EdgeDuet: Tiling Small Object Detection for Edge Assisted Autonomous Mobile Vision

Zheng Yang, *Fellow, IEEE*, Xu Wang, *Member, IEEE*, Jiahang Wu, *Student Member, IEEE*, Yi Zhao, *Student Member, IEEE*, Qiang Ma, *Member, IEEE*, Xin Miao, *Member, IEEE*, Li Zhang, *Member, IEEE*, and Zimu Zhou, *Member, IEEE*

***Abstract*— Accurate, real-time object detection on resource-constrained devices enables autonomous mobile vision applications such as traffic surveillance, situational awareness, and safety inspection, where it is crucial to detect both small and large objects in crowded scenes. Prior studies either perform object detection locally on-board or offload the task to the edge/cloud. Local object detection yields low accuracy on small objects since it operates on low-resolution videos to fit in mobile memory. Offloaded object detection incurs high latency due to uploading high-resolution videos to the edge/cloud. Rather than either pure local processing or offloading, we propose to detect large objects locally while offloading small object detection to the edge. The key challenge is to reduce the latency of small object detection. Accordingly, we develop EdgeDuet, the first edge-device collaborative framework for enhancing small object detection with tile-level parallelism. It optimizes the offloaded detection pipeline in tiles rather than the entire frame for high accuracy and low latency. Evaluations on drone vision datasets under LTE, WiFi 2.4GHz, WiFi 5GHz show that EdgeDuet outperforms local object detection in small object detection accuracy by 233.0%. It also improves the detection accuracy by 44.7% and latency by 34.2% over the state-of-the-art offloading schemes.**

***Index Terms*— Edge computing, object detection, real-time systems, deep learning.**

I. Introduction

BRINGING advanced machine vision to mobile devices such as drones and robots enables a wide spectrum of autonomous mobile vision applications. Examples include mobile phones for localization [1] and navigation [2], drones for cost-effective traffic surveillance [3], and robot dogs to enforce social distancing during the COVID-19 pandemic [4]. Crucial in these applications is the capability to detect objects from video inputs. An ideal object detection engine for autonomous mobile vision applications should be *accurate*, *real-time*, and *resource-efficient*. *(i)* Drones and robots should accurately detect a large number of big and small objects in the scene (*e.g.*, vehicles and pedestrians in an aerial view of a busy street). *(ii)* Fast object detection (*i.e.*, 30-60 fps) on continuous videos enables decision-making on the go. For instance, a robot may identify the crowd density from live videos and broadcast alerts when moving in a park. *(iii)* resource-efficient: For portability and mobility, the computation and memory resources in commercial drones are still limited. Object detection algorithms need to be optimized to fit in the resource budgets of mobile devices.

Existing object recognition solutions for resource-limited devices fail to satisfy the accuracy and real-time requirements. For example, in mobile AR/MR scenarios, their systems require to run at 60fps and 2K resolution for the object detection or object segmentation tasks [5]. For the huge computing resources requirements, the commercial mobile AR solutions such as ARKit [6] and ARCore [7] only track the pose of the camera and fail to track the locations of moving targets in real-time, which results in unnatural virtual effects [8]. *(i)* One promising approach for fast object detection is to run the model locally on-board. Model compression techniques can dramatically reduce the workload of deep learning models [9]. However, local object detection with compressed models is sub-optimal for autonomous mobile vision because accurate small object detection requires high-resolution input [10], which easily overwhelms mobile memory. *(ii)* An alternative is to offload object detection to the edge, which utilizes the powerful edge to run large models on high-resolution inputs for accurate detection. Nevertheless, offloading incurs a long delay since it involves the wireless transmission of high-resolution videos to the edge (*i.e.*, a 2000kbps 2k video with 10Mbps wireless network bandwidth means 200ms delay). Long end-to-end detection delay leads to large detection errors as the mobile device's view is constantly changing [11].

Pioneer studies [5], [11] avoid transmitting every frame by using cached detection results of previous frames to track objects in the current frame and only offloading key frames to update the cached results. This "detect+track" strategy supports real-time object detection in case of high bandwidth

Manuscript received 6 October 2021; revised 23 May 2022, 1 October 2022, and 15 November 2022; accepted 16 November 2022; approved by IEEE/ACM Transactions on Networking Editor R. Lo Cigno. Date of publication 2 December 2022; date of current version 18 August 2023. This work was supported in part by the NSFC under Grant 61832010, Grant 61632008, Grant 61872081, and Grant 61972131; and in part by the Shuimu Tsinghua Scholar Program. The work of Zimu Zhou was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 Grant. *(Corresponding author: Zheng Yang.)*

Zheng Yang, Xu Wang, Jiahang Wu, Yi Zhao, Qiang Ma, and Xin Miao are with the Tsinghua National Laboratory (TNList), School of Software, Tsinghua University, Beijing 100084, China (e-mail: yangzheng@tsinghua.edu.cn; wangxu2020@tsinghua.edu.cn; wujx20@mails.thu.edu.cn; zhaoyi.yuan31@gmail.com; maq@greenorbs.com; xin@greenorbs.com).

Li Zhang is with the School of Mathematics, Hefei University of Technology, Hefei, Anhui 230009, China (e-mail: lizhang@hfut.edu.cn).

Zimu Zhou is with the School of Information Systems, Singapore Management University, Singapore 178902 (e-mail: zimuzhou@smu.edu.sg).

Digital Object Identifier 10.1109/TNET.2022.3223412

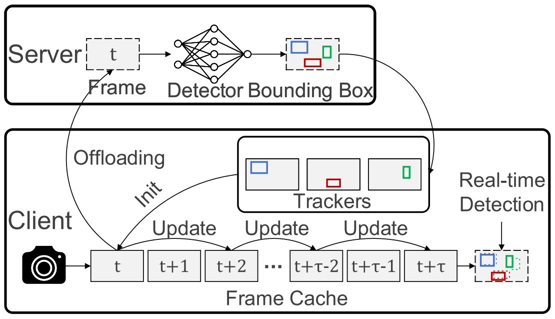


Fig. 1. An illustration of the popular "detect+track" framework for offloaded object detection. The detection results of the current frame are obtained by applying trackers on the cached detection results. The cached results are routinely updated by offloading key frames.

networks. Its performance tends to deteriorate in the case of low-bandwidth, *e.g.*, outdoors, which autonomous mobile vision applications often target at.

Instead of pure local processing or offloading, we propose to split the object detection task between the mobile device and the edge. Specifically, we offload *small object detection* to the edge. The rationale is intuitive. Commercial mobile devices are now able to accurately and rapidly detect large- to medium-sized objects by running compressed models on low-resolution videos [12], [13]. Hence only data relevant to small objects need to be uploaded to the edge in high quality, thus reducing the overall offloading delay and improving detection accuracy. The "small" size is empirically tuned to achieve optimal performance, and we will detail the classification method later.

Realizing the above idea for accurate and real-time object detection needs a systematic design on *(i) what and how to offload to the edge* and *(ii) how to aggregate the detection results*. We base our design upon "detect+track" (Fig. 1), the prevailing framework, to accelerate offloaded object detection [5], [11]. The detection results of the current frame are obtained by adapting cached detection results of prior frames using lightweight trackers [14]. The cached results are routinely updated by offloading key frames for expensive yet highly accurate object detection. In our case, the trackers and the detectors for big objects are lightweight. Hence the bottleneck for real-time detection is the offloaded small object detection. Since the detection results of the current frame rely on the cached results, the bottleneck for accurate detection, especially for small objects, lies in the freshness of the cached results.

We propose EdgeDuet, an accurate, real-time object detection engine, which tiles and offloads small object detection to the edge (Fig. 2). EdgeDuet tackles the aforementioned accuracy and real-time bottlenecks via the following techniques. *(i)* Optimizing offloaded small object detection with *region-of-interest (RoI) frame encoding* and *content-prioritized tile offloading*. EdgeDuet applies RoI frame encoding to save network traffic. Only pixel blocks potentially containing small objects are transmitted in high quality, while the rest of the frame is compressed to low quality. EdgeDuet adopts content-prioritized tile offloading to accelerate small object detection at the edge. It processes videos in the unit of tiles rather than the entire frame, so as to improve the parallelism of offloading. It also prioritizes the offloading of tiles containing more objects, so that the cached detection results of more objects are freshly updated. *(ii)* Real-time tracking via *cache management* and *adaptive tracker configuration*. EdgeDuet aggregates the detection results from the local and remote object detectors to obtain fresh and consistent cached results via a cache management mechanism. It also applies adaptive tracker configuration to improve the resource efficiency and real-time performance of the trackers.

We implement EdgeDuet as a *cross-platform* framework and evaluate its performance with mobile phones on VisDrone [15], a public video dataset captured by drone-mounted cameras. Evaluations show that pure local object detection yields a detection accuracy in terms of only 0.096 for small objects, while EdgeDuet achieves an accuracy of 0.319 for small objects.

The main contributions of this work are summarized below.

- EdgeDuet is the first framework that enhances small object detection in crowded scenes via collaboration between the edge and the mobile device.
- We push the state-of-the-art offloaded object detection studies [5], [11] from task-level parallelism to tile-level parallelism, which notably reduces the offloading latency. EdgeDuet is a systematic design that enables accurate, real-time object detection on mobile devices even in the case of low network bandwidth.
- We implement EdgeDuet as a cross-platform framework. Evaluations on VisDrone [15] show that EdgeDuet improves the overall accuracy by 44.7% and the end-to-end latency by 34.2% over the state-of-the-art object detection offloading schemes [5], [11].

In the rest of this paper, we give an overview of EdgeDuet in Sec. III and elaborate on its functional modules in Sec. IV, Sec. V and Sec. VI. We present the implementation of EdgeDuet in Sec. VII and the evaluations in Sec. VIII. We review related work in Sec. II and finally conclude in Sec. X.

II. Related Work

Our work is relevant to the following categories of research.

Object Detection Models. Advances in deep learning have resulted in various accurate and fast object detection models such as two-stage models *e.g.*, Faster-RCNN [16] and one-stage models *e.g.*, YOLO [17]. Model compression and acceleration techniques [9], [18], [19], [20], [21], [22] can substantially reduce the computation workload of deep learning-based models. However, the compressed models suffer from low accuracy on small object detection if the input image/video is low in resolution [10]. For accurate and fast small object detection, customized models [10], [23], [24] have been developed to detect objects on sub-regions of the input image/video. However, these models are computed heavily. Our work also performs object detection on sub-regions. However, rather than design new object detection models, we exploit existing YOLO-family models of different

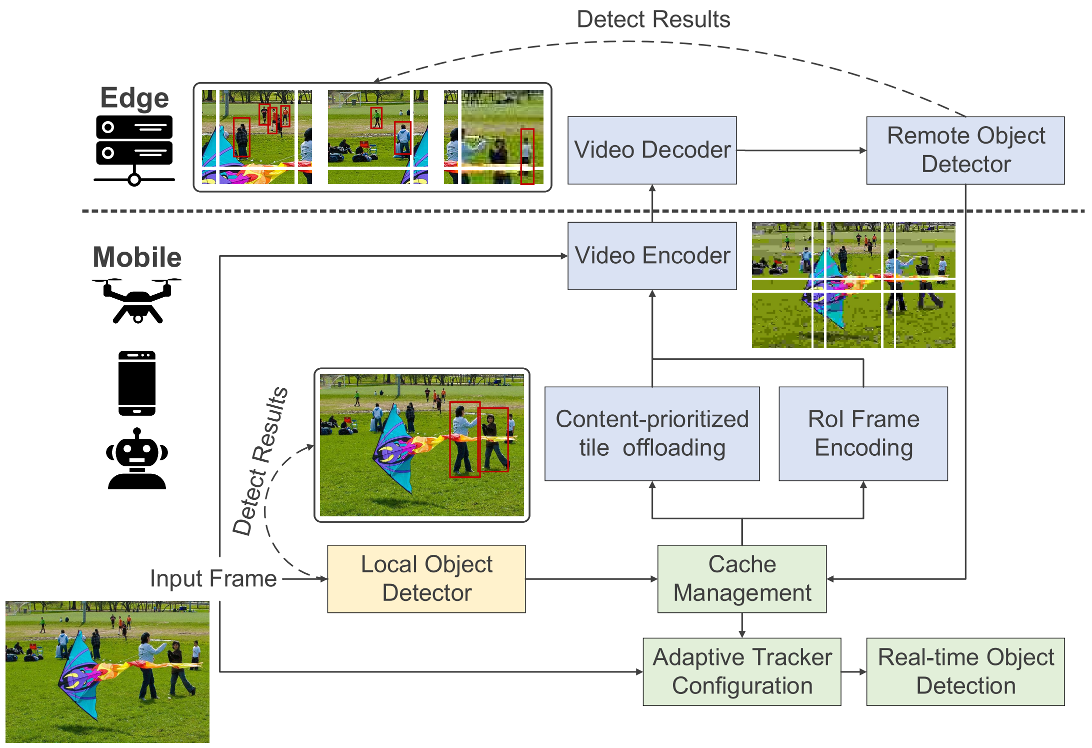


Fig. 2. An overview of EdgeDuet. Rectangles in different colors represent the three functional modules (offloaded small object detection (blue) (Sec. IV), local object detector (yellow) (Sec. V) and real-time tracking (green) (Sec. VI). EdgeDuet is implemented as a cross-platform framework consisting of both edge-side and device-side modules (Sec. VII)).

capabilities [17] to process different sub-regions of video frames.

Edge/Cloud Offloading. A popular strategy to enable highly accurate object detection on resource-constrained mobile devices is to offload the compute-intensive object detection to the powerful edge/cloud server [5], [11], [25], [26], [27], [28], [29], [30], [31], [32], [33]. However, offloading may incur long delays since large amounts of videos need to be uploaded to the server via wireless networks. To enable offloaded object detection on continuous videos, Glimpse [11] proposes to only send trigger frames and proposes the “detect + track” framework for fast object detection. EAAR [5] compresses the uploaded frames via RoI based video encoding and applies parallel streaming and inference techniques to reduce the offloading latency further. Our work is built upon the “detect + track” framework and the pipelined offloading principle, but improves the parallelism of the offloading pipeline to tile-level. Furthermore, these studies do not optimize small object detection. DDS [29] differentiates small and large object detection by first offloading high-resolution, low-quality video frames to detect large objects and locate small objects. Regions containing small objects are then encoded in high quality and offloaded again to detect small objects. The method improves the accuracy of small object detection but doubles the delay for object detection. Unlike DDS, which detects large and small objects sequentially, we run a fast model on low-resolution frames to detect large objects and offload small object detection with high-quality frames at the same time.

Tile-based Video Streaming. Tiling feature [34] in video codecs has provided better quality gains for video streaming [35], [36], [37], [38]. In tile-based video streaming, the video is first cut into tiles and important tiles are transmitted in high quality, whereas others are transmitted in low quality or not transmitted at all. Since all of them are based on existing video codecs, they encode the whole frame into the bit-streams then process each tile’s bit-stream independently. Differently, we redesign the video encoder such that it outputs the bit-stream of each tile once it is encoded and transmits the bit-stream to the edge server immediately to reduce the offloading latency.

Single Object Tracking. There are many existing accurate tracking algorithms [39], such as optical-flow based Lucas-Kanade tracking [40], correlation-filter based KCF [14], deep learning based Siamese RPN [41]. Most of these techniques require lots of computational resources, and are not efficient for applications on mobile devices with real-time requirements. We choose KCF as our tracking algorithm for its excellent efficiency and accuracy, and limited resource requirements.

III. EdgeDuet Overview

As shown in Fig. 2, EdgeDuet consists of three functional modules:

i) An *offloaded small object detection* module which uploads high-resolution frames to the edge to detect small objects. EdgeDuet optimizes offloaded small object detection with region-of-interest (RoI) frame encoding and content-prioritized tile offloading to save network traffic and accelerate small object detection at the edge. EdgeDuet processes videos in the unit of tiles rather than the entire frame, so as to improve the parallelism of offloading.

ii) A *local object detector* module which detects large objects from low-resolution frames. The local object detector aims to detect medium- to large- sized objects in the video frames locally on the mobile device. We empirically decide the model and input resolution for the local object detector to meet the constraints of resources on the mobile device.

iii) A *real-time tracking* module which associates the detection results (bounding boxes, a.k.a `bboxes`) from both the edge and the mobile device and tracks each object with single-object trackers. EdgeDuet adopts multiple single-object trackers for object tracking and it also applies adaptive tracker configuration to improve the resource efficiency and real-time performance of the trackers.

We elaborate on the detailed designs of each functional module in the subsequent sections.

IV. Offloaded Small Object Detection

This module aims to *(i)* reduce the data for transmission to the edge and *(ii)* accelerate the offloading pipeline for timely updates of the cached detection results on the mobile device. EdgeDuet exploits RoI frame encoding to compress video frames, and content-prioritized tile offloading for highly parallel object detection at the edge.

A. RoI Frame Encoding

As mentioned in Sec. I, accurate small object detection relies on high-resolution, high-quality frames as input. Yet uploading high-quality frames to the edge impairs real-time object detection [23], [24], [26]. The RoI frame encoding module reduces the amount of transmitted data by only keeping the pixel blocks containing small objects in high quality while compressing the rest of the frame to low quality. Although RoI frame encoding has been used in other offloading schemes [5], [29], the definition of RoI (*i.e.*, blocks containing small objects in our case) and the compression level varies and should be tuned for specific applications.

1) Determining Blocks Containing Small Objects: A pixel block is considered as containing small objects if *(i)* the local object detector cannot classify the block into a class (or reports low confidence scores); and *(ii)* the remote object detector can classify the block to a class (or reports high confidence scores). Due to the high temporal correlation between successive frames, we use the detection results of the previous frame to identify blocks potentially containing small objects in the current frame. For simplicity, we decide whether an object is small using a fixed size. The size is empirically tuned such that objects below this size *cannot* be accurately detected by the *local* object detector but *can* be accurately detected by the *remote* object detector. The object size is considered as accurately detected if the recall is above 90%. Experiments show that the optimal size threshold for small objects varies across classes. For example, a size of 2000 pixels results in almost 100% recall for pedestrians but less than 40% recall for cars (see Fig. 3). Hence a different size threshold is set for each targeting class.

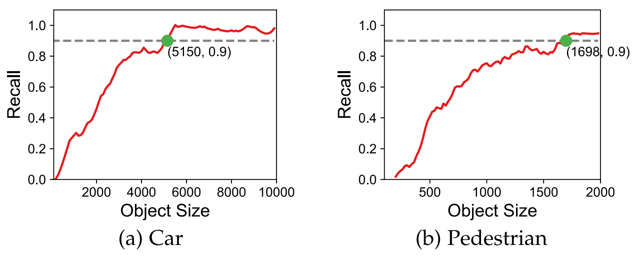


Fig. 3. An example of the class-dependent size threshold for small objects. Details of datasets, local and remote object detectors see Sec. VII.

2) Determining Compression Levels: Blocks which are determined as containing no small objects cannot be compressed to arbitrarily low quality. This is because the decision is made based on the detection results of the *previous* frame. If a new object appears in the *current* frame, the blocks containing this object may be so heavily compressed that the object cannot be detected by the remote object detector. To avoid missing detection of new objects, we tune a low-quality compression level. Fig. 4 shows the rationality. For low-quality images, the object detection model could approximately detect most objects' locations by lowering its confidence score. We choose the compression level such that the remote object detector outputs low confidence scores on the compressed blocks but will not fail to locate objects. These low confidence objects are also returned to the device for offloading their blocks at the next frame, so as to be accurately detected in the next frame. To avoid a cold start, we just offload the whole frame in high quality.

3) Implementing RoI Frame Encoding: We use the High Efficiency Video Coding (HEVC, a.k.a h.265) codec [42] to encode pixel blocks containing small objects to high quality and compress the rest of the frame to low quality. The Quantization Parameter (QP) is one critical arguments in video compression performance. Larger values mean that there will be higher quantization, more compression, and lower quality. Lower values mean the opposite. In our scheme, we generate a delta QP map describing the delta QP values of each macroblock in the raster order and encode the current frame with the HEVC codec. Fig. 5a and Fig. 5b show an example image before and after RoI frame encoding.

B. Content-Prioritized Tile Offloading

This module enables real-time small object detection via fine-grained (tile-level) parallel offloading. It also facilitates timely updates of cached detection results on the mobile device by prioritizing the processing of tiles that contain more small objects. Pipelined offloading proves effective for fast object detection [5], where the offloading process is split into frame encoding, frame upload, frame decoding, object detection, and result downloading. Nevertheless, existing work [5] pipelines the offloading process on a frame basis, which limits the achievable parallelism. In contrast, EdgeDuet breaks a frame into tiles and enables tile-level parallelism, thus allowing faster pipelined and parallel offloading. We explain how to realize tile-level parallelism and content-based priority below.

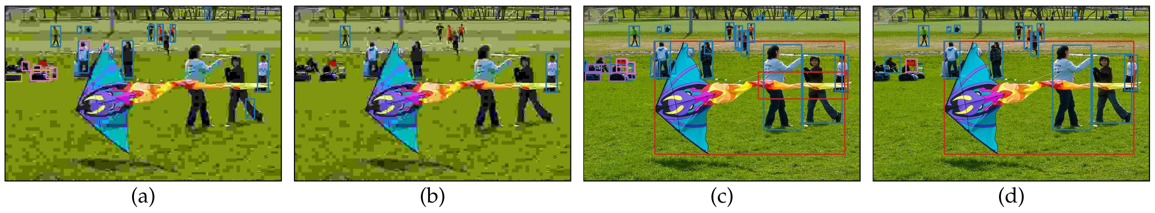


Fig. 4. Detected objects for different image quality. (a) Low quality, low confidence objects. (b) Low quality, high confidence objects. (c) High quality, low confidence objects. (d) High quality, high confidence objects.

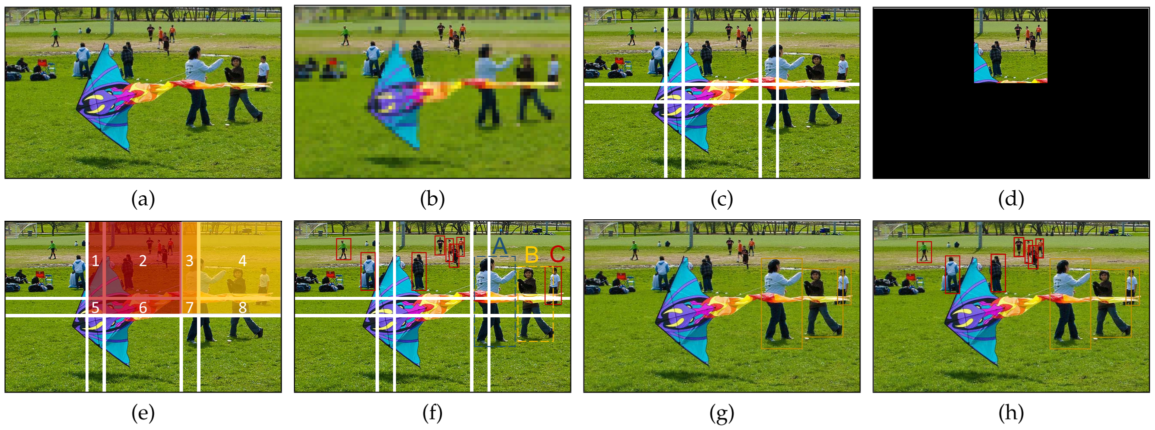


Fig. 5. An example of key steps in EdgeDuet. (a) Input frame. (b) Frame after RoI frame encoding, where blocks containing no small objects are compressed to low quality. (c) Tiles. (d) The output of video decoder after enabling tile-level parallelism. (e) Overlap-tiling. (f) Remote object detector results of tiles (red rectangles). (g) Local object detector results of the low-resolution frame (yellow rectangles). (h) Cache management of remote and local object detectors.

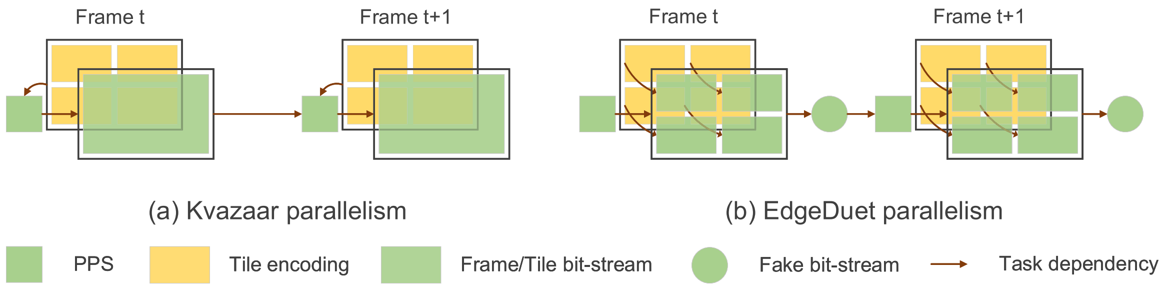


Fig. 6. Video encoding parallelism of Kvazaar and EdgeDuet.

1) Enabling Tile-Level Parallelism: A tile is a rectangular region in a frame defined in HEVC [34]. Fig. 5c shows an example of 5×3 tiles. To support tile-level parallelism, we need to modify the frame encoding, frame decoding, and object detection stage, as they are designed to operate on a frame basis. The principle is to eliminate dependencies among tiles for each stage, as described in detail below.

- *Frame Encoding*. Existing video encoders [43], [44], [45] output the encoded bit-stream after processing *all the tiles* in a frame. We redesign the video encoder such that it outputs the bit-stream of *each tile* once it is encoded. Our method is based on Kvazaar [45], which treats the encoding of each tile as an individual task and allows parallel tile encoding via a dynamic task graph. However, Kvazaar outputs bit-streams on a frame basis. Fig. 6(a) shows the task dependencies among tile encoding tasks and frame bit-stream tasks of the current frame and the next frame in Kvazaar. We modify its bit-stream writing module so that the bit-stream tasks operate on a tile basis, as the task dependencies shown in Fig. 6(b). Specifically, we break the bit-stream of a frame into a picture parameter set (PPS) and each tile's bit-streams. PPS contains the meta information of each frame on entropy coding mode, slice groups, motion prediction, quantization parameters (QP), and deblocking filters. Consequently, each tile's bit-stream only depends on PPS and the tile encoding task. Hence the video encoder will first output the PPS, and once one tile is encoded, its bit-stream will be output and sent for offloading. We also introduce a fake bit-stream task to mark the end of the bit-stream tasks in a given frame.
- *Frame Decoding*. Existing video decoders operate on a frame basis. They assume the bit-streams of all the tiles in a frame arrive sequentially and utilize the offset from the first tile in the frame to locate the other tiles. For example, in HEVC, only the location of the first

tile is signaled in the slice header. All the other tiles transmit their bit-stream offsets in the slice header, which introduces dependencies on the first tile. We eliminate such dependencies and enable tile-level parallelism in frame decoding by forcing every tile in a frame as a "first tile". This is implemented by modifying the bit-stream of each tile in the video encoder (Kvazaar) and the HEVC parser in the video decoder (OpenHevc [46]) accordingly. Fig. 5d shows an example of tile-level frame decoding. Each tile is decoded to its position independent of the other tiles (shown in black).

- *Object Detection*. Performing object detection on each tile separately may miss objects which cross the boundaries of adjacent tiles. We mitigate such dependencies among tiles during object detection via *overlap-tiling*. That is, we split the frame into $M \times N$ tiles, where M and N are odd numbers. We use (i, j) to denote each tile, where $i \in [1, M]$ and $j \in [1, N]$ are the row index and column index, respectively. We classify tiles into two categories. If both i and j are odd numbers, the tile (i, j) is a primary tile. Otherwise, the tile is an overlay tile. Fig. 5e shows an example. Since tile 2 can be denoted as $(1, 3)$, tile 2 is a primary tile. We can conclude that tile 2 and 4 are primary tiles and tile 1, 3, 5, 6, 7, 8 are overlap tiles. We group each primary tile with its surrounding overlap tiles for small object detection. In this example, tile 1, 2, 3, 5, 6, 7 will be grouped. Detecting objects for each tile group reduces the probability of missing objects that exceed the boundary of a primary tile. We only group the surrounding tiles because our remote detector targets at small objects. Only large objects may be present in two primary tiles crossing the overlap tiles, as person A and B in Fig. 5f. The overlap size $S_{overlap}$ (minimal width and height of overlap tiles) is set to the least multiple large coding unit (LCU), which is larger than the maximal size of small objects defined in Sec. IV-A.1 and complies with the tile definition in HEVC. The formula is illustrated as

$$S_{overlap} = \lceil \frac{\max_c\{\max\{w_c, h_c\}\}}{LCU\_WIDTH} \rceil \times LCU\_WIDTH \tag{1}$$

where w_c and h_c are the width and height of the small object threshold for the targeting class c. LCU_WIDTH is the width of LCU. As object size can be arbitrary, the approach of overlap-tiling may not always work well in some extreme cases, we overcome it by simply running object detection on the whole image after the last tile is transmitted. Fig. 5f shows an example of detection results using our method.

2) *Enabling Content-Based Priority:* Prioritizing tiles containing more objects over those containing fewer objects allows the cached detection results of more objects to stay fresh. Since our implementation of tile-level parallelism (Sec. IV-B.1) ensures tiles are offloaded early to return detection results early, we only need to prioritize tiles at the frame encoding stage.

- *Implementing Tile Priority for Frame Encoding*. We modify the task schedule module in Kvazaar by adding a dynamic priority mapping module to enable the ordering of tiles (see Fig. 7). Specifically, the dynamic priority mapping layer associates a priority value p to each primary tile according to the input content, where $p \in [0, N)$ and N is the number of primary tiles, and each overlap tile calculates its priority p as the maximum priority of its surrounding primary tiles. Then the encoding task of each tile is assigned a priority value of p while its bit-stream task is assigned a priority value of $p + N$. This is to force the bit-stream task to execute once the tile is encoded, which can be before other tiles' encoding tasks.
- *Assigning Tile Priority based on Content*. To determine the priorities (*i.e.*, p) of the N primary tiles, we count the number of small objects of the corresponding tile group. The priority value p of each primary tile is the index in ascending order.

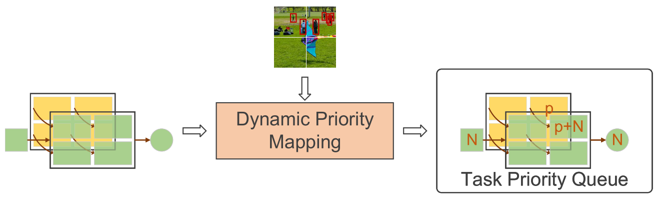


Fig. 7. An illustration of content-based tile priority.

V. Local Object Detector

The local object detector aims to detect medium- to large-sized objects in the video frames locally on the mobile device. Since mobile devices have limited resources compared with the edge, the local object detector should be lightweight and operate on low-resolution frames. We empirically decide the model and input resolution for the local object detector. The local object detector should balance between *offline accuracy* and *latency* to achieve high *online accuracy*. The offline accuracy refers to the accuracy of the object detector, while the online accuracy refers to the accuracy in the "detect+track" framework [5], [11]. Accuracy is measured by metrics such as IoU, as will be defined in Sec. VIII-A.5.

Table I shows the performance of different combinations of object detectors and input resolutions evaluated on the VisDrone dataset with an iPhone 11. For resource efficiency, the models are quantized to `float16`. Since the "real-time" detector is not a must with the "detect+track" framework, we choose YOLOv3 (640 $\times$ 640) as the local object detector based on the analysis. Fig. 5g shows an example of detection results of the local object detector. Note that we only aim to show the feasibility of running an object detector locally for accurate and real-time medium- to large-sized objects. An exhaustive search on the optimal local object detector is out of the scope of this paper.

VI. Real-Time Tracking

This module aggregates the offloaded and the local detection results into the cache and adjusts the cached results via object trackers to output the bounding boxes for the current frame.

TABLE I
PERFORMANCE OF LOCAL DETECTOR ON IPHONE 11

Model	IoU (offline)	Latency	IoU (online)
YOLOv3-tiny (320x320)	0.015	12.4ms	0.012
YOLOv3-tiny (640x640)	0.078	19.5ms	0.052
YOLOv3-tiny (960x960)	0.140	38.9ms	0.090
YOLOv3 (320x320)	0.176	23.8ms	0.092
YOLOv3 (640x640)	0.361	62.5ms	0.193
YOLOv3 (960x960)	0.522	178.7ms	0.161

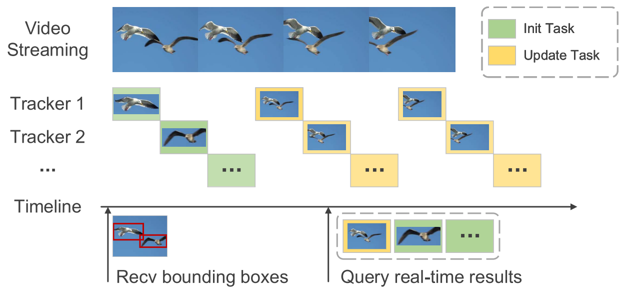


Fig. 8. General workflow using multiple single-object trackers.

EdgeDuet adopts multiple *single-object trackers* for object tracking, as in [5] and [11]. Fig. 8 shows the general workflow. Trackers return latest updated bounding boxes so as to query as the same fps as the video input. Since we target at video streams with high frame rates (30/60/120 fps) and the cached results come from two object detectors, the general workflow needs to be optimized for EdgeDuet, as we describe below.

1) Cache Management: We cache the detection results received from the local or remote object detector and discard the old results upon receiving new ones. One issue in our cache management is that the local and the remote detector may introduce duplicated detection results of the same object. We drop the results of the local detector for small objects and those of the remote detector for medium- to large-sized objects in case of duplicated results. Fig. 5h shows an example of merging local detection results and remote detection results.

2) Adaptive Tracker Configuration: To optimize the tracking performance on mobile devices, we consider the following.

- *Choice of Single-object Tracker*. We empirically choose KCF [14] as our single-object tracker since it is both faster and more accurate than the optical flow based tracker in [11] and has higher accuracy than the motion vector based tracker in [5].
- *Priority-based Tracker Scheduling*. To execute multiple single-object trackers on resource-constrained devices, we adaptively update the tracking results based on the speed of the objects, because it is unnecessary to frequently update the tracking results of objects that are static or moving slowly. Specifically, we estimate the object's speed by the object's move distance in continuously tracked frames. Then we set a different weight for each speed range, and the priority of each tracker is updated to the product of its weight value and the default priority (distance between the current frame and last tracked frame in sequential task scheduling). We schedule the tracker with high priority to track first to ensure high-speed objects frequently updated. To accelerate the tracking module, we use a thread pool to execute multiple trackers parallelly.

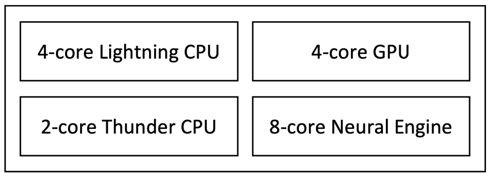


Fig. 9. Internal architecture of apple A13 SoCs.

VII. IMPLEMENTATION

This section presents the implementation of EdgeDuet on the device-side and the edge-side.

A. Implementation of Core Device-Side Modules

We implement the device-side of EdgeDuet on a popular mobile device, iPhone 11, which equips the A13 Bionic chip. As shown in Fig. 9, this SoC contains 4 processors: two high-performance Lightning CPU cores running at 2.65 GHz, four energy-efficient Thunder cores running at 1.80 GHz, four GPU cores with 0.69 TFLOPS for FP32 floating point computing and eight Neural Engine cores with 5.5 TOPS for AI computing [47]. Benefiting from the excellent performance, the Soc makes it efficient for the tasks of video compression, object detection and real-time object tracking.

The device-side modules of EdgeDuet consist of a video streamer, a video encoder, a local object detector and an object tracker. Each module has a unique thread and communicates with other modules by blocking queues. The video streamer simulates the video camera streaming process and feeds raw frames to the video encoder, the local object detector and the object tracker with different frame rates. The video encoder encodes raw frames to bit-stream and sends network packets to the server. The local object detector runs a light-weight object detection model and updates the frame cache for object tracking. The object tracker tracks all objects detected from both the local object detector and the remote object detector. Most modules on the device side are implemented in C++ 17 [48] for easy deployment on different platforms such as iOS Frameworks [49], Android NDK [50] and Nvidia Jetson [51]. We next explain more detail for each module.

1) Video Streamer: This module is used for simulating the video camera streaming process using standard video datasets. The streamer loads a video file and feeds video frames into EdgeDuet at 30/60/120 fps. The module is implemented in C++ using the `VideoCapture` module in OpenCV [52] to read RGB images from a video file and convert the image to I420 format for video encoding.

2) Video Encoder: This module is implemented in C++ based on `Kvazaar` [53], an open-source HEVC encoder. We modify the library to support tile-based parallel encoding and priority, as described in Sec. IV-B.1 and Sec. IV-B.2. The modified library is open-sourced at *https://github.com/xu-wang11/kvazaar*. We empirically encode and offload frames at a fixed frame rate (*e.g.*, 5 fps).

3) Local Object Detector: This module is implemented in Objective-C [54] with `Core ML` [55], which optimizes on-device performance by jointly leveraging the CPU, GPU, and Neural Engine. We use the pre-trained compressed YOLOv3 model YOLOv3FP16 (640×640). for medium- to large-sized object detection, as explained in Sec. V. We empirically run the local object detector at a fixed frame rate (*e.g.*, 10).

4) Object Tracker: This module is implemented in C++ with the `KCF` [14] Tracker and `ThreadPool` [56] to schedule multiple object tracking. We use the implementation of `KCFcpp` [57] without the HOG features [58] for fast object tracking, as described in Sec. VI-.2.

B. Implementation of Core Edge-Side Modules

We implement the edge-side modules of EdgeDuet on a CentOS 7.0 server. It is equipped with two 8-core Intel Xeon CPU E5-2560 v4 CPUs, two GTX 2080ti GPUs and 256GB memory. The edge-side modules of EdgeDuet consist of a video decoder and a remote object detector. The video decoder receives packets from the device-side and converts them to image tiles. The image tiles will be queued in the remote object detector for object detection.

1) Video Decoder: The video decoder is implemented in C++ based on the `OpenHEVC` library [46]. We modify the library to support tile-based parallel decoding as in Sec. IV-B.1. We use OpenCV-Python bindings [59] for Python to run the decoder and access the decoding results from memory.

2) Remote Object Detector: The detector for small objects on the edge is implemented as a pre-trained full-precision YOLOv3-spp [60] model in PyTorch [61]. We run the model in multiple processes for parallel inference on tiles. The GPU is set to CUDA Multi-Process Service mode [62] to reduce GPU context switching.

C. Implementation of Auxiliary Modules

For the edge- and device-side modules to work in synergy, we also implement the necessary functions for network communication, inter-thread communication, and data logging.

1) Network Communication: The communication between the edge and the mobile device is via TCP [63]. We use the `sockpp` [64] library for network programming. The up-link traffic is transmitted as bit-streams, and the down-link traffic (`bboxes` from the edge) is transmitted in the format of JSON. The device will parse it with the JSON [65] library.

2) Inter-Thread Communication: In our implementation, each module works as an independent thread. We use the `concurrentqueue` [66] library to block the thread when no task is put into the queue.

3) Data Logging: We use `spdlog` [67] to log the timestamps of each processing step for tracing the entire workflow.

Similarly, we use "queue" for message communication, "socket" for network communication, "logging" for logging the work flow. All of them are in the python standard library.

VIII. Evaluation

This section presents the evaluations of EdgeDuet.

A. Experiment Setup

1) Datasets: To evaluate the performance for small object detection, we compare different methods on VisDrone, a dataset of videos captured by drone-mounted cameras, which contains lots of small objects. We filter out the low-resolution videos and only keep six 2K videos (2560×1440) captured along a street. The count of frames is 1886. We upsample the origin 30fps videos to 60/120 fps with Super-SloMo [14] to evaluate the performance with the video frame rate.

Table II gives the basic statistics of the dataset. Fig. 10 shows certain vital statistics of the VisDrone dataset. From Fig. 10(a), the average number of objects in each frame is 132. As we will show later, adaptive tracker configuration is beneficial to track such many objects in real-time. We analyze the threshold of small objects by running the local object detector and remote object detector on the dataset. We use the detection results of the remote object detector as the ground truth and the areas of bounding boxes as the sizes of objects to calculate the recall-size curves of each class. Fig. 3 shows the results of the car class and pedestrian class. We set the threshold of the small car as 5150 pixels and the threshold of the small person as 1698 pixels. From Fig. 10(b) and Fig. 10(c), 75.1% of cars and 71.8% of pedestrians are small objects. We will show the performance gain of offloading small object detection over local detection shortly.

2) Compared Methods: We compare our EdgeDuet with the following object detection schemes.

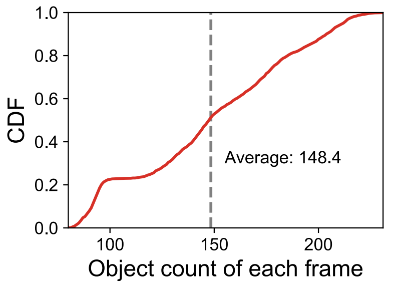
- `Glimpse [11]`: a continuous, real-time object detection system that first proposes the "detect + track" framework on mobile devices. It offloads frames to the cloud and uses optical flow based tracker for real-time tracking.
- `EAAR [5]`: a state-of-the-art real-time object detection system with offloading. It exploits parallel streaming and inference as well as motion vector based object tracking.
- `LaT`: a variant of EdgeDuet (**L**ocal object detector + **a**daptive **T**racking configuration) that only performs local object detection and tracks with our adaptive tracker configuration.

To demonstrate the effectiveness of RoI frame encoding module and Content-prioritized tile offloading module in our EdgeDuet, we also compare the performance of the following variants of EdgeDuet.

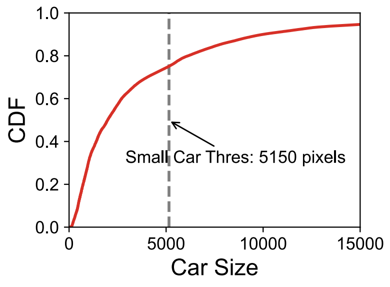
- `EdgeT` The variant only utilizes tile-level parallelism for small object detection.
- `EdgeTR` The variant is EdgeT enhanced with RoI frame encoding.

3) Implementation and Settings of Compared Methods: The implementation of EdgeDuet and `LaT` can be found in Sec. VII. We briefly explain the implementation of `Glimpse`, `EAAR`, and parameter settings for all methods below.

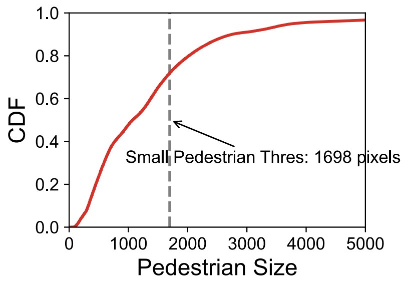
- *Frames Encoding*. To ignore the difference of JPEG encoders and video encoders, we use `Kvazaar` to get compressed JPEGs for `Glimpse` and video frames for `EAAR` and EdgeDuet. This setting ensures the same frame quality for a fair comparison. For `Glimpse`, we encode



(a) CDF of object count in each frame



(b) CDF of car size



(c) CDF of pedestrian size

Fig. 10. Characteristics of VisDrone dataset. There are numbers of objects in each frame, and most objects are small in size.

TABLE II
DATASET STATISTICS

scenario	#videos	#avg duration	#frames	#categories	#avg objects of each frame	resolution	year
drone	6	10.5s	1886	2	132	3840x2160	2018

each frame to I frame by setting Group of pictures (GOP) as 1. For EAAR and EdgeDuet, the GOP is set to 10. We use the `ultrafast` preset for real-time video encoding. We tune high-quality QP as 22 and low-quality QP as 44 for RoI encoding in EAAR and EdgeDuet. EAAR uses `Kvazaar` to encode one frame to 1×4 tiles and pack each tile into one slice, as the default setting. EdgeDuet splits the frame into 5×3 tiles for offloading, as in Fig. 5c.

- *Remote Object Detector*. We use the same remote object detection model, *i.e.*, YOLOv3-spp, for `Glimpse`, EAAR, and EdgeDuet. `Glimpse` operates on the input size of 2560×2560. EAAR operates on 1280×1280 but returns the detection results in 2560×2560. Specifically, we first cache the object detection results of each 2560×2560 frame in the memory and implement the *Dependency Aware Inference* by detecting on only one tile and overlooking other tiles for convenience. Since we return the detection results of the whole frame, our setting has the same accuracy but allows a lower latency of EAAR compared with its origin dependency aware inference. EdgeDuet operates on 960×960 for overlap-tiling inference. The overlap size is set to 128 (2 macroblocks) as in Sec. IV-B.1.
- *Real-time Object Tracking*. We implement the optical flow based tracker with `calcOpticalFlowPyrLK` for `Glimpse`. For EAAR, the motion vector based tracker is implemented as an offline process. When received from the server, each track frame is associated with a refer frame ID. We use `ffmpeg` to compress the refer and track frames and extract the motion vector based on the refer frame to simulate RPS Control in EAAR. `LaT` uses the same tracking module of EdgeDuet. They both split the speed range into two groups and set the weight as 2.0 for the fast speed range and 1.0 for the slow speed range, as explained in Sec. VI-.2. The fast speed range is set as $[30 \text{ pixels/s}, +\infty)$, and the slow speed range is set as $[0, 30 \text{ pixels/s})$. We query the tracking results of the previous frame when the current frame is fed for all methods.

TABLE III
PARAMETERS OF THREE NETWORK CONDITIONS

Parameter	LTE	WiFi 2.4GHz	WiFi 5GHz
In Bandwidth (Kbps)	50000	40000	250000
Out Bandwidth (Kbps)	10000	33000	100000
In Delay (ms)	50	1	1
Out Delay (ms)	65	1	1

4) Network Setting: Since autonomous mobile vision applications are often deployed outdoors, the network connections vary. Accordingly, we compare the methods in different network settings. We connect the mobile device and the edge with WiFi 5GHz and emulate different types of networks with `Network Link Conditioner`, a developer tool provided by Apple. We use it to simulate different network conditions (LTE, WiFi 2.4GHz, WiFi 5GHz), and network bandwidths, Table III summarizes the parameters of the three network conditions. Network conditions with high network propagation delay is overlooked since our system requires offloading high-resolution frame.

5) Metrics: We evaluate the performance of different methods with the following metrics.

- *Latency*: The evaluation metric measures the delay of the detected objects. Lower latency will benefit the accuracy of object tracking. Since our detection results are composed of medium- to large-sized objects in frames from the local object detector and small objects in tiles from the remote object detector. We average latency for all objects, which is compatible with the definition of latency in EAAR. Specifically, supposing that there are M frames F_m uploaded to the edge server, the M-th frame is captured at time $t(F_m)$ and contains N_m objects. The bounding box and class label of each object O_m^n are sent to the client at time $t(O_m^n)$. Therefore, the latency can be

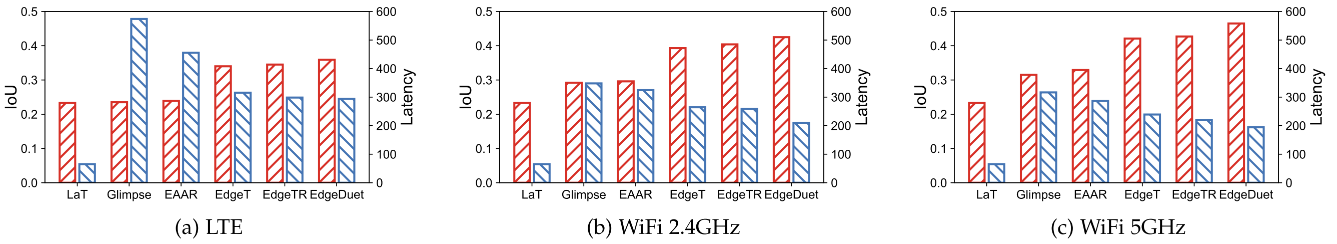


Fig. 11. End-to-end object detection accuracy (bars in red) and latency (bars in blue) of different methods under three network connections.

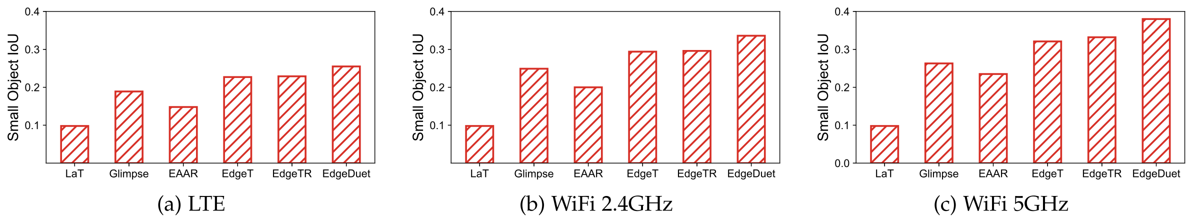


Fig. 12. Small object detection accuracy of different methods under three network connections. Pure local object detection (LaT) is excluded from subsequent evaluations for its low detection accuracy on small objects.

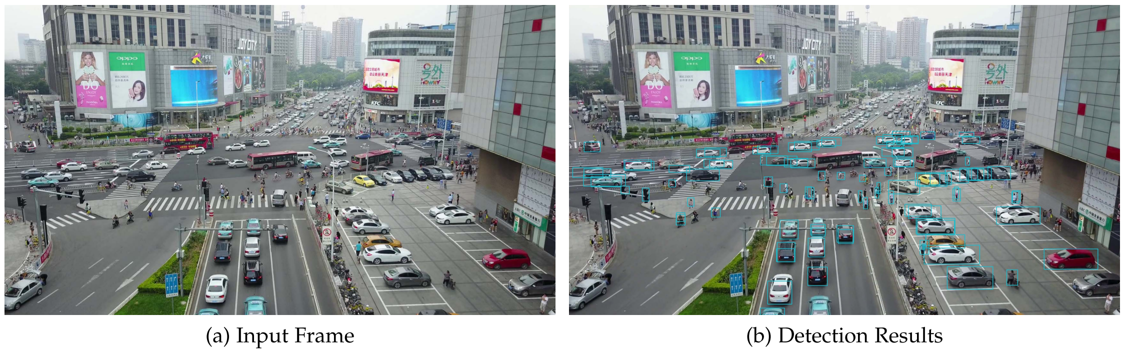


Fig. 13. Output sample of EdgeDuet.

obtained by:

$$Latency = \frac{1}{\sum_{m=1}^{M} N_m} \sum_{m=1}^{M} \sum_{n=1}^{N_m} t(O_m^n) - t(F_m) \quad (2)$$

- *Accuracy*: We use the average IoU [68] to measure the real-time object detection accuracy as in Glimpse and EAAR. The IoU of object o is defined as the Intersection over Union overlap with its ground-truth box.

$$IoU_o = \frac{|\text{BBox}(o) \cap \text{BBox}(g)|}{|\text{BBox}(o) \cup \text{BBox}(g)|} \quad (3)$$

where $|\cdot|$ is the area of the geometry, $\text{BBox}(\cdot)$ is the bounding box of the geometry. g is the ground truth of the object o. We take the object accurately detected if IoU $>=$ 0.8. The IoU is averaged over all objects in all frames.

B. End-to-End Performance

Fig. 11 summarizes the accuracy and latency of different methods under LTE, WiFi 2.4GHz, WiFi 5GHz network conditions. Fig. 12 highlights the accuracy of small objects. We present our observations and explain the results below. Fig. 13 shows the sample of detection results.

1) Overall Comparison: EdgeDuet notably outperforms the two offloading schemes, Glimpse and EAAR, in both accuracy and latency under all three network conditions. LaT is the fastest because it only performs local detection. However, pure local detection has the worst accuracy, especially for small object detection. EdgeDuet achieves 160.2%, 242.9%, 287.8% improvement in IoU metric for small object detection accuracy under the three network conditions, respectively. Under slow network connection, *e.g.*, LTE, LaT achieves similar accuracy with Glimpse and EAAR. This indicates the necessity of a local object detector when network conditions vary, which is common outdoors. Since LaT performs badly for small objects, we exclude it for the subsequent evaluations.

Among the variants of EdgeDuet, EdgeRT achieves a higher IoU (1.47%, 2.80%, 1.43%) and lower latency(5.42%, 2.16%, 8.28%) than EdgeT. This demonstrates the effectiveness of RoI Frame Encoding. EdgeDuet achieves a higher IoU (4.06%, 5.20%, 8.90%) and lower latency (1.47%, 18.79%, 11.41%) than EdgeTR. This shows the effectiveness of Content-prioritized tile offloading.

2) Comparison With Offloading Schemes on Latency: EdgeDuet achieves 48.8%, 39.6%, 38.6% latency improvement than Glimpse and 35.4%, 35.2%, 32.2% latency

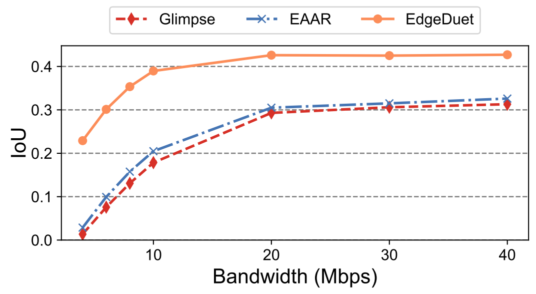


Fig. 14. Impact of network bandwidth.

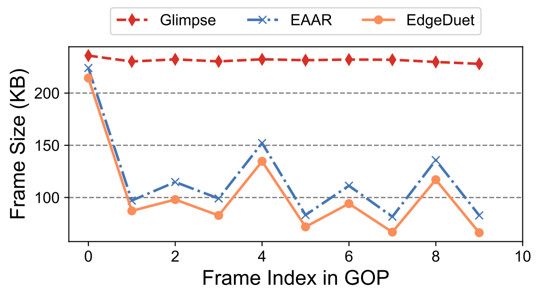


Fig. 15. Benefits of RoI frame encoding.

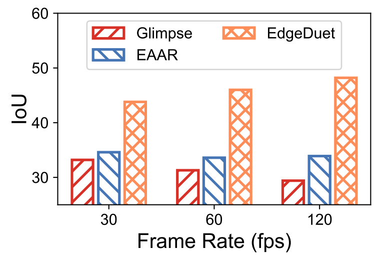


Fig. 16. Impact of frame rate.

improvement than EAAR under the three network conditions. The improvement in latency is more notable under slower networks, *e.g.*, LTE. EAAR achieves shorter latency than Glimpse since it transmits encoded videos instead of raw JPEGs. EdgeDuet is faster than EAAR for the following reasons.

- Detection of medium- to large-sized objects of EdgeDuet is from a local object detector. The latency of the local object detector is lower than offloading.
- Only small object detection is offloaded in EdgeDuet. Therefore fewer data need to be transmitted.
- EdgeDuet accelerates the offloading pipeline with tile-level parallelism. EAAR only implements task-level parallelism, so that the detection results have to wait for processing the entire frame.

3) Comparison With Offloading Schemes on Accuracy: EdgeDuet achieves 52.8%, 45.5%, 47.6% IoU improvement gain over Glimpse, 50.2%, 43.6%, 41.3% IoU improvement over EAAR under the three network conditions. EAAR achieves slightly better accuracy than Glimpse under LTE and WiFi 2.4GHz. The reason might be that motion vector based tracker behaves badly when latency increases. EdgeDuet yields the highest accuracy because it trades off between the tracker's accuracy and efficiency and employs adaptive tracking to update fast-moving objects.

4) Comparison With Offloading Schemes on Small Object Detection: EdgeDuet achieves 34.9%, 34.9%, 44.5% IoU improvement for small objects over Glimpse, 72.3%, 68.0%, 61.7% IoU improvement for small objects over EAAR. EAAR is worse than Glimpse for small object detection, although it has a higher overall detection accuracy. This is because small objects contain very few macroblocks to extract motion vectors, making it inaccurate to represent the object.

Among the variants of EdgeDuet, EdgeRT achieves a little higher IoU (0.9%, 0.7%, 3.4%) than EdgeT. EdgeDuet achieves higher IoU (11.4%, 13.5%, 14.5%) than EdgeTR. This demonstrates Content-prioritized offloading module contributes a lot for small object detection.

C. Impacting Factors on Overall Performance

1) Impact of Bandwidth: Fig. 14 shows the accuracy of different methods under different bandwidths. Thanks to the local object detector and optimized offloading, EdgeDuet consistently achieves better accuracy than Glimpse and EAAR. Particularly, when the bandwidth is limited (below 10Mbps), the accuracy of Glimpse and EAAR drops dramatically.

2) Impact of Frame Rate: Fig. 16 shows the accuracy of different methods when feeding videos of different frame rates. EdgeDuet consistently achieves higher accuracy than Glimpse and EAAR, even at 120fps. With the increase of frame rate, the accuracy of Glimpse drops. This is because the real-time tracker of Glimpse only works in an fps lower than 30fps. As for EAAR, increasing the frame rate motion does not impact the motion vector based tracker and thus the accuracy. An interesting finding is the accuracy of EdgeDuet increases with the frame rate. The reason may be that we use adaptive tracker configuration to update trackers of high speed objects frequently to reduce the influence of the skipped frames. Our tile-level parallelism may also help since once each tile's results are received, they do not wait for the new frame fed with high fps video input.

D. Benefits of Individual Modules in EdgeDuet

1) Benefits of RoI Frame Encoding: We evaluate the benefits of RoI frame encoding by comparing the offloading file size with EAAR and Glimpse. We average the bits count of frames with the same index in GOP. Since Glimpse only contains I frame, we only average the corresponding frames with the same frame index. Fig. 15 shows the average frame size of GOP. Since Glimpse does not apply inter-frame prediction and RoI frame encoding, its frame size is the largest, especially when the frame is encoded to P frame in EAAR and EdgeDuet. Since EdgeDuet does not offload medium- to large-sized objects, its frame size is smaller than EAAR.

2) Benefits of Content-Prioritized Tile Offloading: We show the benefits of content-prioritized tile offloading by comparing EdgeDuet with two variants. The variant Frame-Level encodes frames without splitting into tiles. The variant

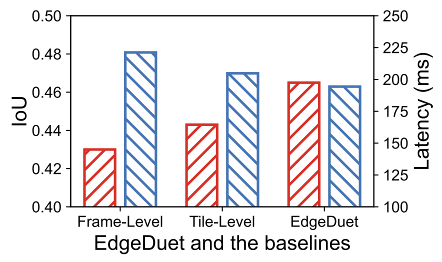


Fig. 17. Benefits of content-prioritized tile offloading.

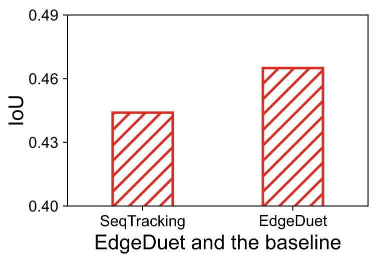


Fig. 18. Benefits of adaptive tracker.

`Tile-Level` splits frames into tiles, but does not change their priority. Fig. 17 shows the accuracy and latency of EdgeDuet and the two variants. EdgeDuet achieves 8.1% and 5.0% accuracy improvement over `Frame-Level` and `Tile-Level`. EdgeDuet achieves 12.1% and 5.1% latency improvement over `Frame-Level` and `Tile-Level`.

3) Benefits of Adaptive Tracker Configuration: We evaluate the benefits of adaptive tracker configuration by comparing EdgeDuet with a variant `SeqTracking` which sequentially updates each tracker. Fig. 18 shows the accuracy of EdgeDuet and `SeqTracking`. Our adaptive tracker configuration improves the overall accuracy by 4.7%.

IX. Limitations and Discussion

- **Commercial HEVC Codec Support.** EdgeDuet utilizes Kvazaar as the video codec. Since Kvazaar is a CPU-based video encoder, it requires lots of computing resources for high-resolution video compression, which burdens a lot for most mobile devices. Hardware acceleration will help it for efficient video compression, however, the commercial codecs have not supported the tile feature yet. We hope some AI-driven solutions like EdgeDuet will give more attention to the feature.
- **Adaptive Local Object Detector.** The performance of EdgeDuet is influenced by the local object detector. An efficient and accurate local object detector will reduce the size of the uploaded video frame and improve the accuracy of the trackers. However, the local object detector will compete for the computing resources with the trackers, which is not analyzed in EdgeDuet. We don't search the optimal local object detector choice according to the computing resources and the video input.

X. Conclusion

This paper presents EdgeDuet, the first splits object detection between the mobile device and the edge for accurate, real-time object detection on resource-constrained devices. Specifically, EdgeDuet offloads small object detection to the edge while detecting medium- to large-sized objects locally on the mobile device. EdgeDuet exploits RoI frame encoding and priority-based tile offloading to reduce the network traffic and accelerate the offloading pipeline. It also optimizes the cache detection results and tracker configurations for real-time object tracking. Evaluations on VisDrone, a video dataset from drone-mounted cameras, show that EdgeDuet outperforms local object detection in small object detection accuracy by 233.0%. It also improves the overall accuracy by 44.7% and end-to-end latency by 34.2% over the state-of-the-art offloading schemes, especially in low bandwidth and high frame-rate input.

References

[1] J. Xu et al., "IVR: Integrated vision and radio localization with zero human effort," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 3, no. 3, pp. 1–22, Sep. 2019.
[2] J. Xu et al., "Edge assisted mobile semantic visual SLAM," in *Proc. IEEE INFOCOM*, Jul. 2020, pp. 1828–1837.
[3] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (UAVs) for traffic monitoring," in *Proc. IEEE ICUAS*, May 2013, pp. 221–234.
[4] E. Su. (2020). *Roaming 'Robodog' Politely Tells Singapore Park Goers to Keep Apart*. [Online]. Available: https://www.reuters.com/
[5] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. ACM MobiCom*, Aug. 2019, pp. 1–16.
[6] *Augmented Reality—Apple Developer*. Accessed: Jul. 1, 2022. [Online]. Available: https://developer.apple.com/augmented-reality/
[7] *Build New Augmented Reality Experiences That Seamlessly Blend the Digital and Physical Worlds | ARCore | Google Developers*. Accessed: Jul. 1, 2022. [Online]. Available: https://developers.google.com/ar
[8] J. Xu et al., "FollowUpAR: Enabling follow-up effects in mobile AR applications," in *Proc. 19th Annu. Int. Conf. Mobile Syst., Appl., Services*, 2021, pp. 1–13.
[9] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
[10] F. O. Unel, B. O. Ozkalayci, and C. Cigla, "The power of tiling for small object detection," in *Proc. IEEE CVPR Workshops*, Jun. 2019, pp. 1–10.
[11] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. ACM SenSys*, 2015, pp. 155–168.
[12] M. Liu, X. Ding, and W. Du, "Continuous, real-time object detection on mobile devices without offloading," in *Proc. IEEE ICDCS*, Nov. 2020, pp. 976–986.
[13] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[14] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 583–596, Mar. 2015.
[15] P. Zhu et al., "Detection and tracking meet drones challenge," 2020, *arXiv:2001.06303*.
[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. NeurIPS*, 2015, pp. 1–9.
[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE CVPR*, Jun. 2016, pp. 779–788.
[18] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. ACM MobiCom*, Oct. 2018, pp. 115–127.
[19] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE ISCA*, Jun. 2016, pp. 243–254.
[20] X. He, Z. Zhou, and L. Thiele, "Multi-task zipping via layer-wise neuron sharing," in *Proc. NeurIPS*, 2018, pp. 1–11.
[21] R. Xu et al., "ApproxDet: Content and contention-aware approximate object detection for mobiles," in *Proc. 18th Conf. Embedded Netw. Sensor Syst.*, Nov. 2020, pp. 449–462.

[22] J. Zhang et al., "MobiPose: Real-time multi-person pose estimation on mobile devices," in *Proc. 18th Conf. Embedded Netw. Sensor Syst.*, Nov. 2020, pp. 136–149.

[23] V. Růžička and F. Franchetti, "Fast and accurate object detection in high resolution 4K and 8K video using GPUs," in *Proc. IEEE HPEC*, Sep. 2018, pp. 1–7.

[24] M. Gao, R. Yu, A. Li, V. I. Morariu, and L. S. Davis, "Dynamic zoom-in network for fast object detection in large images," in *Proc. IEEE CVPR*, Jun. 2018, pp. 6926–6935.

[25] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. ACM MobiSys*, Jun. 2016, pp. 123–136.

[26] J. Wang et al., "Bandwidth-efficient live video analytics for drones via edge computing," in *Proc. IEEE/ACM SEC*, Oct. 2018, pp. 159–173.

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

[28] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski, "Latency and throughput characterization of convolutional neural networks for mobile computer vision," in *Proc. 9th ACM Multimedia Syst. Conf.*, Jun. 2018, pp. 204–215.

[29] K. Du et al., "Server-driven video streaming for deep learning inference," in *Proc. ACM SIGCOMM*, 2020, pp. 557–570.

[30] J. Yi, S. Choi, and Y. Lee, "EagleEye: Wearable camera-based person identification in crowded urban spaces," in *Proc. ACM MobiCom*, Apr. 2020, pp. 1–14.

[31] T. Tan and G. Cao, "FastVA: Deep learning video analytics through edge processing and NPU in mobile," in *Proc. IEEE INFOCOM*, Jul. 2020, pp. 1947–1956.

[32] S. P. Chinchali, E. Cidon, E. Pergament, T. Chu, and S. Katti, "Neural networks meet physical networks: Distributed inference between edge devices and the cloud," in *Proc. ACM HotNets*, Nov. 2018, pp. 50–56.

[33] A. Galanopoulos, V. Valls, G. Iosifidis, and D. J. Leith, "Measurement-driven analysis of an edge-assisted object recognition system," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–7.

[34] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An overview of tiles in HEVC," *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 6, pp. 969–977, Dec. 2013.

[35] R. Skupin, Y. Sanchez, D. Podborski, C. Hellge, and T. Schierl, "HEVC tile based streaming to head mounted displays," in *Proc. IEEE CCNC*, Jan. 2017, pp. 613–615.

[36] J. Son, D. Jang, and E.-S. Ryu, "Implementing motion-constrained tile and viewport extraction for VR streaming," in *Proc. 28th ACM SIGMM Workshop Netw. Operating Syst. Support Digit. Audio Video*, Jun. 2018, pp. 61–66.

[37] T. Biedert, P. Messmer, T. Fogal, and C. Garth, "Hardware-accelerated multi-tile streaming for realtime remote visualization," in *EGPGV*, 2018, pp. 33–43.

[38] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2018, pp. 99–114.

[39] Y. Zhang, T. Wang, K. Liu, B. Zhang, and L. Chen, "Recent advances of single-object tracking methods: A brief survey," *Neurocomputing*, vol. 455, pp. 1–11, Sep. 2021.

[40] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. 7th Int. Joint Conf. Artif. Intell. (IJCAI)*, Vancouver, BC, Canada, vol. 2. San Francisco, CA, USA: Morgan Kaufmann, 1981, pp. 674–679.

[41] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with Siamese region proposal network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8971–8980.

[42] *High Efficiency Video Coding*, document ITU-T H.265, ISO, 2013.

[43] *x265, Free H.265/HEVC Encoder*. Accessed: Jul. 1, 2022. [Online]. Available: https://www.videolan.org/developers/x265.html

[44] *NVIDIA Video Codec SDK*. Accessed: Jul. 1, 2022. [Online]. Available: https://developer.nvidia.com/nvidia-video-codec-sdk

[45] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: Open-source HEVC/H.265 encoder," in *Proc. ACM MM*, Oct. 2016, pp. 1179–1182.

[46] *OpenHEVC*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/OpenHEVC/openHEVC

[47] *Intel Core i5-9500 vs Apple A13 Bionic*. Accessed: Jul. 1, 2022. [Online]. Available: https://gadgetversus.com/processor/intel-core-i5-9500-vs-apple-a13-bionic/

[48] *Information Technology—Programming Languages—C++*, Standard ISO/IEC 14882:2017, 2017.

[49] *Introduction to Framework Programming Guide*. Accessed: Jul. 1, 2022. [Online]. Available: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPFrameworks/Frameworks.html

[50] *Android NDK*. Accessed: Jul. 1, 2022. [Online]. Available: https://developer.android.com/ndk

[51] *NVIDIA Embedded Systems for Next-Gen Autonomous Machines*. Accessed: Jul. 1, 2022. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

[52] *OpenCV 4.2.0*. Accessed: Jul. 1, 2022. [Online]. Available: https://opencv.org/opencv-4-2-0

[53] *Kvazaar*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/ultravideo/kvazaar

[54] *Objective-C Runtime*. Accessed: Jul. 1, 2022. [Online]. Available: https://developer.apple.com/documentation/objectivec

[55] *CoreML*. Accessed: Jul. 1, 2022. [Online]. Available: https://developer.apple.com/documentation/coreml

[56] *ThreadPool*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/progschj/ThreadPool

[57] *KCFcpp*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/joaofaro/KCFcpp

[58] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[59] *How OpenCV-Python Bindings Works?* Accessed: Jul. 1, 2022. [Online]. Available: https://docs.opencv.org/3.4.9/da/d49/tutorial_py_bindings_basics.html

[60] *Ultralytics/YOLOv3*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/ultralytics/yolov3

[61] A. Paszke et al., "Automatic differentiation in PyTorch," in *Proc. NIPS Workshop Autodiff*, Long Beach, CA, USA, 2017. [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ

[62] *Multi-Process Service*, Santa Clara, CA, USA, 2014.

[63] J. Postel, "Transmission control protocol," Internet Eng. Task Force, Tech. Rep. 793, Sep. 1981, p. 85. [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt

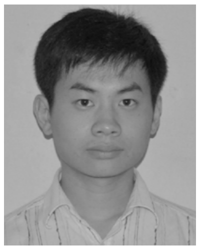
[64] *Sockpp*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/fpagliughi/sockpp

[65] *JSON*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/nlohmann/json

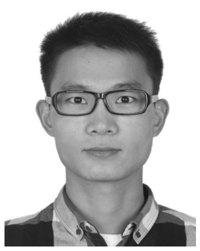
[66] *ConcurrentQueue*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/cameron314/concurrentqueue

[67] *Spdlog*. Accessed: Jul. 1, 2022. [Online]. Available: https://github.com/gabime/spdlog

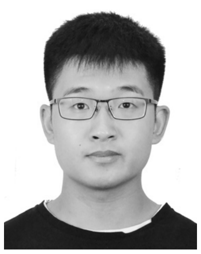
[68] M. Dantone, L. Bossard, T. Quack, and L. van Gool, "Augmented faces," in *Proc. IEEE ICCV Workshops*, Nov. 2011, pp. 24–31.



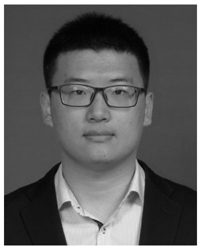
Zheng Yang (Fellow, IEEE) received the B.E. degree in computer science from Tsinghua University in 2006 and the Ph.D. degree in computer science from The Hong Kong University of Science and Technology in 2010. He is currently a Professor with Tsinghua University. His main research interests include wireless *ad-hoc*/sensor networks and mobile computing.



Xu Wang (Member, IEEE) received the Ph.D. degree from the School of Software, Tsinghua University, in 2020. He is currently a Post-Doctoral Researcher with the School of Software, Tsinghua University. He is a member of the Tsinghua National Laboratory for Information Science and Technology. His research interests include edge computing and machine learning.



Jiahang Wu (Student Member, IEEE) received the B.E. degree from the School of Software, Beijing Jiaotong University, in 2020. He is currently pursuing the M.E. degree with the School of Software, Tsinghua University. He is a member of the Tsinghua National Laboratory for Information Science and Technology. His research interests include the Internet of Things and edge computing.

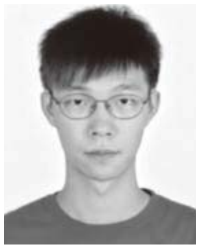


Yi Zhao (Student Member, IEEE) received the B.E. degree from the School of Software, Tsinghua University, in 2017, where he is currently pursuing the Ph.D. degree with the School of Software. He is a member of the Tsinghua National Laboratory for Information Science and Technology. His research interests include mobile computing and human mobility.

Qiang Ma (Member, IEEE) received the B.S. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, in 2013. He is currently an Assistant Researcher with Tsinghua University. His research interests include sensor networks, mobile computing, and data privacy.

Xin Miao (Member, IEEE) received the B.E. degree from the Department of Computer Science and Technology, Tsinghua University, and the Ph.D. degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology. He is currently with the School of Software, Tsinghua University. His research interests include wireless sensor networks, RFID, and mobile computing.

Li Zhang (Member, IEEE) received the Ph.D. degree from the School of Computer Science, Hefei University of Technology, Hefei, China, in 2009. She is currently a Professor with the School of Mathematics, Hefei University of Technology. Her current research interests include computer aided geometric design, computer graphics, the Internet of Things, and mobile computing.



Zimu Zhou (Member, IEEE) received the Ph.D. degree from The Hong Kong University of Science and Technology in 2015. He is currently an Assistant Professor (Tenure-Track) at the School of Information Systems (SIS), Singapore Management University (SMU). His research interests include model compression, federated machine learning, and applied machine learning.