.edu.cn.

Dieli Hu is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100086, China, also with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China, and also with the University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: hudieli21@mails.ucas.ac.cn.

Gareth Tyson is with the Hong Kong University of Science and Technology, Guangzhou, Guangdong 511442, China. Email: gtyson@ust.hk.

Z. Yuan is with the School of Engineering, University of Warwick, UK; e-mail: zhenhui.yuan@warwick.ac.uk

Manuscript received January 16, 2024. *(Corresponding author: Qing Li, Lianbo Ma.)*

model complexity and inference accuracy. Thus, existing approaches suffer from either high transmission latency, caused by the limited bandwidth between the edge and cloud; or accuracy loss, caused by constrained computing resources on a single edge device.

In contrast to existing approaches, we propose to conduct model inference by leveraging the computing resources of a cluster of edge devices. In this scenario, all edge devices can communicate with each other and possess certain computational resources (although significantly less than those available in the cloud server). These edge devices are not only more cost-effective than cloud servers, but also ensure lower latency due to their geographical proximity, as the network transmission within a cluster is notably faster than transmission to the cloud. For instance, as shown in Fig. 1, an edge cluster could be formed by all intelligent cameras connected via a Local Area Network (**LAN**) in a district. The inference task of one camera at a busy crossroad can be distributed to other cameras where traffic is less heavy. Similar edge clusters can be found in smart home, smart drones, *etc.* Note, it is preferable for the edge cluster to be owned by the same entity. This helps mitigate certain security and privacy concerns. Alternatively, data encryption and secure transmission [19] can be implemented to ensure the privacy of the data. This approach of leveraging multiple edge devices enables us to effectively utilize the idle computing resources in the edge cluster where the workload is unevenly distributed.

To achieve this, we introduce **ParaLoupe**, a real-time video analytics system for inference on edge cluster. ParaLoupe makes use of mini model parallelization, where a set of collaborating edge devices each uses a mini model to infer a part of the task (*e.g.,* a small image patch[1]). Each mini model works like a small loupe to check details in parallel for a small magnified region of the original image. The idea of leveraging multiple cooperative devices for distributed inference has also been adopted by previous works via data partition [6], [7], [20] and model partition [21]–[24]. The key difference of ParaLoupe lies in that it introduces a novel object-centric patch generator to generate multiple single-object patches from the frame and designs a new set of task-oriented inference models that are both lightweight and accurate for the single-object patches. Inferring a small image patch instead of a large video frame enables finer scheduling of the task to take advantage of the limited computing resources available on edge devices. As a result, we show that this leads to a better accuracy and latency tradeoff.

Designing ParaLoupe presents several core technical challenges. First, it is difficult to design inference models that are both lightweight and accurate, due to the fundamental tradeoff of model size and accuracy [25]. Second, decomposing the task of inferring a single frame into multiple independently executable tasks is non-trivial. On the one hand, data partition-based approaches such as uniformly partitioning a video frame horizontally and vertically may result in objects being split across multiple patches, making accurate inference difficult [7]. On the other hand, model partition-based approaches work only for shallow and simple architectures, while not suitable for modern deep and complex models. Third, task scheduling is crucial to maximize the inference performance in the edge cluster, yet it is not straightforward to how best to allocate tasks in real time. To address these challenges, ParaLoupe first adopts an object-centric patch generator to break down a captured video frame into multiple single-object patches. Ideally, each single-object patch contains only one object to detect and the object accounts for more than 50% of the patch's size. Due to the simplicity of single-object patches, a much smaller model can be designed to accurately infer them. These generated patches are distributed to multiple edge devices for parallel inference by a task scheduling algorithm to efficiently leverage the available computing resources in the cluster.

The process of first locating single-object patches and then performing inference on each patch in ParaLoupe is conceptually similar to standard two-stage object detectors (*e.g.,* R-CNN [26], Fast-RCNN [27], Faster-RCNN [28]). In standard two-stage object detectors, anchors are generated by a Region Proposal Network (RPN) and further refined to obtain the final bounding boxes. However, there are several key differences between them. First, typical object detectors are designed to infer independent images on a single powerful device equipped with a GPU, while ParaLoupe is designed for inferring videos on a cluster of resource-constrained edge devices. Consequently, ParaLoupe must take into account the limited and distributed computing resources to achieve optimal performance. Second, RPN is designed to generate numerous anchors at predefined locations in the feature map that may contain objects, while our object-centric patch generator aims to produce a small number of image patches, each containing only one object. This prior knowledge of a single object in each image patch allows us to design more efficient models for edge devices. Additionally, RPN is implemented in computation-intensive CNNs, while our patch generator is more mobile-friendly by leveraging frame continuity in the video stream.

The main contributions of this paper are:

- We propose ParaLoupe, a real-time video analytics system based on mini model parallelization in the edge cluster. By parallelizing inference of object-centric patches with mini models on multiple edge devices, ParaLoupe achieves a better accuracy and latency tradeoff.
- We develop a set of task-oriented mini models using neural architecture search (**NAS**) to accurately infer single-object patches, and design an object-centric patch generator to efficiently produce them from video frames.
- We propose a task scheduling solution to distribute the patch inference task among edge devices, which fully utilizes the distributed computing resources.
- We implement ParaLoupe on a cluster of commercial edge devices including a Jetson, Raspberry Pi, and smart phone. Evaluation on the object detection task shows that ParaLoupe achieves up to 14.1× speedup with an accuracy drop of less than 4.6% compared to state-of-the-art (**SOTA**) models. Compared to SOTA models under the same latency constraints, ParaLoupe obtains 42.4%-45.1% accuracy improvements.

[1]A patch is a subset (usually rectangular) of an image.

(a) Original image with D7 (car: 29%) (b) Original image with D0 (missed) (c) Cropped image with D0 (car: 86%)

Fig. 2. An example image (before/after cropped) detected with EfficietDet-D7 and EfficientDet-D0.

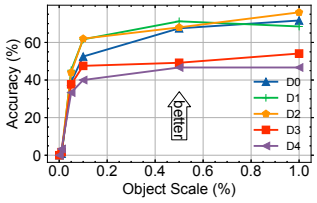

Fig. 3. Detection accuracy of D0-D4 on missed objects.

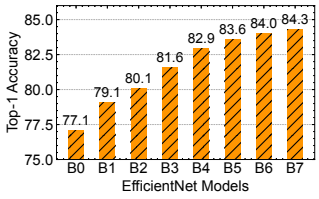

Fig. 4. Top-1 accuracy of B0-B7 (source: [31]).

II. Motivational Study

A. Background and Observations

It is widely believed that higher model complexity provides better generalizability and improved accuracy. For example, to detect objects at vastly different scales,[2] a neural network needs to be deep in order to capture multi-scale features [29]. However, if the object scale is large, shallow models are enough to detect objects accurately. To illustrate this, we evaluate the EfficientDet [30] series of object detectors. EfficientDet consists of 8 models ranging from EfficientDet-D0 to EfficientDet-D7, with D0 being the smallest and D7 being the largest and most powerful.

Fig. 2 displays the detection results for three images using either D7 or D0. The two images in Fig. 2a and Fig. 2b are the same, while the image in Fig. 2c is cropped and resized from Fig. 2a. We find that D7 correctly detects the car (Fig. 2a) although the object scale is small, whereas D0 fails to do so (Fig. 2b). The reason is that the model architecture of D0 is much shallower and simpler than D7, which fails to extract features of the car in the small region. However, D0 successfully detects the car in the cropped image (Fig. 2c) and even achieves a very high confidence score (86%). The reason is that the object scale in Fig. 2c is much larger than the other two, making it easier to be correctly detected.

To further evaluate the impact of object scale on detection accuracy, we retrieve the images in a sample video from the Yoda dataset (§IV-A3) where there are objects successfully detected by D7 but missed to detect by D0-D4. Each of these missed objects is cropped to be new images with varying object scales. Fig. 3 illustrates the detection accuracy of D0-D4 on these missed objects under varying object scales. We observe that more missed objects can be correctly detected by D0-D4 when the object scale increases from 0.001% to 1.0%. The detection accuracy under an object scale of 1.0% ranges from 46.67% (D0) to 76% (D4). This means that the objects in a frame can be accurately detected by simple models like D0 if cropped to a suitable object scale. Note that a scale of 1.0% is enough for D0 to detect the objects. By further increasing the object scale, shallower and simpler models than D0 can be used for accurate inference.

Observation 1: The accuracy of inference is contingent upon the object scale, and simple models can perform well on objects of large scale.

[2]The scale of an object refers to the object's size as a proportion of the overall image size.

Besides the object scale, input size also impacts the inference accuracy significantly. While larger input sizes generally result in higher inference accuracy, they also induce higher computational load. To mitigate the impact of object scale, we employ the image classification model and task to assess the influence of input size. The reason is that typically each image in the classification task usually contains a single target, and the object scales are relatively consistent. Fig. 4 presents the top-1 accuracy attained by the EfficientNet [31] series of models for image classification. Similar to EfficientDet, EfficientNet also consists of 8 models ranging from EfficientNet-B0 to EfficientNet-B7, with B0 being the smallest and B7 being the largest and most powerful. The target input size increases from B0 to B7. Unsurprisingly, accuracy gradually increases from B0 to B7. Fig. 5 further illustrates the computational complexity (in GFLOPs) of B0 to B7 with varying input sizes. It is shown that the complexity grows as the input size increases.

In practice, the smallest input size that yields correct inference results (*i.e.,* the optimal input size) is preferable. Experimentally we found that the optimal input size depends on the class of the object. Typically, classes of objects that require more detailed information (*e.g.,* textures) to distinguish from others have a larger optimal input size. Fig. 6 shows the accuracy of EfficientNet-B0 with varying input sizes for a set of test images. The images are also generated from the Yoda dataset (§IV-A3) and the object scale is set to 0.5 to control the complexity of images. We find that *traffic light* and *cab* are relatively easier to recognize, achieving high accuracy (85%+) even with small input sizes. In contrast, accurately classifying *moving van*, *minivan*, and *sports car* necessitates larger input sizes. This means that different input sizes should be set for different classes of objects in order to achieve high accuracy and low computational load simultaneously.

Observation 2: The optimal input size for inference varies among different classes of objects, leading to different computational loads.

B. Opportunities

Based on the aforementioned observations, we conclude that simple models can accurately infer objects of large scale. Under a fixed object scale, an appropriate input size should be selected to increase inference accuracy and minimize computational load. These two observations jointly lead to the design of task-oriented mini models, which are targeted on inferring

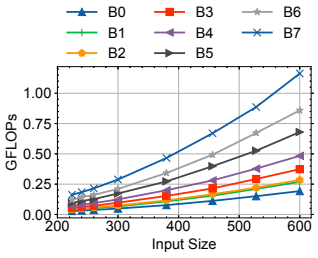

Fig. 5. Computational complexity of B0-B7.

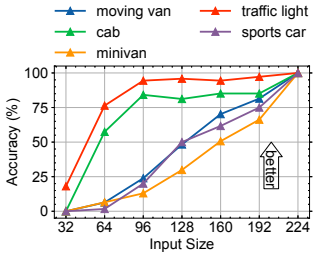

Fig. 6. Inference accuracy of EfficientNet-B0.

images with large object scale and tuned to a suitable input size. To generate the image patches with large object scale for accurately inference with mini models, we propose an object-centric patch generator to decompose a single frame into multiple single-object patches. To efficiently infer the single-object patches in the edge cluster, a task-scheduling strategy is proposed to leverage the available computing resources of edge devices to meet the service-level objective (**SLO**). These factors together inspire the design of ParaLoupe.

III. ParaLoupe: System Design

A. System Overview

The ParaLoupe framework is illustrated in Fig. 7. It comprises of three main components: the ① ***Task-oriented Mini Model Designer*** (Fig. 8); ② ***Object-centric Patch Generator*** (Fig. 9); and ③ ***Patch-based Task Scheduler*** (Fig. 11).

The *Task-oriented Mini Model Designer* generates the models offline in a remote cloud server before running the video analytics pipeline. The mini models are generated in two steps: (*i*) context-aware search to obtain the architecture of backbone part of the mini models, and (*ii*) task-specific inference to prepare the final mini models. The backbone is a feature extracting network specialized for simple inputs of large object scale, targeted on specific hardware and aimed at specific tasks (*e.g.,* detecting a specific class of objects). The identified backbone structure is used to further construct the complete model by adding remaining neural network layers, trained and optimized for inference on edge devices. Finally, these mini models are deployed on all edge devices in the cluster. At the same time, the accuracy as well as latency (on various hardware platforms) of the models are also profiled offline and saved for later task scheduling.

When a video frame is captured by the camera on an edge device, the *Object-centric Patch Generator* on the same edge device first utilizes a lightweight *Prediction Model* to identify Regions of Interest (**RoIs**) where objects might appear. The *Patch Proposer* initially generates high-confidence RoIs by slightly expanding the bounding boxes of previously detected objects, leveraging the frame continuity in videos. Following this, low-confidence RoIs are generated within the remaining regions using a predefined threshold. Subsequently, these patches undergo further refinement by the Patch Refiner to create the task queue of single-object patches. In this procedure, the *Object-centric Patch Generator* decomposes the task of inferring a single video frame into multiple smaller

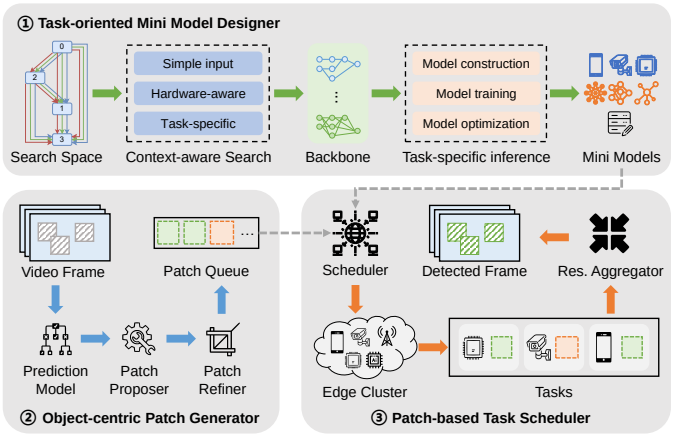

Fig. 7. System overview of ParaLoupe.

tasks that can be efficiently executed on multiple edge devices in parallel.

Given the single-object patches queue, the *Patch-based Task Scheduler* then determines which device and model a patch is sent to for inference, in order to efficiently leverage the distributed computing resources of edge clusters and simultaneously meet the SLO. The inference results are finally aggregated from edge devices to generate the ultimate detection results.

In our set-up, each edge device can initiate analytics tasks on its own captured frames, and also execute patch inference tasks from other edge devices. The *Object-centric Patch Generator* is deployed on each device in the cluster to balance workload across the cluster, while the *Patch-based Task Scheduler* is deployed on only one dedicated device for global optimization.

In the following sections, we use object detection as an example to explain the details of each component. Note that the same design principles can be applied to other vision tasks, such as keypoint detection (see results in §IV-H) and image segmentation. Table I summarizes the key symbols used in this paper.

B. Task-oriented Mini Model Designer

1) Defining inference complexity of images: The goal of task-oriented mini model designer is to generate models that are both lightweight and accurate for inference on edge devices. Before diving into the design details of mini models, we first introduce the inference complexity of images, which forms the core rationale behind the design of mini models. As discussed in *Observation 1*, simple models are enough to accurately infer images with large object scale. Therefore, the key to designing mini models is to increase the object scale in images and then design models accordingly. Based on the object scale, we first define the inference complexity O of an image I formally by the per-object area percentage, *i.e.,*

$$O(I) = 1 - \frac{\sum Norm.(obj\_area)}{\#obj}, O(I) \in [0, 1]. \quad (1)$$

where $Norm.(obj\_area) \in (0, 1]$ denotes the area of an object as a percentage of the total image size, and $\#obj$ represents the total number of objects in the image.

TABLE I
DEFINITIONS OF KEY SYMBOLS

Symbol	Definition
$O(I)$	Inference complexity of Image I
$Norm.(obj\_area)$	Normalized area of an object
$\#obj$	Total number of objects in the image
x_i, x_j	Latent representations
(i, j)	An edge connecting node i and j in DAG
$o^{(i,j)}$	An operation between node i and j in DAG
a	Architecture design of backbone
w	Weights of a
$\mathcal{L}_{\text{CE}}$	Cross-entropy loss
$\mathcal{L}_{D_{\text{train}}}, \mathcal{L}_{D_{\text{val}}}$	Loss on training and validation datasets
M_i	The i-th model during searching
α, β, γ	Hyperparameters controlling the weight
$\#operations$	Total number of operations within the cell
$FLOPS$	FLOPS of target hardware
$f_{edge}, f_{diff}, f_{hist}$	Features for RoI
M	The objectiveness prediction model
I_{prob}	Probability of objectiveness in the image
λ_1, λ_2	Hyperparameters controlling the threshold
d_i, m_i, p_i	The i-th device, model and patch
b_i	Bandwidth between client device and d_i
t_i^j	Computational latency of m_j on d_i
c_i^j	Computational cost of m_j on d_i
α_i^j	Accuracy of m_j on d_i
$x_{k,i}^j$	Binary task scheduling decision
$\mathcal{D}, \mathcal{M}, \mathcal{P}, \mathcal{X}$	Collection of d_i, m_i, p_i and $x_{k,i}^j$
C	Total number of server devices
N_i	Total number of models on d_i
K	Total number of single-object patches
s_k	Image size of p_k
y_k	Confidence level of p_k
T_{SLO}	Target latency in SLO
T_i	Total latency of patches assigned to d_i
$x_k, \tilde{x}_k$	Decision vector of p_k before/after update
$U, \tilde{U}$	Objective function value before/after update

Under this definition, larger per-object area percentage (which means fewer objects with larger total normalized area) indicates lower complexity. If an image has low complexity O, we call it simple; otherwise, we refer to it as complex.

In ParaLoupe, our goal is to design models specifically for images with inference complexity below 0.5. Lower complexity implies the need for more accurate object localization (which is challenging), while higher complexity requires the use of more complex models for inference, increasing computational burden. A complexity of below 0.5 means that an image should only contain one object, and its area percentage must be above 50%; otherwise, its complexity will be higher than 0.5. Therefore, we refer to images with complexity of below 0.5 as single-object images.

The generation of single-object images will be covered in §III-C. In the remaining of this section, we will introduce how to design lightweight and accurate mini models specifically for single-object images. In brief, the mini models are characterized by two aspects: (*i*) employing shallower architecture with a latency-aware searching objective in NAS to ensure optimal performance on edge devices, and (*ii*) trained with a specially pre-processed dataset of low complexity.

2) Context-aware search: The design of task-oriented mini models in ParaLoupe is accomplished through *Context-aware Search* (***CAS***) which searches the architecture of the backbone and *Task-specific Inference* (***TSI***) which prepare the final

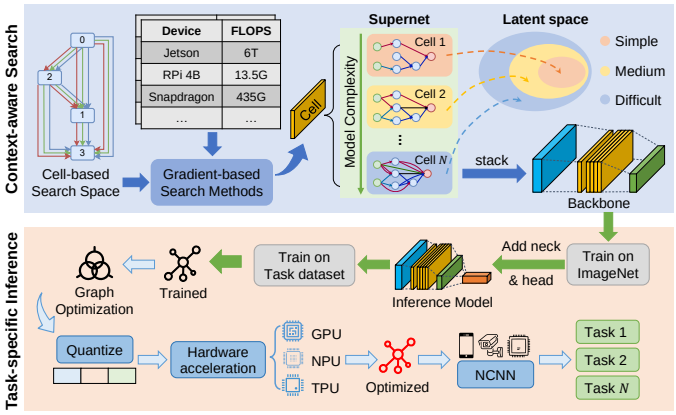

Fig. 8. The workflow of task-oriented mini model designer.

models. Fig. 8 shows the workflow of these two components (upper half for *CAS* and lower half for *TSI*). In this section, we introduce the key elements of *CAS*, while the next section explains *TSI*.

Typically, CNN models can be divided into the backbone, neck, and head parts. Among them, the backbone is responsible for feature extraction, the neck is for feature fusion and the head is for prediction generation. The architecture design of neural network affects the inference accuracy and latency of models. In ParaLoupe, *CAS* focuses on searching the architecture for backbone since it accounts for most of the computational overhead [32].

The backbone searching process involves two steps. First, we search for the architecture of basic cells (left half of *CAS* in Fig. 8). Second, we stack the basic cells to form the backbone (lower right of *CAS* in Fig. 8). In a CNN, a basic cell is a directed acyclic graph (**DAG**) consisting of a topologically ordered sequence of N nodes. Each node denotes a latent representation, x_i, to be learned. Each edge (i, j) denotes an operation $o^{(i,j)}$ (*e.g.,* convolution, average-pooling) that transforms x_j to compose x_i, *i.e.,* $x_i = \sum_{j<i} o^{(i,j)}(x_j)$. The collection of all candidate cell structures is called the *supernet*. The different structures in *supernet* correspond to image inputs of differing complexity (upper right of *CAS* in Fig. 8), *i.e.,* a more difficult input requires a more complex cell structure. Given the basic cells, the complete structure of the backbone can be obtained by stacking them together [33], and adding other necessary layers.

Denoting the architecture design of backbone as a, and the weights as w, the objective function during search is:

$$\min_{a} \mathcal{L}_{\text{CE}}\left(a, w^*\right) = \mathcal{L}_{\text{D}_{\text{val}}}\left(a, w^*\left(a\right)\right), \tag{2a}$$

$$\text{s.t. } w^*(a) = \arg\min_{w} \mathcal{L}_{D_{\text{train}}}\left(w, \text{M}_i\right), \tag{2b}$$

where $\mathcal{L}_{\text{CE(a,w*)}}$ denotes the cross-entropy loss of architecture a with weights w^*. $\mathcal{L}_{D_{\text{train}}}$ and $\mathcal{L}_{D_{\text{val}}}$ are the training and validation dataset. M_i denotes the i-th model during searching. By adopting the bi-level optimization [34] technique, Eq. (2a) and Eq. (2b) can be optimized alternatively to obtain the a^* and w^*.

In order to obtain the best architecture on the target hardware, we incorporate the computational load of each cell into the loss function during the search (the table of *CAS* in Fig. 8). Specifically, we estimate the inference latency using the number of operations in a cell and the **FLOPS** (floating point operations per second) of the target hardware. On the one hand, more operations performed in a cell result in higher inference latency. On the other hand, FLOPS measures the computation capability of the hardware device, with higher FLOPS indicates lower inference latency. To make the loss function differentiable, the inference latency is modeled as a weight, *i.e.,*

$$\mathcal{L} = \alpha \log\left(\frac{\#operations}{FLOPS}\right)^{\beta} \cdot \mathcal{L}_{\mathrm{CE}}\left(a, w^{*}\right), \tag{3}$$

where α and β are hyperparameters controlling the weight, $\#operations$ denote the number of operations in the specific cell structure, and $FLOPS$ represents the FLOPS of the target hardware. $\mathcal{L}_{\mathrm{CE}}\left(a, w^{*}\right)$ denotes the original cross-entropy loss as described in Eq. (2a). The backbone search process is performed on the ImageNet [35] dataset where each image contains only one object to find the best model for single-object patches.

3) Task-specific inference: *CAS* is used to obtain the architecture of backbone suitable for single-object patches and target hardware. After obtaining the backbone, we must construct the complete model for the specific inference task and further optimize it for acceleration.

As the backbone is an important component of the model, we first train it on the ImageNet dataset to make it easier to transfer to other tasks (upper right of *TSI* in Fig. 8). Taking advantage of ImageNet's diversity and size, the backbone can effectively extract features from the images. Once the backbone has been trained, we add the neck and head parts to obtain the complete inference model. Specifically, we reduce the input size (according to *Observation 2*) and the number of proposals of the model (to detect only objects of large scale) to best fit the complexity and category of input images. Having obtained the ultimate model structure, we train it on the task dataset (*e.g.,* the MS COCO dataset [36] for object detection) and select the best model for deployment.

It is noteworthy that the task dataset for final model training is processed to be of low complexity. Specifically, given an image, we first conduct object detection to identify all objects present in the image. Following that, we crop each object into a new image patch by expanding the detected bounding box. We ensure that the proportion of the object size is $r \in [0.5, 1)$ of the total size of the new image. The new generated image patches are then used as the task dataset to train the final model.

To further reduce the computational load of the model, we perform graph optimization as well as quantization (INT8 and FP16) on the model. The ultimate model is deployed with NCNN [37] on various hardware using corresponding acceleration techniques (lower half of *TSI* in Fig. 8).

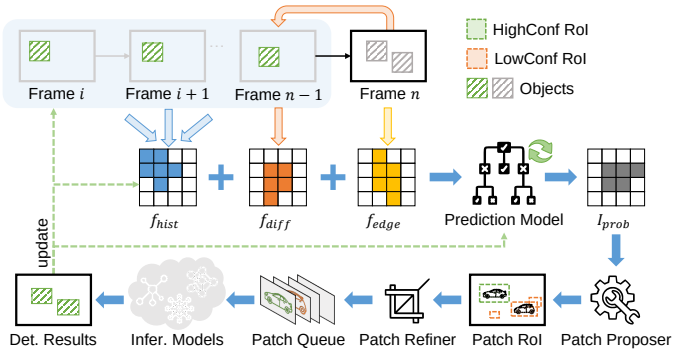

Fig. 9. Workflow of the object-centric patch generator.

C. Object-centric Patch Generator

1) Design Principles: We next describe the object-centric patch generator, which is essential to achieve high inference accuracy for mini models. The goal of object-centric patch generator is to identify RoIs with high objectness and crop image patches from a video frame to achieve low complexity (*i.e.,* $O < 0.5$) for the patches. This means that ideally an image patch should contain only one object and it accounts for over 50% of the total image size. While previous works have addressed RoI localization in images, such as localizing with motion vectors [12] or LSTM models [7], the novelty of object-centric patch generator in ParaLoupe resides in its ability to generate single-object patches of low complexity. The workflow of the object-centric patch generator is illustrated in Fig. 9.

To generate single-object patches from a given frame efficiently and accurately, the patches are identified in two confidence levels: high confidence and low confidence. High confidence patches (referred to as *HighConf RoI*) signify objects previously detected in preceding frames. In video analytics, an essential feature is the minimal variance between consecutive frames, known as the frame continuity or frame-to-frame coherence. Utilizing the frame continuity, these HighConf RoI can be readily generated by slightly expanding the previously detected regions. Conversely, a low confidence patch (*LowConf RoI*) corresponds to an object appearing for the first time in the current frame. These objects are significantly more challenging to localize accurately, especially in the context of a moving camera. To generate low confidence patches, ParaLoupe exploits features indicative of RoIs and trains a lightweight *Prediction Model* to learn the mapping from features to the probability of objectness. We elaborate on this process in the following section.

2) Indicative features for RoIs: We first extract indicative features for predicting RoIs that may contain objects. In ParaLoupe, we use three features: low-level image features (f_{edge}), pixel-to-pixel differences with the last frame (f_{diff}), and historical object location distributions (f_{hist}). These features (middle of Fig. 9) relate to the scale of the current frame, the previous frame, and all preceding frames respectively.

First, low-level image features can be used to identify objects effectively. In ParaLoupe, we use the feature f_{edge}, which measures the texture of images to predict the presence

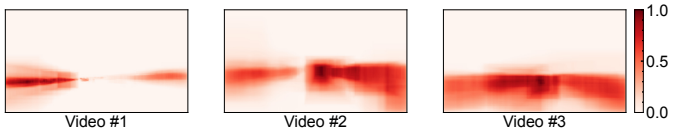

Fig. 10. Heatmap of objectness probability for three videos.

of objects. Typically regions with fewer edge patterns are less likely to contain objects.

Second, pixel-to-pixel differences with the last frame (f_{diff}) are used. This is an effective way to discard redundant RoIs that are the same as those in the previous frame. By skipping regions that remain unchanged, computational resources can be saved. This is especially useful when the camera is static.

Third, a historical object location distribution feature (f_{hist}) is used. This is able to reveal the probability of objects occurring in different locations throughout the video. Typically, objects tend to occur in specific regions of the video frame, even for moving cameras. Fig. 10 shows the object occurring probability heatmap of each pixel for three dashcam videos (each with 1800 frames) from the Yoda dataset (§IV-A3). The darker the pixel is, the more objects have appeared in that region. For these three videos, we observe that most objects appear in the lower half of the video, which need to be paid more attention to when generating patches.

3) Design of prediction model: Based on the three aforementioned features, we design a lightweight model, M, to learn a mapping from features to the probability of objectness, *i.e.,* $I_{prob} = M(f_{edge}, f_{diff}, f_{hist})$. As the prediction model is executed on the edge device, it must be lightweight and capable of updates over time. In ParaLoupe, we utilize Logistic Regression [38] (although other models could be employed) as the prediction model to learn the probability of objectness for each pixel.

4) RoI postprocessing: After the probability of object occurrences is calculated for each pixel, we generate RoIs and apply further processing to improve inference efficiency and accuracy (bottom of Fig. 9). First, the *Patch Proposer* generates HighConf RoI by slightly enlarging the bounding boxes of objects detected in the most recent frame. Subsequently, using a predefined threshold, the Patch Proposer identifies the remaining regions with high likelihood and segments them into independent connected components (*i.e.,* distinct clusters of pixels connected to each other, yet separated from other clusters). However, these partitioned patches (Patch RoIs) could become excessively large (leading to high complexity) or overlap significantly with existing patches (resulting in unnecessary computational overhead). To mitigate these issues, the *Patch Refiner* further refines the patches, removing those whose area percentage exceeds a certain threshold λ_1 (*e.g.,* 0.5), or whose Intersection over Union (**IoU**) with existing patches surpasses threshold λ_2 (*e.g.,* 0.3).

The patch cropping results are then forwarded to the *Patch Queue*, which stores the *Patch Image* and *Patch Metadata*. This task queue is subsequently distributed by the patch-based task scheduler to infer across the edge cluster for inference. Note that *Patch Metadata* includes information about the frame

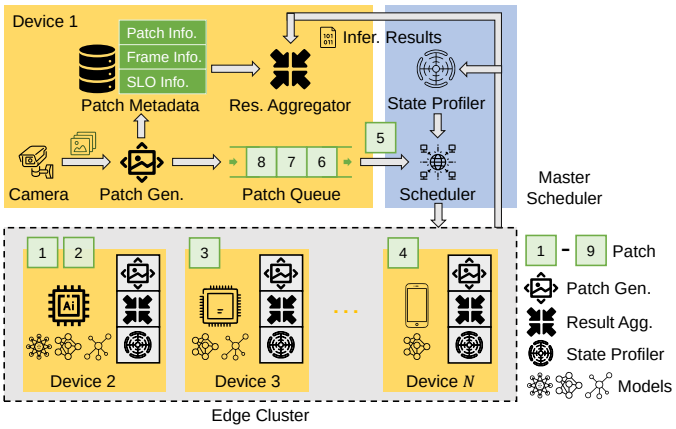

Fig. 11. Framework for the patch-based task scheduler.

(*i.e.,* the number of patches generated and the original frame size) and the patch (*i.e.,* the original location of the patch in the frame and the confidence level), whereas the *Patch Image* includes the actual image to be inferred.

D. Patch-based Task Scheduler

The patch-based task scheduler's responsibility is to determine which device and model each patch is sent to for inference. Unlike the object-centric patch generator which is deployed on each edge device to balance the workload across the cluster, the patch-based task scheduler is deployed on only one dedicated device (which we call *master scheduler*) in the cluster. The reason to use a centralized scheduler is two-fold: (*i*) In our application scenario, all the edge devices in a cluster are usually owned by one entity (*e.g.,* intelligent cameras on road are owned by the government in the district), and the network connection between them is usually stable with a high bandwidth limit; (*ii*) A centralized scheduler makes it easier to monitor the overall system status and optimize towards the entire cluster's resources for better performance.

In order to manage the global state of edge clusters, a system state profiler is deployed on each edge device. This profiler is designed to send information regarding network conditions, device computational capabilities, and model accuracy to the master scheduler for scheduling purpose. The details of system state profiler will be cover shortly (§III-D1). Additionally, a result aggregator is deployed on each edge device to collect and combine inference results obtained from the respective edge devices. The framework for the patch-based task scheduler is shown in Fig. 11.

In ParaLoupe, we consider a scenario where each device in the edge cluster has the capability to initiate requests for analytics tasks (*i.e.,* send tasks to other devices for inference) and also execute inference tasks from other devices or itself. For simplicity, we refer to the device initiating the analytics task request as the *client device* and the device performing the inference task as the *server device*. Note that the client device only needs to send patch metadata (*e.g.,* patch size, confidence level) to the master scheduler. The master scheduler will then respond with information about the designated server device and corresponding model to handle the inference task.

After that, the client device will directly send patches to server devices and collect inference results from them. Hence, the bandwidth consumption between the client device and the master scheduler is minimal.

1) System State Profiler: Three profilers are deployed on each edge device to monitor the state of the cluster, and provide information to the master scheduler for the global optimization of scheduling. They are: the (*i*) *Network Profiler*, (*ii*) *Device Profiler*, and (*iii*) *Model Profiler*. Hereafter, we use a subscript i to indicate the i-th device and superscript j to indicate the j-th model.

The *Network Profiler* monitors the available bandwidth b_i between the client device and the i-th server device, which is used to model the transmission latency. The available bandwidth is measured by *iperf* [39]. Note that the available bandwidth is measured between each pair of devices, instead of between each device and the master scheduler. The reason is that the transfer of patch images between the client device and server devices typically consumes the most bandwidth. To avoid too much overhead in evaluating the available bandwidth between every two devices when the cluster size is large, each client device consistently updates the bandwidth only with n (*e.g.,* $n = 10$) devices with the highest available bandwidth. The available bandwidth b_i is measured and updated periodically (*e.g.,* every 15 seconds). When scheduling is required for a patch, the available bandwidth is transmitted together with the patch metadata by the client device to the master scheduler for decision-making.

For the *Device Profiler*, two key parameters are monitored: the computational latency, t_i^j, and the computational cost, c_i^j, of the j-th model on the i-th device. The computational latency is considered fixed for a specific device and model, thus only requires profiling once. The computational cost is designed to measure a device's willingness to perform inference, which can be influenced by factors such as the workload of analytics tasks and sensitivity to energy consumption. Higher computational cost indicates the device should be allocated fewer patch inference tasks. For instance, an intelligent camera at a busy crossroad has a higher computational cost than another camera where the traffic is less heavy, and a battery-powered smart phone has higher computational cost than a Jetson device with consistent power supply. Note that the computational cost is dynamic, *i.e.,* it varies from time to time. In ParaLoupe, we measure the computational cost of a device by the percentage of video frames requiring inference over the past m minutes (*e.g.,* $m = 5$). The reason is that the workload of analytics tasks is usually unevenly distributed across the cluster, *e.g.,* most of the frames can be skipped via frame difference in a camera with no traffic.

Finally, the *Model Profiler* logs the accuracy α_i^j of the j-th model on the i-th device. Similar to the computational latency, this information is also fixed and can be obtained beforehand. These profilers collectively enable ParaLoupe to be scalable and robust in dynamic edge computing scenarios. The network profiler considers dynamic network conditions, while the device profiler takes into account heterogeneous device capabilities. Note that in ParaLoupe, we focus on designing the entire system and defer more advanced modeling and solutions [40] to future work.

2) Problem Formulation: The goal of the master scheduler is to determine which device and model each patch is sent to for inference based on the patch metadata and system states. Given C candidate server devices (denoted as $\mathcal{D} = \{d_1, d_2, \cdots, d_C\}$) and N_i models (denoted as $\mathcal{M}_i = \{m_1, m_2, \cdots, m_{N_i}\}$) deployed on the i-th server device d_i, the master scheduler needs to dispatch K single-object patches (denoted as $\mathcal{P} = \{p_1, p_2, \cdots, p_K\}$) generated from a video frame F to determine which device and model is selected to infer the patch. The k-th patch is of size s_k and confidence level y_k (high or low confidence, as in §III-C). Our objective is to maximize the inference accuracy within the latency constraints (T_{SLO}) on a per-frame basis. To maximize the average achieved utility of all single-object patches in a given video frame, the problem can be formulated as:

$$\mathbf{P}: \max_{\mathcal{X}} \frac{1}{K} \sum_{p_k \in \mathcal{P}} \sum_{d_i \in \mathcal{D}} \sum_{m_j \in \mathcal{M}} x_{k,i}^j y_k \left(\alpha_i^j - \gamma c_i^j\right) \tag{4a}$$

$$\text{s.t. } \max\{T_1, T_2, \cdots, T_C\} \leq T_{SLO} \tag{4b}$$

$$\sum_{d_i \in \mathcal{D}} \sum_{m_j \in \mathcal{M}} x_{k,i}^j = 1, \ \forall p_k \in \mathcal{P} \tag{4c}$$

$$x_{k,i}^j \in \{0, 1\}, \forall p_k \in \mathcal{P}, \ d_i \in \mathcal{D}, \ m_j \in \mathcal{M}, \tag{4d}$$

where T_i is the total latency (including transmission latency and computation latency) assigned to the i-th device, defined as $T_i = \sum_{p_k \in P} \sum_{m_j \in M_i} x_{k,i}^j (\frac{s_k}{b_i} + t_i^j), \forall d_i \in \mathcal{D}$, $\mathcal{M} = \{\mathcal{M}_i | \, d_i \in \mathcal{D}\}$, $\mathcal{X} = \left\{ x_{k,i}^j \middle| \, p_k \in \mathcal{P}, d_i \in \mathcal{D}, m_j \in \mathcal{M} \right\}$. $x_{k,i}^j$ is a binary variable representing the task scheduling decision for the k-th patch p_k, with $x_{k,i}^j = 1$ indicating that the j-th model m_j on device d_i is assigned for p_k. s_k is the image size of the k-th patch. b_i is the available bandwidth between the client device and the i-th server device. t_i^j denotes the computational latency of the i-th model on the j-th device. y_k is the confidence level of the k-th patch. The parameter $\gamma > 0$ controls the tradeoff between inference accuracy α_i^j and computational cost c_i^j. Including computational cost c_i^j as part of the objective function is aimed at scheduling patch inference tasks onto devices with lower computational costs, promoting load balancing among devices within the cluster. T_{SLO} denotes the target latency in SLO. The constraint (Eq. 4b) ensures that the total latency of processing all single-object patches in a frame does not exceed the given latency constraint T_{SLO}. The constraints in Eq. 4c and Eq. 4d ensure that each patch is assigned to only one model on one device. Note that one device can receive multiple patches for inference, in order to fully leverage the available computing resource. For example, in Fig. 11, *Patch 1* and *Patch 2* are scheduled for inference on *Device 2* at the same time, while other devices receive only one patch.

The formulated problem $\mathbf{P}$ is similar to the Multidimensional Multiple-choice Knapsack Problem [41], which has been proven to be NP-hard, making it difficult to obtain the optimal solution in polynomial time. The total number of candidate task scheduling decisions is 2^n, where $n = K \sum_{i=1}^{C} N_i$. Thus, an exhaustive search method is not feasible.

Algorithm 1: Markov-based Model Selection

Input: $\mathcal{P}$ for a given frame, $\mathcal{D}$, $\mathcal{M}$, T_{SLO}, γ, $\{y_k, s_k | p_k \in \mathcal{P}\}$, $\{b_i | d_i \in \mathcal{D}\}$ and $\{\alpha_i^j, c_i^j | d_i \in \mathcal{D}, m_j \in \mathcal{M}\}$

Output: Task scheduling decision $\mathcal{X}$

$\mathcal{X} \leftarrow$ Random initialization;

repeat

  Randomly select a patch p_k and change its task scheduling decision vector x_k to $\tilde{x}_k$;

  if *new decision vector* $\tilde{x}_k$ *is feasible* **then**

    Obtain $\rho \leftarrow \frac{1}{1+e^{\frac{U-\tilde{U}}{\beta}}}$;

    With probability ρ, update p_k to the new decision vector $\tilde{x}_k$, otherwise unchanged;

until *stop condition is reached*;

In ParaLoupe, we utilize Markov approximation to obtain task scheduling decisions, which we will cover in the next section.

3) Algorithmic Solution: The Markov approximation method has been proven to be near-optimal for model selection [42], [43]. In this section, we leverage Markov approximation to obtain high-quality task scheduling decisions with a low overhead.

We initialize the task scheduling decision $\mathcal{X}$ by randomly assigning each patch to a model on a device subject to constraints Eq. 4b. Let $x_k = \left\{x_{k,1}^1, x_{k,1}^2, \cdots, x_{k,C}^{N_C}\right\}$ denote the task scheduling decision vector of p_k. In each iteration, we randomly select a patch and update its task scheduling decision vector while keeping others unchanged, which gives us a new decision vector $\tilde{x}_k$ and a corresponding objective function value $\tilde{U}$ (U is the objective function value for the old task scheduling decision $\mathcal{X}$). After that, the task scheduling decision, vector x_k of p_k, is updated to $\tilde{x}_k$ with probability $\rho = \frac{1}{1+e^{\frac{U-\tilde{U}}{\beta}}}$ [43]. $\beta > 0$ controls the degree of randomness. The iteration stops when T_{max} rounds are reached, or no significant improvements (*e.g.,* $|U - \tilde{U}| < 0.01$) are found in N (*e.g.,* $N = 10$) iterations. In this approach, we are able to efficiently find a high-quality task scheduling solution with a reduced search space.

The process of our proposed Markov-based model selection is shown in Algorithm 1. Given a collection of single-object patches $\mathcal{P}$ generated from a video frame, a task scheduling decision $\mathcal{X}$ is obtained to distribute each patch to a specific model on a device. This algorithm iteratively refines the task scheduling decision based on a Markov approximation, ensuring efficient allocation of computational resources.

4) Algorithmic Complexity: Finally, we analyze the complexity of the algorithm. In each iteration, the system randomly selects a patch to update its task scheduling decision vector. The total number of all feasible decisions can be enumerated up to $n = K\sum_{i=1}^{C} N_i$. Assuming that L iterations are required in order to converge, the time complexity is $\mathcal{O}(Ln)$.

IV. Evaluation

A. Experimental Setup

1) Hardware Setup: To represent real-world edge clusters, we utilize three types of commercial edge devices: one NVIDIA Jetson Xavier NX (Ubuntu 18.04), two Raspberry Pi 4B (Ubuntu 20.04) and one XiaoMi 11 Lite (Android 12, with a Qualcomm SM7150 Snapdragon 732G chipset). In addition, a Dell Precision 7920 Tower (Ubuntu 20.04, equipped with a NVIDIA GeForce RTX 3090 GPU) is used as the remote cloud server for searching and training mini models. The edge devices are connected via a Tenda F3 router (Wi-Fi 4, 802.11n, 300Mbps). The hardware setup is shown in Fig. 12.

2) Software Setup: We develop a C++ prototype of ParaLoupe for easy deployment on a variety of hardware platforms. To make it mobile-friendly, the C++ library is wrapped with Java Native Library (JNI) as an Android application. For deep learning inference on all platforms, we use NCNN [37], a high-performance neural network inference computing framework optimized for mobile platforms. For communication between devices, we use ZeroMQ [44], an asynchronous messaging library widely used in distributed systems.

3) Task and Dataset: For evaluation purpose, we consider the object detection task and detect two classes of objects: *vehicle* and *person* as in previous works [6], [45]. We use F1 score as the evaluation metric. To calculate it, we first compute the IoU between the real and predicted object bounding box. We then convert the IoU into a binary value, with a threshold of 0.5 if not stated explicitly otherwise, *i.e.,* if over 50% of the predicted box overlaps with the ground truth, we consider it a match. The selection of an IoU thresholds of 0.5 is consistent with previous works [46]–[48]. This is then used to calculate the F1 score.

We use datasets from two publicly available sources representing real-world scenarios: (*i*) We use the highway traffic and dashcam videos from the Yoda dataset [49]; and (*ii*) We use video sequences from the Udacity Self Driving Car Dataset [50]. The two datasets contain over 15 hours of videos. We combine the results of the two datasets in the following experiments unless otherwise stated explicitly. For all images, we use the detection results of YOLOv7 [51], a SOTA real-time object detector, as the ground truth.

4) Comparison Baselines: ParaLoupe is the first to parallelize the model inference on multiple edge devices by customizing both the data (*i.e.,* generating single-object patches) and model (*i.e.,* designing task-oriented mini models). Traditional approaches merely focus on either partitioning the data (*e.g.,* [6], [7], [20]) or partitioning the model (*e.g.,* [21], [23], [24]). For performance comparison with existing methods, we compare ParaLoupe with various inference models (§IV-B, §IV-C1), various data partition methods (§IV-C2, §IV-D) and various scheduling strategy (§IV-E). We also show the sensitivity of ParaLoupe to the key parameters (§IV-C3, §IV-C4, §IV-C5), impact of network variation (§IV-F), its system overhead (§IV-G) and the generalizability to other tasks (§IV-H). Note that the design space of ParaLoupe is orthogonal to model-related optimizations such as model splitting, distillation and pruning. Therefore, these optimizations are not

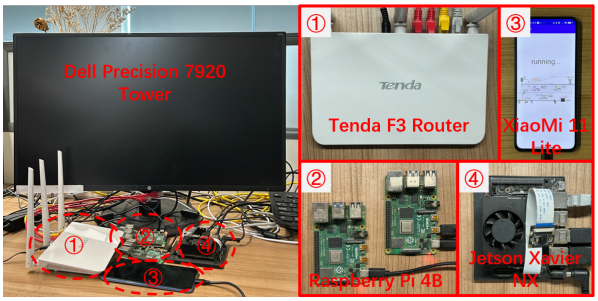

Fig. 12. Our experimental hardware setup.

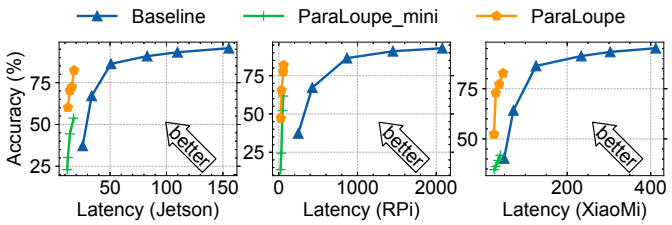

Fig. 13. Tradeoff between accuracy and latency for *Baseline*, *ParaLoupe_mini* and *ParaLoupe*.

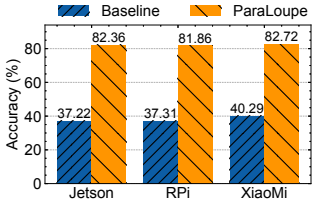

Fig. 14. Accuracy improvements under the same latency.

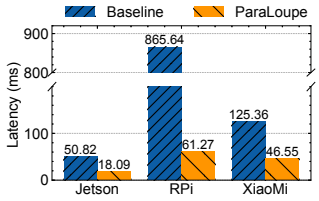

Fig. 15. Latency improvements with comparable accuracy levels.

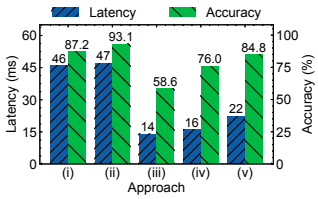

Fig. 16. Latency and accuracy achieved by various approaches.

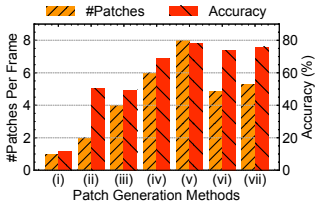

Fig. 17. #Patches generated and accuracy with various methods.

incorporated during the evaluation, as they can be applied to the mini models within ParaLoupe to further enhance its performance in a synergistic manner.

B. Overall Performance

1) Tradeoff between accuracy and latency: We first present the tradeoff between accuracy and latency for ParaLoupe. Fig. 13 displays the accuracy-latency tradeoff of ParaLoupe as well as two other approaches on Jetson Xavier NX (***Jetson***), Raspberry Pi 4B (***RPi***), and XiaoMi 11 Lite (***XiaoMi***). *Baseline* refers to the straightforward approach of inferring the original video frame on the edge device with the YOLOv7 model. *ParaLoupe_mini* represents inferring the original video frame with the task-oriented mini models in ParaLoupe. *ParaLoupe* shows the results of inferring generated single-object patches with the task-oriented mini models. Each point on the curves denotes the achieved averaged accuracy with the averaged inference latency (on the patch basis) for all videos. The tradeoff between accuracy and latency is achieved by varying the input size of the inference model.

On *Jetson*, *Baseline* with the SOTA model of YOLOv7 achieves high accuracy (95.6%) but with a relatively high latency (155.3ms). By inferring with *ParaLoupe_mini*, the latency drops to less than 17.7ms, but the accuracy also decreases significantly to less than 53.8%. This is because the mini models in *ParaLoupe_mini* is specifically designed for single-object patches and is not suitable for the original frames with multiple objects. By inferring generated single-object patches in *ParaLoupe*, the accuracy increases to as high as 82.4% while the latency remains low ($\leq$18.1ms). Overall, *ParaLoupe* achieves the best accuracy-latency tradeoff compared to the other two approaches.

It is important to note that the primary advantage of *ParaLoupe* lies in its ability to achieve very low latency while maintaining satisfactory accuracy. In this experimental setting, we only use task-oriented mini models for inference within the *ParaLoupe* approach. Consequently, the highest achievable accuracy of *ParaLoupe* is inherently lower than that of *Baseline*. However, in practical scenarios, *ParaLoupe* can adapt its inference strategy based on latency constraints. For example, when faced with higher latency constraints, *ParaLoupe* can selectively use larger models such as YOLOv7 for inference on certain frames, potentially enhancing its accuracy beyond the limitations of task-oriented mini models alone.

2) Accuracy improvements: To demonstrate the accuracy improvements of *ParaLoupe*, Fig. 14 shows the accuracy of *Baseline* and *ParaLoupe* under the same latency constraints (same experimental settings as in §IV-B1). The latency constraints are set as 30ms (*Jetson*), 250ms (*RPi*) and 50ms (*XiaoMi*), respectively. The accuracy of the same approach (*Baseline* or *ParaLoupe*) is similar across different hardware platforms. Under the same latency constraints, the accuracy improvement of *ParaLoupe* compared to *Baseline* is 45.1% (*Jetson*), 44.6% (*RPi*) and 42.4% (*XiaoMi*), respectively.

3) Latency improvements: Fig. 15 shows the latency of *Baseline* and *ParaLoupe* with comparable accuracy levels ($\geq$82%). Experimental settings are the same as in §IV-B1. Specifically, the accuracy drop of *ParaLoupe* compared to *Baseline* is 3.9% (*Jetson*), 4.6% (*RPi*) and 3.6% (*XiaoMi*). From the results, we observe that the latency reduction of *ParaLoupe* is substantial. Compared to *Baseline*, *ParaLoupe* achieves a latency speedup of 2.8$\times$ (*Jetson*), 14.1$\times$ (*RPi*) and 2.7$\times$ (*XiaoMi*), respectively.

C. Effects of Key Components

1) Sources of improvements: To examine the sources of improvements for ParaLoupe, Fig. 16 shows the latency (per-patch) and accuracy of five approaches. The five approaches are: (*i*) *Baseline-320-S1*: YOLOv7 model with input size of 320 in one patch (*i.e.,* no partition); (*ii*) *Baseline-320-S4*: YOLOv7 model with input size of 320 in four patches

(*i.e.,* evenly partitioned in half on the horizontal and vertical direction); (*iii*) *ParaLoupe-160-S4*: task-oriented mini models in ParaLoupe with input size of 160 in four patches; (*iv*) *ParaLoupe-160-PG*: task-oriented mini models in ParaLoupe with input size of 160 and enabled with object-centric patch generator. (*v*) *ParaLoupe-240-PG*: same as *ParaLoupe-160-PG* but the input size is 240. These approaches reflect the effects of naive even partitioning (*i and* ii), benefits of our designed mini models (*ii and* iii), improvements from object-centric patch generator (*iii and* iv), and further performance boost (*iv and* v).

Baseline approach (*i*) achieves 87.2% accuracy with a latency of 46ms. After partitioning the input image into four patches in approach (*ii*), the accuracy increases to 93.1%. The reason is that more small objects are correctly detected. In approach (*iii*), we replace the SOTA model with task-oriented mini models in ParaLoupe, and the latency decreases to just 14ms but the accuracy also drops significantly. By enabling the object-centric patch generator to produce single-object patches in approach (*iv*), the accuracy improves to 76.0%. Finally, we enlarge the input size of mini models from 160 to 240 in approach (*v*) to further boost the accuracy to 84.8% with a latency of 22ms. Compared to the baseline approach (*i*), ParaLoupe achieves similar accuracy (only 2.4% drop) but with 2.1× inference speedup.

2) Varying patch generation methods: To evaluate the effectiveness of our proposed object-centric patch generator, Fig. 17 presents the number of patches generated per frame and the accuracy attained on average for seven methods, which are: (*i*) *EP-1*, (*ii*) *EP-2*, (*iii*) *EP-4*, (*iv*) *EP-6*, (*v*) *EP-8*, (*vi*) *lastDet*, and (*vii*) *ParaLoupe*. The *EP* denotes equal partition. *EP-1* represents a single patch (*i.e.,* no partition); *EP-2* refers to equal partition in two patches, and so forth. In addition, *lastDet* indicates the utilization of solely the patches generated from the latest detection results, while *ParaLoupe* denotes our proposed object-centric patch generator. The inference model is the mini model and the target input size is set to 160.

The results show that accuracy is highest (78.0%) when the original frame is partitioned into 8 patches (*EP-8*). Nevertheless, the number of patches also increases linearly with the number of partitions, resulting in a heavy computational burden overall. Approach (*vi*) *lastDet*, which exclusively utilizes high-confidence patches to leverage temporal locality, generates 4.9 patches per frame on average with an accuracy of 74.0%. By incorporating low-confidence patches, *ParaLoupe* boosts accuracy to 76.0% (which is comparable to *EP-8*) but with only 5.3 patches generated per frame. Our proposed object-centric patch generator generates patches that facilitate high inference accuracy, as evidenced by these results.

3) Impact of number of patches: ParaLoupe parallelizes the inference with mini models by generating multiple single-object patches from a frame. While generating more patches results in higher overall accuracy, it may also increase the computational load. Fig. 18 presents the per-frame inference latency of YOLOv7 and ParaLoupe on three types of hardware. Specifically, *ParaLoupe (mean)* represents the average latency across all patches in a frame, akin to parallel inference by multiple devices. Conversely, *ParaLoupe (sum)* signifies the

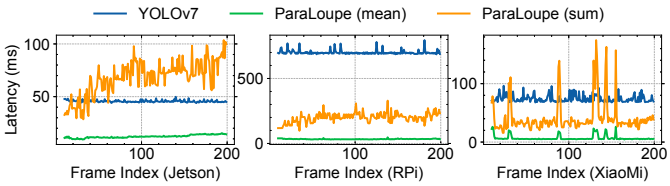

Fig. 18. Per-frame inference latency of YOLOv7, ParaLoupe (mean) and ParaLoupe (sum).

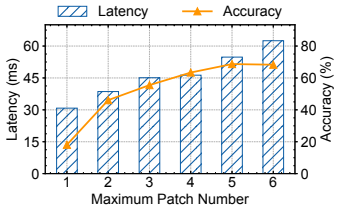

Fig. 19. Achieving latency SLO with maximum patch number per frame.

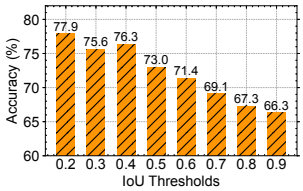

Fig. 20. Accuracy under various IoU thresholds.

summed latency across all patches in a frame, resembling sequential processing by a single device. Our experimental settings are the same across different hardware and only one device is used for inference. We observe that the average per-frame inference latency of ParaLoupe is always much lower than YOLOv7 since the mini models have lower computational costs. However, the summed per-frame latency of ParaLoupe can be higher than YOLOv7 when too many patches are generated. The tie-break point of per-frame latency for YOLOv7 and ParaLoupe on a specific device depends on the hardware, and more resource-constrained hardware results in better improvements by ParaLoupe. For example, ParaLoupe always outperforms YOLOv7 on RPi, and for most of the time on XiaoMi. Additionally, the latency variance of ParaLoupe (sum) across different frames is larger than YOLOv7 and ParaLoupe (mean). The reason is that YOLOv7 and ParaLoupe (mean) measure the latency of a single frame/patch, while ParaLoupe (sum) measures the total latency of multiple patches within a frame. Hence, an increase in the number of patches generated in a frame leads to higher total latency. These results show the superiority of ParaLoupe over baseline methods in resource-constrained settings.

4) Achieving different latency SLOs: Real-time video analytics usually has stringent latency requirements to achieve the target quality of service. The SLO we focus on in this paper is the end-to-end latency, defined as the time difference from when a frame is captured to when the analysis results are obtained. The latency SLO varies depending on the business domain, typically ranging from 20ms to 3 seconds [52]–[57]. For example, an AR application requires at most 20ms of latency to avoid simulation sickness [58]. In contrast, a trigger-based sensing application, such as drowsiness detection, requires consistent latency of under 1 second to give users enough time to react [59].

For *ParaLoupe*, several techniques can adjust the end-to-end latency to consistently meet the latency SLO. These techniques include adjusting the input size, distributing the

inference workload to different number of edge devices, and controlling the maximum number of single-object patches generated within a frame. To demonstrate *ParaLoupe*'s ability to achieve different latency SLOs, Fig. 19 shows the end-to-end latency and accuracy of *ParaLoupe* with various maximum number of single-object patches allowed to generate per frame. In this experiment, we use *Jetson* as the sole server device for inference with the task-oriented mini models. From the results, we observe that a smaller maximum patch number leads to lower latency and lower accuracy. With six single-object patches allowed to generate per frame, the end-to-end latency is 62.5ms with an accuracy of 68.3%. With one single-object patch allowed to generate per frame, the latency drops to 30.8ms and accuracy drops to 18.1%. These results demonstrate *ParaLoupe*'s capability to consistently adhere to the latency SLO with different latency-accuracy tradeoffs.

5) Impact of IoU thresholds: In §IV-A3, we set an IoU threshold of 0.5 to calculate the F1 score for accuracy measurement. To show the impact of IoU threshold, we vary the IoU threshold from 0.2 to 0.9, and the accuracy is shown in Fig. 20. In this experiment, we use *Jetson* as the sole server device for inference with the task-oriented mini models. The results show that the accuracy drops with increasing IoU thresholds. The reason is that higher IoU thresholds require a predicted bounding box to be closer to the ground truth to be considered a match.

D. Performance Boost with Predefined RoIs

The object-centric patch generator (§III-C) is capable of generating both high-confidence patches based on the latest detection results, as well as low-confidence patches based on the features of the given frame. In order to further boost the accuracy, predefined RoIs can be added to generate patches in specific regions. For fixed cameras, the RoIs can be set to areas where new objects or small objects tend to appear. Fig. 21 presents the number of patches generated per frame (Fig. 21a) as well as the accuracy (Fig. 21b) with various numbers of predefined RoIs added (from 0 to 6) for a surveillance video from the Yoda dataset. The inference model is our designed mini models and we test four different input sizes (*i.e.,* 160, 200, 240, and 320). As a result of including more predefined RoIs, more patches are generated (Fig. 21a), and the inference accuracy also increases (Fig. 21b). By setting the predefined RoIs properly, lower input size (*e.g.,* accuracy is 80.6% under input size of 160 with 6 predefined RoIs) suffices to achieve comparable accuracy as large input size (*e.g.,* accuracy is 76.7% under input size of 320 with 3 predefined RoIs), which further helps to reduce latency.

E. Scheduling Performance in Clusters

Fig. 22 presents the CDF of per-frame latency when (*i*) inferring with only one device (*i.e., Jetson*, *RPi*, *XiaoMi*); (*ii*) uniformly distributes workloads to four devices (*i.e., Uniform*); (*iii*) distributing the original frames to four devices based on device capabilities (*YOLOv7*); and (*iv*) our designed Markov approximation-based scheduling method (*i.e., Markov*). Note that *Jetson*, *RPi*, *XiaoMi*, *Uniform* and *Markov*

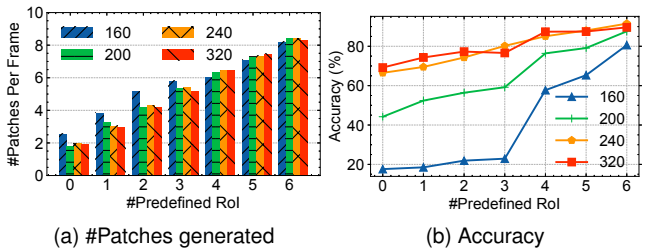

Fig. 21. #Patches generated and accuracy with predefined RoIs under various input size.

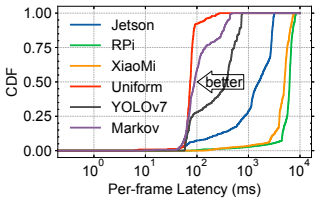

Fig. 22. Per-frame latency with different scheduling strategy.

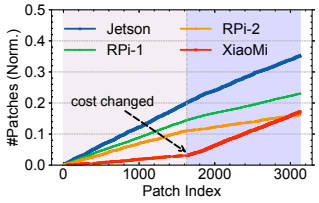

Fig. 23. #Patches dispatched to each device.

infer on single-object patches with task-oriented mini models, while *YOLOv7* infers on original frames (*i.e.,* no partition) with the YOLOv7 model. The target input size of *YOLOv7* is selected to achieve the same level of accuracy as the other methods. The number of frames dispatched to each device in *YOLOv7* is inversely proportional to the inference latency. We use *Jetson* as the client for all these six approaches to represent a typical usage case.

The results demonstrate that when using only one device for inference, *Jetson* achieves the lowest per-frame latency among the three types of edge devices. *RPi* has the highest latency since it does not have GPUs. Compared to using only one device for inference, uniformly assigning tasks to four devices for parallel inference (*i.e., Uniform*) effectively shortens the per-frame latency to 80ms on average. This shows that using a single device fails to process the frames in real time due to insufficient resources, while the cluster of them collectively has the ability to do so. When inferred on original frames with the YOLOv7 model, the average per-frame latency increases to 345ms. Although in this approach the frames are dispatched to each device according to their computational capabilities, the latency is high due to the heavy computational load of the YOLOv7 model. By adopting our designed Markov approximation-based scheduling method (*Markov*), we achieve an average latency of 141ms, which is much lower than *YOLOv7*, but slightly higher than that of *Uniform* (which we will cover shortly). This demonstrates the advantage of ParaLoupe in achieving lower latency than the typical approach (*i.e., YOLOv7*) by inferring on single-object patches using specifically designed task-oriented mini models.

To explain why the latency of *Markov* is higher than *Uniform*, we perform a more detailed examination of the *Markov* approach. For *XiaoMi*, we set the computational cost during scheduling to be high for the first 50% of frames to simulate the case where the workload of analytics task is high, and low for the latter 50% of frames. Fig. 23 presents the

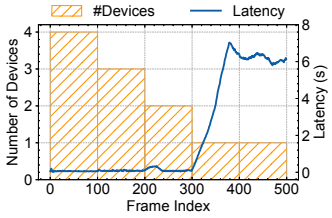

Fig. 24. End-to-end latency with different number of devices available.

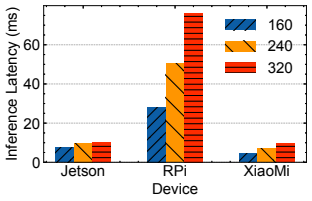

Fig. 25. Inference latency with various input size.

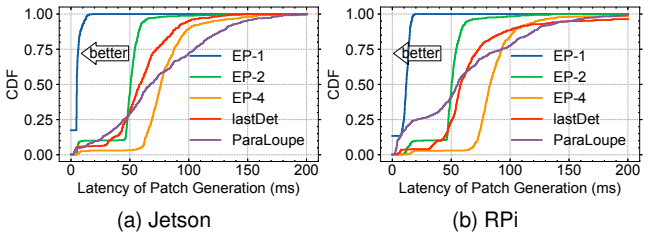

Fig. 28. CDF of patch generation latency on various hardware.

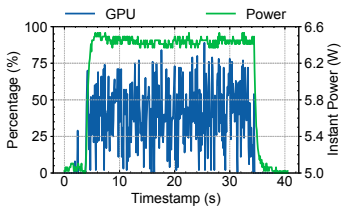

Fig. 26. GPU and power usage trace of the server side on Jetson.

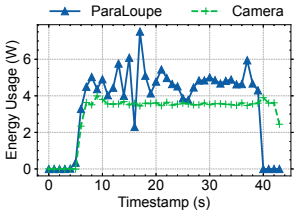

Fig. 27. Energy usage of ParaLoupe and video capture on XiaoMi.

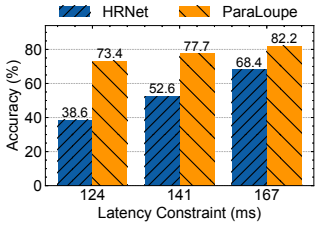

Fig. 29. Accuracy improvements under the same latency for the keypoint detection task.

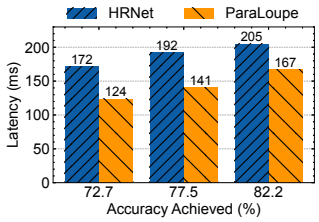

Fig. 30. Latency improvements with comparable accuracy levels for the keypoint detection task.

cumulative number of patches (normalized) sent to each of the four devices. We observe that the Markov-based scheduling assigns fewer tasks to *XiaoMi* (due to its high cost) than the other three devices in the beginning 50% of frames. This is the reason why the average latency of *Markov* is higher than *Uniform* in Fig. 22, which is also in line with our expectations. When the cost of *XiaoMi* drops at the 50th percentile frames, ParaLoupe sends more patches to it for inference to leverage the available resources. These results demonstrate the ability of ParaLoupe to achieve low per-frame latency on edge clusters, while effectively considering the actual situation of the clusters in scheduling decisions.

F. Impact of Network Variation

Although ParaLoupe is designed for edge clusters where the network transmission latency between edge devices is low, network variation remains an important factor to consider. To demonstrate ParaLoupe's robustness to network performance variations, Fig. 24 shows the end-to-end latency with different number of server devices available for inference. There are four server devices in total, which are sequentially eliminated (*i.e.,* disconnected) in the order of *Jetson*, *XiaoMi*, *RPi-1*, *RPi-2*. For example, all four devices are available for inference from frame 0 to 100, and only *RPi-2* is available from frame 300 to 500. This setup simulates scenarios where some devices are disconnected from the client device. The results show that the latency remains low (around 100ms) when at least two devices (*i.e., RPi-1* and *RPi-2*) are available. When only one device (*RPi-2*) is available, the latency gradually increases to around 6s and then stabilizes due to its low computational capacity. This demonstrates that ParaLoupe is robust to network variations, *i.e.,* it can still achieve low latency by leveraging the remaining available devices even when some devices are disconnected.

G. System Microbenchmarks

In this section, we examine the system overheads of ParaLoupe on the server device (mainly for the inference with mini models) and client device (mainly for the object-centric patch generator), respectively. To represent real-world usage scenario, we show the server-side overhead on all three kinds of devices in our experimental settings, and client-side overhead on *Jetson* and *RPi*.

1) Server-side overhead: The server-side processing latency of ParaLoupe is mainly composed of inference latency with mini models. Fig. 25 presents the inference latency of our designed task-oriented mini models with three input sizes on edge devices. The results show that the inference latency on *Jetson* and *XiaoMi* is around 10ms, as the mini models are extremely lightweight and the NCNN framework used for inference is specially optimized for embedded devices. The inference latency becomes higher as the input size increases, and *RPi* has the highest inference latency. We also evaluate the memory footprint of the mini models with an input size of 240. The memory consumed is 121.9MB, 99.2MB, and 51.3MB for *Jetson*, *RPi*, and *XiaoMi*, respectively.

Fig. 26 further presents the GPU and power usage of the server side of ParaLoupe on *Jetson*. The average GPU usage is 30.9%, and the peak power consumption is 6.5W (with power consumption at 4.6W when the server side is idle). To demonstrate the energy usage on battery-powered edge devices, Fig. 27 shows the energy usage of the server side of *ParaLoupe* as well as the energy usage of video capture using the camera on *XiaoMi* for comparison. Note that the energy usage is initially zero as it is measured relative to each process. It can be observed that the energy usage of *ParaLoupe* (with an average of 4.46W) is only slightly higher than that of video capture with the camera (3.56W). Additionally, the energy consumption of *ParaLoupe* is proportional to the number of dispatched patches for inference. Therefore, it is crucial to

control the number of patches dispatched to each device based on its computational cost (§III-D1). These results demonstrate the lightweight nature of the mini models in ParaLoupe and the superiority of deploying them on edge devices.

2) Client-side overhead: The client-side latency of ParaLoupe is mainly incurred by patch generation on the CPU. Fig. 28 presents the CDF of processing latency for five different patch generation methods on *Jetson* (Fig. 28a) and *RPi* (Fig. 28b). Compared to no partition (*i.e., EP-1*), generating two patches (*i.e., EP-2*) induces an average latency of 48.6ms (*Jetson*) and 49.8ms (*RPi*). The patch generation latency increases with the number of patches generated (*e.g., EP-4>EP-2>EP-1*). Compared to equal partition, *lastDet* induces latency of 60.1ms (*Jetson*) and 70.1ms (*RPi*) which is between *EP-2* and *EP-4*. By adding low confidence patches on the basis of *lastDet*, *ParaLoupe* achieves a latency of 74.7ms (*Jetson*) and 62.5ms (*RPi*). Moreover, the client side of ParaLoupe consumes at most 148.3MB (*Jetson*) and 144.5MB (*RPi*) of memory, and the CPU usage is around 24.7% (*Jetson*) and 39.7% (*RPi*). Similarly, *Jetson*'s peak power consumption is also $6.5W$. These results show that the client side of ParaLoupe is also lightweight and suitable for edge devices.

H. Generalization to Other Tasks

To demonstrate the ability of ParaLoupe to generalize to other tasks, Fig. 29 and Fig. 30 show the accuracy and latency improvements of *ParaLoupe* over *HRNet* for the keypoint detection task. Keypoint detection involves identifying specific points within an image. It plays a crucial role in various fields including facial analysis [60] and gesture recognition [61]. For the evaluation, we use the Human3.6M Dataset [62], [63], and measure the accuracy using the OKS (Object Keypoint Similarity) metric. The baseline approach, *HRNet*, which uses the HRNet model [64] for keypoint detection, distributes the inference tasks to all four devices based on their computational capacity. In comparison, *ParaLoupe* employs task-oriented mini models designed for keypoint detection and also distributes the frames to the four devices based on their computational capacity. To solely evaluate the performance of keypoint detection, we preprocess the frames by including only one object and disabling the patch generator module in ParaLoupe. This is because keypoint detection models typically depend on an object detection model to first detect objects and then identify keypoints on each object.

Fig. 29 presents the results. We see that, under the same latency constraints (with a difference smaller than 3ms), ParaLoupe improves the accuracy by 90.2%, 47.7%, and 20.2% for the three tested latency constraints. Additionally, as shown in Fig. 30, at a similar level of accuracy (with a difference smaller than 1%), ParaLoupe reduces the latency by 27.9%, 26.6%, and 18.5% for the three tested accuracy levels, respectively. These findings indicate that when applied to other tasks, ParaLoupe still achieves higher accuracy and lower latency compared to existing approaches, due to the task-oriented mini models.

V. Related Work

A. Real-time Video Analytics System

Real-time video analytics systems are attracting attention as deep learning techniques and computation-enabled cameras develop. A large number of works [6]–[11], [13] have been proposed to perform real-time video analytics on edge devices with low latency, low cost, high accuracy, and high reliability. For example, NeuLens [8] leverages an assemble region-aware convolution supernet for spatial-based dynamic acceleration of convolutional neural networks on the edge. Reducto [11] performs on-camera filtering with low-level video features for resource-constrained edge devices. REMIX [6] designs an image partition and model execution plan for high-resolution object detection on edge devices with tunable latency. EAAR [12] employs offloading and fast offline object tracking for mobile augmented reality. Compared to these systems, ParaLoupe shares the same design goal but focuses on inference with the edge cluster using task-oriented mini models.

B. Distributed Model Inference

Distributed model inference is important for resource-constrained edge devices to not only avoid large memory footprint but also allow inference parallelism [21], [65]. MoDNN [20] partitions the input neurons along the longer edge of the input matrix of convolution layers for distributed computation. DeepThings [21] divides images and feature maps of convolutional layers in a grid fashion for parallel execution. With a similar parallel workflow as DeepThings [21], CoEdge [24] optimally partitions the input inference workload according to devices' computing capabilities and network conditions to improve latency and energy metrics. DeepSlicing [65] models CNNs as a DAG, where each node represents a latent feature map and each edge represents a specific layer (e.g., convolution, average-pooling). It accelerates the inference process using flexible fine-grained scheduling and in-time memory reclamation. These works focus on splitting model inference on multiple devices while obtaining exactly the same inference results. In contrast, ParaLoupe sacrifices minor performance degradation for better parallelism by partitioning the input frame into multiple single-object patches.

C. Hardware-aware NAS

In recent years, neural architecture search (NAS) has gained significant attention in the field of deep learning due to its ability to automatically search for better architectures. However, a major disadvantage of NAS is the substantial computational cost it incurs. To mitigate this issue, novel techniques such as weight-sharing and single-path methods have been proposed [66]–[68]. Several works [69]–[74] have also emerged to make NAS more hardware-friendly by taking latency into consideration. For end-to-end model searching, some studies [75], [76] proposed optimizing the backbone, neck, and head parts of the network simultaneously for downstream tasks. However, they still encounter challenges such as high search cost and low accuracy. To tackle this issue, our proposed mini

models focus solely on searching for the backbone, while we manually optimize the neck and head parts to achieve a better tradeoff between latency and accuracy.

VI. Conclusion

In this paper, we have presented ParaLoupe, a novel real-time video analytics framework based on mini model parallelization. ParaLoupe intelligently generates multiple single-object patches from the original frame, designs task-oriented mini models for inference, and proposes a patch-based task scheduler to optimize the overall performance of the system. We evaluate ParaLoupe on real-world videos with various hardware platforms. The results demonstrate that ParaLoupe can achieve either 45.1% accuracy improvements for a given latency budget, or speed up the inference by up to 14.1× while obtaining similar accuracy compared to SOTA models. We believe that ParaLoupe offers a promising solution for edge devices with limited resources, and will facilitate the deployment of video-related applications in real-world scenarios.

There are several interesting directions for future research. First, we are keen to apply ParaLoupe to more vision tasks (*e.g.,* image segmentation) to enhance its adaptability and generalizability. Second, we wish to explore the potential security and privacy issues of running distributed inference across multiple devices, alongside developing appropriate mitigation approaches. Third, we wish to deploy further model-related optimizations (*e.g.,* model splitting, distillation, pruning) to improve the performance of the mini models. Finally, we would like to deploy ParaLoupe in the wild, and evaluate its efficacy.

Acknowledgments

This work is supported by the Major Key Project of PCL under grant No. PCL2023A06, the National Key Research and Development Program of China under grant No. 2022YFB3105000, the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989, National Natural Science Foundation of China (Grant No. 62072440), and Beijing Natural Science Foundation (Grant No. L221004).

References

[1] A. Balasundaram and C. Chellappan, "An intelligent video analytics model for abnormal event detection in online surveillance video," *Journal of Real-Time Image Processing*, vol. 17, no. 7, pp. 4487–4495, 2020.

[2] A. Fitwi and Y. Chen, "Privacy-preserving selective video surveillance," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, Virtual Event, United States, Aug. 2020.

[3] B. Yang, X. Cao, K. Xiong, C. Yuen, Y. L. Guan, S. Leng, L. Qian, and Z. Han, "Edge intelligence for autonomous driving in 6g wireless system: Design challenges and solutions," *IEEE Wireless Communications*, vol. 28, no. 2, pp. 40–47, 2021.

[4] A. Walker, D. Maeda, and J. Acharya, "Lightweight video analytics for cycle time detection in manufacturing," in *2021 IEEE International Conference on Big Data (Big Data)*, Orlando, United States, Dec. 2021.

[5] Q. Zhang, H. Sun, X. Wu, and H. Zhong, "Edge video analytics for public safety: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1675–1696, 2019.

[6] S. Jiang, Z. Lin, Y. Li, Y. Shu, and Y. Liu, "Flexible high-resolution object detection on edge devices with tunable latency," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, New Orleans, United States, Mar. 2021.

[7] 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*, New Orleans, United States, Oct. 2021.

[8] X. Hou, Y. Guan, and T. Han, "Neulens: spatial-based dynamic acceleration of convolutional neural networks on edge," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, Sydney, Australia, Oct. 2022.

[9] J. Yi, S. Choi, and Y. Lee, "Eagleeye: Wearable camera-based person identification in crowded urban spaces," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, London, United Kingdom, Apr. 2020.

[10] 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*, Virtual Event, United States, Jul. 2020.

[11] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, Virtual Event, United States, Jul. 2020.

[12] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, Los Cabos, Mexico, Aug. 2019.

[13] M. Yuan, L. Zhang, F. He, X. Tong, and X.-Y. Li, "Infi: End-to-end learnable input filter for resource-efficient mobile-centric inference," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, Sydney, Australia, Oct. 2022.

[14] 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.

[15] K. Huang and W. Gao, "Real-time neural network inference on extremely weak devices: agile offloading with explainable ai," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, Sydney, Australia, Oct. 2022.

[16] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Los Alamitos, United states, Jun. 2016.

[17] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Honolulu, United states, Jul. 2017.

[18] B. Lyu, H. Yuan, L. Lu, and Y. Zhang, "Resource-constrained neural architecture search on edge devices," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 134–142, 2021.

[19] X. Xu, Q. Wu, L. Qi, W. Dou, S.-B. Tsai, and M. Z. A. Bhuiyan, "Trust-aware service offloading for video surveillance in edge computing enabled internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1787–1796, 2020.

[20] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, Lausanne, Switzerland, Mar. 2017.

[21] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[22] R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Deeperthings: Fully distributed cnn inference on resource-constrained edge devices," *International Journal of Parallel Programming*, vol. 49, pp. 600–624, 2021.

[23] 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-IEEE Conference on Computer Communications*, New York, United states, May 2023.

[24] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[25] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Honolulu, United states, Jul. 2017.

[26] 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*, Columbus, USA, Jun. 2014.

[27] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision*, Santiago, Chile, Dec. 2015.

[28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, Montreal, Canada, Dec. 2015.

[29] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, United States, Jul. 2017.

[30] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, United States, Jun. 2020.

[31] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, Los Angeles, United States, Jun. 2019.

[32] F. Paissan, A. Ancilotto, and E. Farella, "Phinets: a scalable backbone for low-power ai at the edge," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 5, pp. 1–18, 2022.

[33] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Colorado, United States, Jun. 2018.

[34] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, pp. 235–256, 2007.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, United States, Jun. 2009.

[36] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *European Conference on Computer Vision*, Zurich, Switzerland, Sep. 2009.

[37] "Ncnn," Accessed Jan. 5, 2024. [Online]. Available: https://github.com/Tencent/ncnn

[38] "Logistic regression," Accessed Jan. 8, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Logistic_regression

[39] "iperf3," Accessed Jan. 9, 2024. [Online]. Available: https://github.com/esnet/iperf

[40] T. X. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 1965–1978, 2018.

[41] B. Han, J. Leblet, and G. Simon, "Hard multidimensional multiple choice knapsack problems, an empirical study," *Computers & operations research*, vol. 37, no. 1, pp. 172–181, 2010.

[42] Z. Z. T. Ouyang and X. Chen, "Follow me at the edge: Mobility aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.

[43] S. Zhang, C. Wang, Y. Jin, J. Wu, Z. Qian, and S. Lu, "Adaptive configuration selection and bandwidth allocation for edge-based video analytics," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 258–298, 2022.

[44] "Zeromq," Accessed Jan. 5, 2024. [Online]. Available: https://zeromq.org/

[45] A. Padmanabhan, N. Agarwal, A. Iyer, G. Ananthanarayanan, Y. Shu, N. Karianakis, G. H. Xu, and R. Netravali, "Gemel: Model merging for memory-efficient, real-time video analytics at the edge," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, United states, April 2023.

[46] M. A. Arefeen, S. T. Nimi, and M. Y. S. Uddin, "Framehopper: Selective processing of video frames in detection-driven real-time video analytics," in *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Los Angeles, USA, May 2022.

[47] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, Seoul, South Korea, Nov. 2015.

[48] V. S. Sindhu, "Vehicle identification from traffic video surveillance using yolov4," in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, May 2021.

[49] Z. Xiao, Z. Xia, H. Zheng, B. Y. Zhao, and J. Jiang, "Towards performance clarity of edge video analytics," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, Los Alamitos, United States, Dec. 2021.

[50] "Udacity self-driving car dataset," Accessed Jan. 5, 2024. [Online]. Available: https://github.com/udacity/self-driving-car

[51] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Vancouver, Canada, Jun. 2023.

[52] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee, and B.-G. Chun, "Band: coordinated multi-dnn inference on heterogeneous mobile processors," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, Portland, Oregon, Jun. 2022.

[53] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: Scaling live video analytics with workload-adaptive distributed edge intelligence," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, New York, United States, Nov. 2020.

[54] R. Xu, J. Lee, P. Wang, S. Bagchi, Y. Li, and S. Chaterji, "Litereconfig: Cost and content aware reconfiguration of video object detection systems for mobile gpus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, Rennes, France, Apr. 2022.

[55] C. Zhang, M. Yu, W. Wang, and F. Yan, "Enabling cost-effective, slo-aware machine learning inference serving on public cloud," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1765–1779, 2020.

[56] J. Wu, L. Wang, Q. Jin, and F. Liu, "Graft: Efficient inference serving for hybrid deep learning with slo guarantees via dnn re-alignment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 2, pp. 280 – 296, 2023.

[57] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, J. He, G. Yang, and C. Xu, "Erms: Efficient resource management for shared microservices with sla guarantees," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, Vancouver, Canada, Mar. 2023.

[58] D. Kanter, "Graphics processing requirements for enabling immersive vr," *AMD White Paper*, pp. 1–12, 2015.

[59] B. Reddy, Y.-H. Kim, S. Yun, C. Seo, and J. Jang, "Real-time driver drowsiness detection for embedded system using model compression of deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Honolulu, United States, Jul. 2017.

[60] G. S. Parra-Dominguez, R. E. Sanchez-Yanez, and C. H. Garcia-Capulin, "Facial paralysis detection on images using key point analysis," *Applied Sciences*, vol. 11, no. 5, p. 2435, 2021.

[61] S.-K. Ko, J. G. Son, and H. Jung, "Sign language recognition with recurrent neural network using human keypoint detection," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, Honolulu, Hawaii, Oct. 2018.

[62] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, 2013.

[63] C. Ionescu, F. Li, and C. Sminchisescu, "Latent structured models for human pose estimation," in *2011 International Conference on Computer Vision*, Barcelona, Spain, Nov. 2011.

[64] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang *et al.*, "Deep high-resolution representation learning for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 10, pp. 3349–3364, 2020.

[65] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "Deepslicing: collaborative and adaptive cnn inference with low latency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2175–2187, 2021.

[66] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *International Conference on Machine Learning*, Stockholm, SWEDEN, Jul. 2018.

[67] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, Ernest N. Morial Convention Center, New Orleans, May 2019.

[68] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path nas: Designing hardware-efficient convnets in less than 4 hours," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II*, Würzburg, Germany, Sep. 2019.
[69] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, C. Hao, and Y. Lin, "Hw-nas-bench: Hardware-aware neural architecture search benchmark," in *International Conference on Learning Representations*, Virtual Event, May 2021.
[70] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, Ernest N. Morial Convention Center, New Orleans, May 2019.
[71] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platforms," *ArXiv*, vol. abs/1911.00105, 2019.
[72] M. Tan, B. Chen, R. Pang, V. K. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, United States, Jul. 2019.
[73] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, United States, Jul. 2020.
[74] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, United States, Jul. 2019.
[75] J. Guo, K. Han, Y. Wang, C. Zhang, Z. Yang, H. Wu, X. Chen, and C. Xu, "Hit-detector: Hierarchical trinity architecture search for object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, United States, Jul. 2020.
[76] X. Hang, Y. Lewei, L. Zhenguo, L. Xiaodan, and Z. Wei, "Auto-fpn: Automatic network architecture adaptation for object detection beyond classification," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Los Alamitos, United States, Nov. 2019.

Hanling Wang received the B.S. degree in electronic information science and technology from Central South University, China, in 2017, and the M.S. degree in data science and information technology from Tsinghua University, China in 2020. He is currently pursuing the Ph.D. degree in computer science and technology with Tsinghua University (joint program with Peng Cheng Laboratory), China. His main research interests include video analytics, edge computing, semantic communication, and deep learning.

Qing Li (Member, IEEE) received the B.S. degree in computer science and technology from Dalian University of Technology, Dalian, China, in 2008, the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2013. He is currently an Associate Researcher with Peng Cheng Laboratory, Shenzhen, China. His research interests include network function virtualization, in-network caching/computing, intelligent self-running network, edge computing, etc.

Haidong Kang received the M.S. degree in software engineering from Northeastern University, Shenyang, China, in 2020. He is currently pursuing the Ph.D. degree in software engineering with Northeastern University, Shenyang, China. His current research interests include deep learning, computer vision, video analytics, and transfer learning.

Dieli hu received her B.Sc. degree from China Three Gorges University, China, in 2017 and her M.S. degree in the Faculty of Electronics and Communication, Guangzhou University, China, in 2021. Currently, she is pursuing a Ph.D. at the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China. Her current research interests include multimedia edge computing, video transmission, and machine learning.

Lianbo Ma (Senior Member, IEEE) received the B.S. degree in communication engineering and the M.S. degree in communication and information systems from Northeastern University, Shenyang, China, in 2004 and 2007, respectively, and the Ph.D. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2015. He is currently a Professor with Northeastern University. He has published over 90 journal articles, books, and refereed conference papers. His current research interests include computational intelligence and machine learning.

Gareth Tyson is based at the Hong Kong University of Science and Technology (GZ) and Queen Mary University of London. He holds a PhD in Computer Science from Lancaster University. His research interests focus on Internet measurements, computing networking and social computing. He regularly publishes on these topics in venues such as SIGCOMM, SIGMETRICS, IMC and WWW, and has received several awards for his work.

Zhenhui Yuan is an Assistant Professor at the University of Warwick, UK. He received his B.degree (Software Engineering) at Wuhan University, China, in 2008 and PhD (Electronic Engineering) at Dublin City University, Ireland, in 2012. His research interests lie in communications and networking of sensors, robots and vehicles. He was the postdoc researcher (2012-2014) at Ericsson and Enterprise Ireland and the visiting scholar (2014) at Network Research Lab (led by Prof. Mario Gerla), UCLA, U.S. He was 3GPP delegate (2014-2015) on 5G at Huawei (Shanghai). During 2015-2019, He was the Associate Professor in the Key Lab of RF Circuits and Systems (Ministry of Education) in Hangzhou Dianzi Univeristy, China. He was the co-founder and CTO (2015-2019) at RobSense (Hangzhou) Technology Co.Ltd , a startup that designs and produces UAV flight controller and IoT gateway. RobSense has received over £1m venture investment since 2015. He joined Northumbria University (UK) as the Senior Lecturer in 2020 and University of Warwick (UK) as an Assistant Professor in 2023. He won the best paper awards at IEEE ICCRE 2016 and IEEE BMSB 2014. He served as the Guest Editor in IEEE Network and IEEE IoT-Journal, and the Associate Editor at IEEE ACCESS, IEEE Transactions on Consumer Electronics and Journal. Pervasive Computing & Communications. He is a member of IEEE P1954 on UAV communications.

Yong Jiang (Member, IEEE) received the B.S. and Ph.D. degrees from Tsinghua University, Beijing, China, in 1998 and 2002, respectively. He is currently a full professor with Division of Information Science and Technology, Tsinghua Shenzhen International Graduate School, Shenzhen, China and Department of Mathematics and Theories, Peng Cheng Laboratory, Shenzhen, China. He mainly focuses on the future Internet architecture, Internet of Things, edge computing, AI for networks, etc.