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

Xu Wang*, Zheng Yang*‡, Jiahang Wu*, Yi Zhao* and Zimu Zhou†
*School of Software and BNRist, Tsinghua University
†School of Information Systems, Singapore Management University
‡ Corresponding Author
{wangxu2020,yangzheng}@tsinghua.edu.cn,wujx20@mails.thu.edu.cn,zhaoyi.yuan31@gmail.com, zimuzhou@smu.edu.sg

***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.**

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 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. *(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 [5]. However, local object detection with compressed models is sub-optimal for autonomous mobile vision because accurate small object detection requires high-resolution input [6], which easily overwhelm 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 wireless transmission of high-resolution videos to the edge. Long end-to-end detection delay leads to large detection errors as the mobile device's view is constantly changing [7].

Pioneer studies [7], [8] 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 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 [9], [10]. 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.

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 [7], [8]. The detection results of the current frame is obtained by adapting cached detection results of prior frames using lightweight trackers. 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

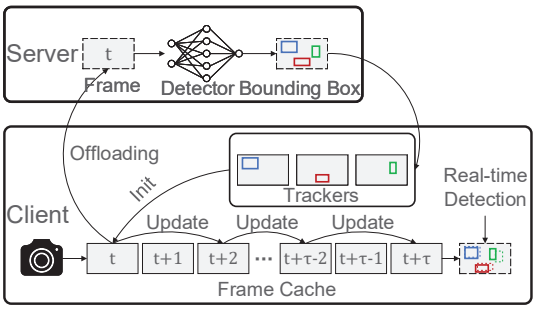


Fig. 1. An illustration of the popular "detect+track" framework for offloaded object detection [7], [8]. The detection results of the current frame are obtained by applying trackers on the cached detection results. The cached results are routinely updated by offloading key frames.

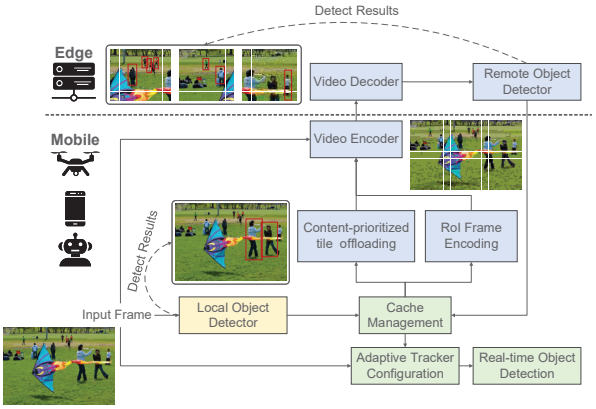


Fig. 2. An overview of EdgeDuet. Rectangles in different colors represent the three functional modules (offloaded small object detection (blue) (Sec. III), local object detector (yellow) (Sec. IV) and real-time tracking (green) (Sec. V). EdgeDuet is implemented as a cross-platform framework consisting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

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 [11], a public video dataset captured by drone-mounted cameras. Evaluations show that pure local object detection yields a detection accuracy (in terms of 0.232 of only 0.096 for small objects, while EdgeDuet achieves an accuracy of 0.319 for small objects. EdgeDuet also 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 [7], [8], especially in the case of low bandwidth and high input frame rate.

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 [7], [8] 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 [11] 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 [7], [8].

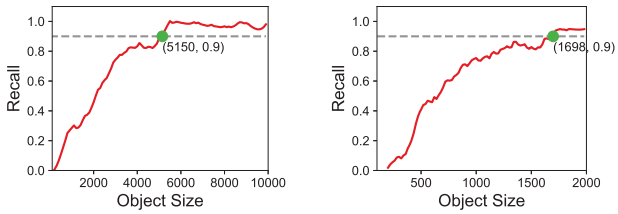
In the rest of this paper, we give an overview of EdgeDuet in Sec. II and elaborate on its functional modules in Sec. III, Sec. IV and Sec. V. We present the implementation of EdgeDuet in Sec. VI and the evaluations in Sec. VII. We review related work in Sec. VIII and finally conclude in Sec. IX.

II. 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; *(ii)* a *local object detector* module which detects large objects from low-resolution frames; and *(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. We elaborate on the detailed designs of each functional module in the subsequent sections.

III. 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) Car (b) Pedestrian

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

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 [12]–[14]. 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 [8], [15], the definition of RoI (*i.e.*, blocks containing small objects in our case) and the compression level vary 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. An 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 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.

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, the compression level is chosen 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 return to the device for offloading their blocks at the next frame.

3) Implementing RoI Frame Encoding: We use the Efficiency Video Coding (HEVC, a.k.a h.265) codec [16] to encode pixel blocks containing small objects to high quality and compress the rest of the frame to low quality. 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. 4a and Fig. 4b 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 [8], where the offloading process is split into frame encoding, frame upload, frame decoding, object detection, and result downloading. Nevertheless, existing work [8] 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.

1) Enabling Tile-Level Parallelism: A tile is a rectangular region in a frame defined in HEVC [17]. Fig. 4c shows an example of 5x3 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 [18]–[20] 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 [20], 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. 5(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. 5(b). Specifically, we break the bit-stream of a frame into a picture parameter set (PPB) and each tile's bit-streams. Consequently, each tile's bit-stream only depends on PPB and the tile encoding task. Hence the video encoder will first output the PPB, 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.

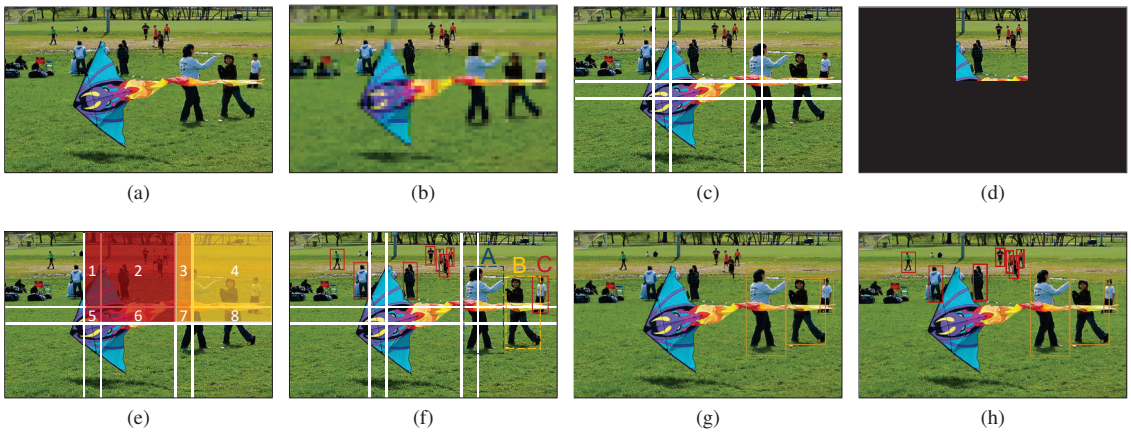


Fig. 4. An example of key steps in EdgeDuet. (a) Input frame. (b) Frame after RoI frame encoding, where blocks containing no small objects are compressed to low quality. (c) Tiles. (d) The output of video decoder after enabling tile-level parallelism. (e) Overlap-tiling. (f) Remote object detector results 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.

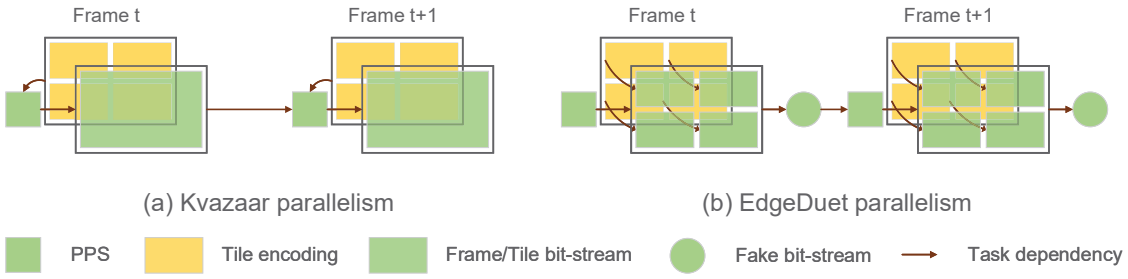


Fig. 5. Video encoding parallelism of Kvazaar and EdgeDuet.

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 [21]) accordingly. Fig. 4d 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*. Fig. 4e shows an example, where 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 together. 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. 4f. The overlap size (minimal width and height of overlap tiles) is set to the least multiple coding tree blocks, which is larger than the maximal size of small objects defined in Sec. III-A1 and complies to the tile definition in HEVC. Fig. 4f 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. III-B1) ensures tiles 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. 6). 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, 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,

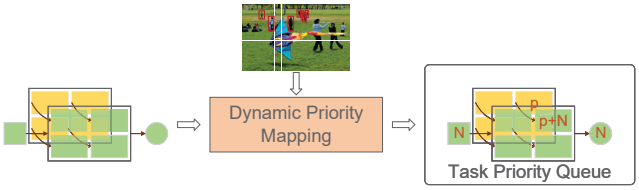


Fig. 6. An illustration of content-based tile priority.

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

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.

IV. 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 the 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 [7], [8]. Accuracy is measured by metrics such as IoU, as will be defined in Sec. VII-A5.

Table I shows the performance of different combinations of object detectors and input resolutions evaluated on the VisDrone [11] dataset with an iPhone 11. For resource efficiency, the models are quantized to `float16`. Based on the analysis, we choose YOLOv3 (640x640) as the local object detector. Fig. 4g 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 exhausted search on the optimal local object detector is out of the scope of this paper.

V. 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. EdgeDuet adopts multiple *single-object trackers* for object tracking, as in [7], [8]. Fig. 7 shows the general workflow.

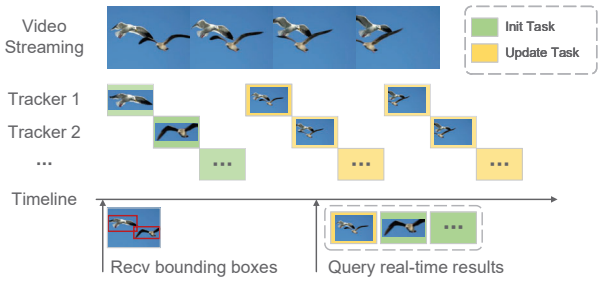


Fig. 7. General workflow using multiple single-object trackers.

Trackers returns 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. 4h 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 [22] as our single-object tracker since it is both faster and more accurate than the optical flow based tracker in [7] and has a higher accuracy than the motion vector based tracker in [8].
- *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 to 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.

VI. Implementation

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

A. 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.

1) Video Decoder: The video decoder is implemented in C++ based on the `OpenHEVC` library [21]. We modify the library to support tile-based parallel decoding as in Sec. III-B1. We use OpenCV-Python bindings [23] 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 [24] model in PyTorch [25]. We run the model in multiple processes for parallel inference on tiles. The GPU is set to CUDA Multi-Process Service mode [26] to reduce GPU context switching.

B. Implementation of Core Device-Side Modules

We implement the device-side of EdgeDuet on an iPhone 11, with the A13 Bionic chip embedded with a four-core GPU. The device-side modules of EdgeDuet consist of a video streamer, a video encoder, a local object detector, and an object tracker. Most modules on the device side are implemented in C++ 17 [27] for easy deployment on different platforms such as iOS Frameworks [28] and Android NDK [29].

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 [30] 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` [31], an open-source HEVC encoder. We modify the library to support tile-based parallel encoding and priority, as described in Sec. III-B1 and Sec. III-B2. 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 [32] with `Core ML` [33], which optimizes on-device performance by jointly leveraging the CPU, GPU, and Neural Engine. We use the pre-trained compressed YOLOv3 model YOLOv3FP16 (640x640). for medium- to large-sized object detection, as explained in Sec. IV. 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` [22] Tracker and `ThreadPool` [34] to schedule multiple object tracking. We use the implementation of `KCFcpp` [35] without the HOG features [36] for fast object tracking, as described in Sec. V-2.

VII. Evaluation

This section presents the evaluations of EdgeDuet.

A. Experiment Setup

1) Datasets: We compare different methods on VisDrone [11], a dataset of videos captured by drone-mounted cameras. We filter out the low-resolution videos and only keep six 2K videos (2560x1440) captured along a street. We upsample the origin 30fps videos to 60/120 fps with Super-SloMo [22] to evaluate the performance with the video frame rate. Although the VisDrone dataset contains annotated bounding boxes, some small objects are not annotated. Therefore we use the outputs (cars and pedestrians) of a YOLOv3-spp (2560x2560) model to re-annotate these videos and treat them as the ground truth, as with other recent studies on video analytics [15], [37], [38].

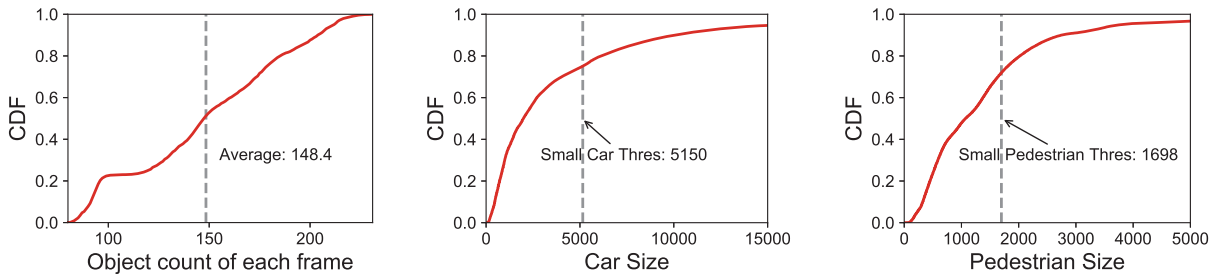
Fig. 8 shows certain vital statistics of the VisDrone dataset. From Fig. 8a, the average number of objects in each frame is 148.4. As we will show later, adaptive tracker configuration is beneficial to track such many objects in real-time. From Fig. 8b and Fig. 8c, 75.1% of cars and 71.8% of pedestrians 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` [7]: 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` [8]: 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.

3) Implementation and Settings of Compared Methods: The implementation of EdgeDuet and `LaT` can be found in Sec. VI. 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 fair comparison. For `Glimpse`, we encode 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 1x4 tiles and pack each tile into one slice, as the default setting. EdgeDuet splits the frame into 5x3 tiles for offloading, as in Fig. 4c.
- *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 2560x2560. `EAAR` operates on 1280x1280 but returns the detection results in 2560x2560. This setting allows a lower latency of `EAAR` compared with its origin dependency aware inference. EdgeDuet operates on 960x960 for overlap-tiling inference. The overlap size is set to 128 (2 macroblocks) as in Sec. III-B1.
- *Real-time Object Tracking*. We implement the optical flow based tracker with `calcOpticalFlowPyrLK` for



(a) CDF of object count in each frame (b) CDF of car size (c) CDF of pedestrian size

Fig. 8. Characteristics of VisDrone [11] dataset. There are a number of objects in each frame, and most objects are small in size.

`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 speed range into two groups and set weight as 2.0 for fast speed range and 1.0 for slow speed range, as explained in Sec. V-2. We query the tracking results of the previous frame when the current frame is feed for all methods.

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.

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

- *Latency*: Since our detection results are composed of medium- to large-sized objects in frames from local object detector and small objects in tiles from remote object detector. We average latency for all objects, which is compatible with the definition of latency in `EAAR`.
- *Accuracy*: We use the average IoU [39] to measure the real-time object detection accuracy as in `Glimpse` and `EAAR`. The IoU is averaged over all objects in all frames.

B. End-to-End Performance

Fig. 9 summarizes the accuracy (IoU) and latency of different methods under LTE, WiFi 2.4GHz, WiFi 5GHz network conditions. Fig. 10 highlights the accuracy of small objects. We present our observations and explain the results below.

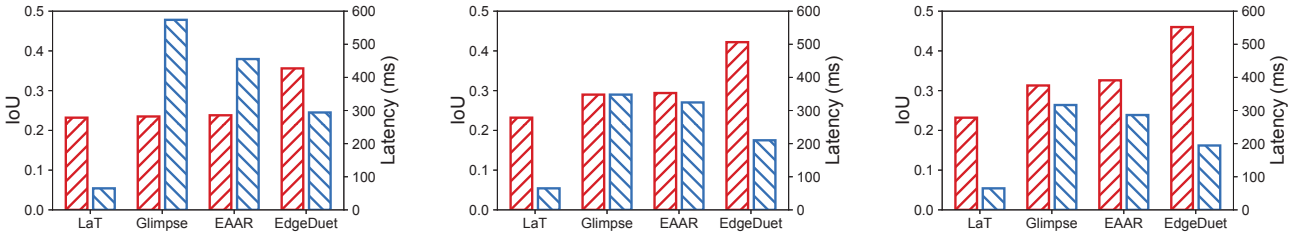
1) Overall Comparison: EdgeDuet notably outperforms the two offloading schemes, `Glimpse` and `EAAR`, in both accuracy and latency under all the three network conditions. `LaT` is the fastest because it only performs local detection. However, pure local detection has the worst accuracy, especially for small object detection. EdgeDuet achieves 161.5%, 245.0%, 292.4% improvement 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.

2) Comparison with Offloading Schemes on Latency: EdgeDuet achieves 48.7%, 39.6%, 38.6% latency improvement than `Glimpse` and 35.4%, 35.2%, 32.1% latency 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.

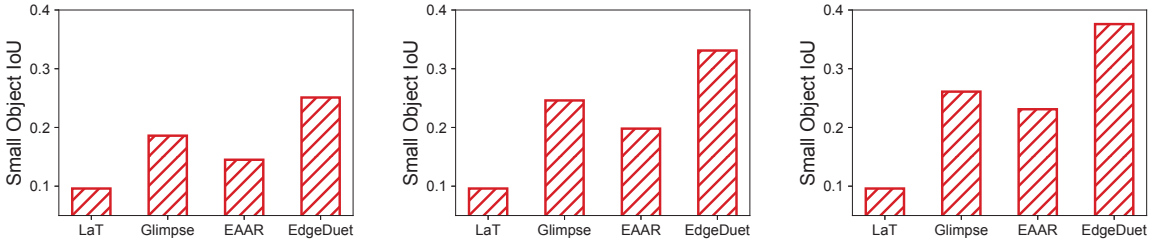
3) Comparison with Offloading Schemes on Accuracy: EdgeDuet achieves 51.6%, 45.3%, 47.0% accuracy improvement gain over `Glimpse` and 49.5%, 43.4%, 41.1% accuracy improvement over `EAAR` under the three network conditions. `EAAR` achieves slightly better accuracy than `Glimpse` under LTE and WiFi 2.4GHz. 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 35.2%, 34.3%, 44.3% small object accuracy improvement over `Glimpse` and 73.3%, 67.0%, 62.6% small object accuracy improvement over `EAAR`. `EAAR` is worse than `Glimpse` for small object detection, although it has a higher overall detection accuracy. This is



(a) LTE (b) WiFi 2.4GHz (c) WiFi 5GHz

Fig. 9. End-to-end object detection accuracy (bars in red) and latency (bars in blue) of different methods under three network connections.



(a) LTE (b) WiFi 2.4GHz (c) WiFi 5GHz

Fig. 10. 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.

because small objects contain very few macroblocks to extract motion vectors, making it inaccurate to represent the object.

C. Impacting Factors on Overall Performance

1) Impact of Bandwidth: Fig. 11 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. 13 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. 12 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 `Tile-Level` splits frames into tiles, but does not change their priority. Fig. 14 shows the accuracy and latency of EdgeDuet and the two variants. EdgeDuet achieves 7.3% and 4.3% accuracy improvement over `Frame-Level` and `Tile-Level`. EdgeDuet achieves 12.2% 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. 15 shows the accuracy of EdgeDuet and `SeqTracking`. Our adaptive tracker configuration improves the overall accuracy by 4.2%.

VIII. 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

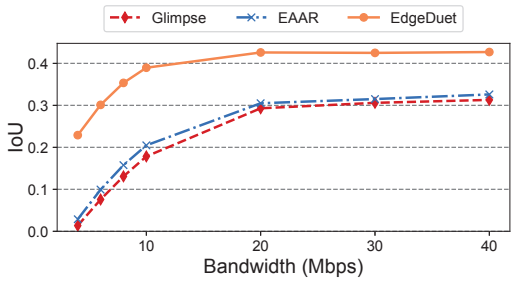


Fig. 11. Impact of Network Bandwidth

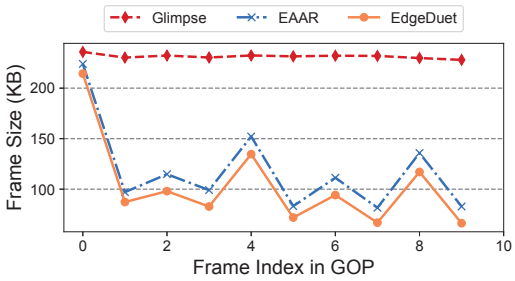


Fig. 12. Benefits of RoI Frame Encoding

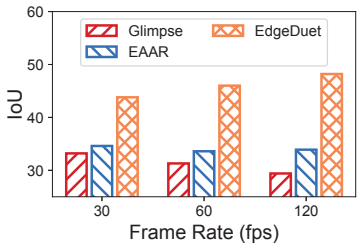


Fig. 13. Impact of Frame Rate

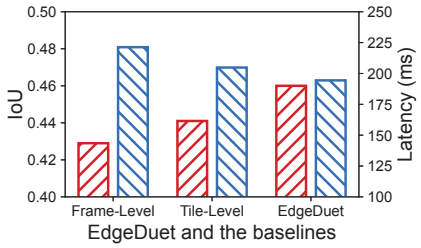


Fig. 14. Benefits of Content-Prioritized Tile Offloading

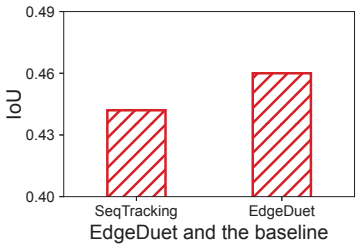


Fig. 15. Benefits of Adaptive Tracker

such as two-stage models *e.g.*, Faster-RCNN [40] and one-stage models *e.g.*, YOLO [41]. Model compression and acceleration techniques [5], [42]–[44] 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 [6]. For accurate and fast small object detection, customized models [6], [13], [14] have been developed to detect objects on sub-regions of the input image/video. 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 capabilities [41] 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 [7], [8], [12], [15], [45]–[51]. 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 [7] proposes to only send trigger frames and proposes the "detect + track" framework for fast object detection. EAAR [8] 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 [15] 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.

IX. 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.

Acknowledgement

This work is supported in part by the NSFC under grant 61832010, 61632008, 61872081, 61972131. Zimu Zhou's research is supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

REFERENCES

[1] J. Xu, H. Chen, K. Qian, E. Dong, M. Sun, C. Wu, L. Zhang, and Z. Yang, "ivr: Integrated vision and radio localization with zero human effort," in *PACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, Sep 2019.

[2] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual slam," in *Proceedings of the IEEE INFOCOM*, 27-30 April 2020.

[3] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (uavs) for traffic monitoring," in *Proc. of IEEE ICUAS*, 2013.

[4] E. Su, "Roaming 'robodog' politely tells singapore park goers to keep apart," https://www.reuters.com/, 2020.

[5] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[6] F. Ozge Unel, B. O. Ozkalayci, and C. Cigla, "The power of tiling for small object detection," in *Proc. of IEEE CVPR Workshops*, 2019.

[7] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. of ACM SenSys*, 2015.

[8] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. of ACM MobiCom*, 2019.

[9] M. Liu, X. Ding, and W. Du, "Continuous, real-time object detection on mobile devices without offloading," in *Proc. of IEEE ICDCS*, 2020.

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

[11] P. Zhu, L. Wen, D. Du, X. Bian, Q. Hu, and H. Ling, "Vision meets drones: Past, present and future," *arXiv preprint arXiv:2001.06303*, 2020.

[12] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *Proc. of IEEE/ACM SEC*, 2018.

[13] V. Růžička and F. Franchetti, "Fast and accurate object detection in high resolution 4k and 8k video using gpus," in *Proc. of IEEE HPEC*, 2018.

[14] 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. of IEEE CVPR*, 2018.

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

[16] H. E. V. Coding and I. Rec, "H. 265 and iso," 2013.

[17] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An overview of tiles in hevc," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 969–977, 2013.

[18] *x265, the free H.265/HEVC encoder*. [Online]. Available: https://www.videolan.org/developers/x265.html

[19] *NVIDIA Video Codec SDK*. [Online]. Available: https://developer.nvidia.com/nvidia-video-codec-sdk

[20] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: open-source hevc/h. 265 encoder," in *Proc. of ACM MM*, 2016.

[21] *OpenHEVC*. [Online]. Available: https://github.com/OpenHEVC/openHEVC

[22] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2014.

[23] *How OpenCV-Python Bindings Works?* [Online]. Available: https://docs.opencv.org/3.4.9/da/d49/tutorial_py_bindings_basics.html

[24] *ultralytics/yolov3*. [Online]. Available: https://github.com/ultralytics/yolov3

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[26] N. NVIDIA, "Multi-process service," 2014.

[27] ISO, "Iso/iec 14882: 2017 information technology—programming languages—c++." 2017.

[28] *Introduction to Framework Programming Guide*. [Online]. Available: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPFrameworks/Frameworks.html

[29] https://developer.android.com/ndk, *Android NDK*.

[30] *OpenCV 4.2.0*. [Online]. Available: https://opencv.org/opencv-4-2-0

[31] *Kvazaar*. [Online]. Available: https://github.com/ultravideo/kvazaar

[32] https://developer.apple.com/documentation/objectivec, *Objective-C Runtime*.

[33] *CoreML*. [Online]. Available: https://developer.apple.com/documentation/coreml

[34] *ThreadPool*. [Online]. Available: https://github.com/progschj/ThreadPool

[35] *KCFcpp*. [Online]. Available: https://github.com/joaofaro/KCFcpp

[36] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[37] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proc. of ACM SIGCOMM*, 2018.

[38] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proc. of ACM SIGCOMM*, 2018.

[39] M. Dantone, L. Bossard, T. Quack, and L. Van Gool, "Augmented faces," in *Proc. of IEEE ICCV Workshops*, 2011.

[40] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proc. of NeurIPS*, 2015.

[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. of IEEE CVPR*, 2016.

[42] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. of ACM MobiCom*, 2018.

[43] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *Proc. of ACM/IEEE ISCA*, 2016.

[44] X. He, Z. Zhou, and L. Thiele, "Multi-task zipping via layer-wise neuron sharing," in *Proc. of NeurIPS*, 2018.

[45] 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. of ACM MobiSys*, 2016.

[46] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proc. of IEEE INFOCOM*, 2018.

[47] 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 *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018, pp. 204–215.

[48] J. Yi, S. Choi, and Y. Lee, "Eagleeye: wearable camera-based person identification in crowded urban spaces," in *Proc. of ACM MobiCom*, 2020.

[49] T. Tan and G. Cao, "Fastva: Deep learning video analytics through edge processing and npu in mobile," in *Proc. of IEEE INFOCOM*, 2020.

[50] 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. of ACM HotNets*, 2018.

[51] A. Galanopoulos, V. Valls, G. Iosifidis, and D. J. Leith, "Measurement-driven analysis of an edge-assisted object recognition system," *arXiv preprint arXiv:2003.03584*, 2020.