Flexible High-resolution Object Detection on Edge Devices with Tunable Latency

Shiqi Jiang[1], Zhiqi Lin[2†], Yuanchun Li[1], Yuanchao Shu[1], Yunxin Liu[3†*]
[1]Microsoft Research [2]University of Science and Technology of China
[3]Institute for AI Industry Research (AIR), Tsinghua University
[1]{shijiang, yuanchun.li, yuanchao.shu}@microsoft.com
[2]zhiqi.0@mail.ustc.edu.cn, [3]liuyunxin@air.tsinghua.edu.cn

ABSTRACT

Object detection is a fundamental building block of video analytics applications. While Neural Networks (NNs)-based object detection models have shown excellent accuracy on benchmark datasets, they are not well positioned for high-resolution images inference on resource-constrained edge devices. Common approaches, including down-sampling inputs and scaling up neural networks, fall short of adapting to video content changes and various latency requirements. This paper presents Remix, a flexible framework for high-resolution object detection on edge devices. Remix takes as input a latency budget, and come up with an image partition and model execution plan which runs off-the-shelf neural networks on non-uniformly partitioned image blocks. As a result, it maximizes the overall detection accuracy by allocating various amount of compute power onto different areas of an image. We evaluate Remix on public dataset as well as real-world videos collected by ourselves. Experimental results show that Remix can either improve the detection accuracy by 18%-120% for a given latency budget, or achieve up to 8.1× inference speedup with accuracy on par with the state-of-the-art NNs.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; • **Computing methodologies → Neural networks**.

KEYWORDS

Edge Computing, Deep Neural Networks, Video Analytics, Object Detection, Tunable Latency

ACM Reference Format:
Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, Yunxin Liu. 2022. Flexible High-resolution Object Detection on Edge Devices with Tunable Latency. In *The 27th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '21), January 31-February 4, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3447993.3483274

[†]This work was done when authors were with Microsoft Research.
[*]Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACM MobiCom '21, January 31-February 4, 2022, New Orleans, LA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8342-4/22/01…$15.00
https://doi.org/10.1145/3447993.3483274

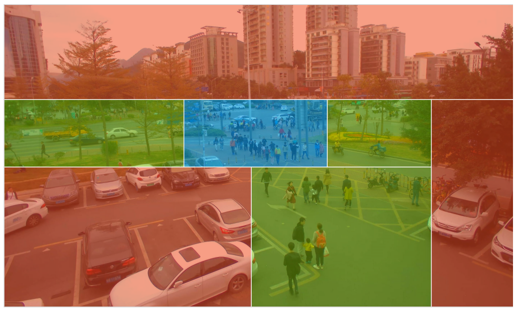


Figure 1: An illustration of Remix for pedestrian detection in a 4K image. For the blue block in the middle, an expensive network is applied due to the high density of objects. Networks of medium sizes are used in green blocks. Red blocks are processed using cheap networks or even skipped.

1 INTRODUCTION

Organizations are deploying cameras at scale for video analytics. For instance, large camera networks are installed in cities (*e.g.,* London, Beijing, New York) for public safety and urban planning [1, 5, 71], and cameras on enterprise campuses (*e.g.,* corporate offices, retail shops) are used to improve business operations [41] and shopping experience [11, 54]. In the vast majority of video analytics applications, object detection plays a pivotal role due to its ability to identify and localize all objects instances of certain categories in an image. Driven by advances in machine learning and hardware acceleration, a number of deep convolutional neural network (CNN)-based object detection techniques have been proposed and with excellent performance achieved on benchmark datasets [6, 26, 49].

In many video analytics scenarios, video feeds from cameras are analyzed on edge devices placed on-premise to accommodate limited network bandwidth and privacy requirements [17, 21, 40, 77, 78]. Edge compute, however, is provisioned with limited resources, posing significant challenges on running object detection NNs for live video analytics.

The inherent tension between resources-constrained edge devices and compute-intensive inference workloads is worsened by the ever-increasing video quality from high resolution cameras. Prices of high-resolution cameras have been falling off steadily in recent years. Nowadays people can get a 4K security camera for less than 100 USD, an order of magnitude cheaper than what was five years ago [4, 12]. Some consumer devices are even equipped with 64 mega-pixel high-resolution camera which offers 8K@24FPS video capturing ability [10]. To take advantage of the smoother and more detailed image, object detection networks have to be designed with a much large capacity (*e.g.,* more convolutional layers, higher dimension, etc.) to work with high-resolution inputs, resulting in a prohibitively high latency when running on edge devices [65, 69, 70, 85].

To fill the gap between the needs of fast and accurate object detection in high-resolution videos and the limited resources on edge devices, we present Remix, a flexible framework with tunable latency of object detection in high-resolution videos. Remix strikes the balance between edge resources and object detection performance by *non-uniformly* partitioning high-resolution images, and runs *diverse* object detection networks *selectively* in image blocks. Unlike existing solutions that run one network on image blocks of equal size [65, 85, 90], Remix uses edge devices' compute cycles judiciously in areas that tend to be more informative, hence resulting in a higher overall detection accuracy. Fig. 1 shows an example of our key idea behind Remix for pedestrian detection in a 4K image.

Design of Remix, however, faces two core technical challenges. First, the search space of image partition and network selection is huge, and it is non-trivial to decide an optimal partition plan, best balancing the trade-off between accuracy and latency. Second, executing partitions effectively requires the awareness of the variations of video content, the NNs' capabilities as well as edge resource availability, which is unknown and difficult to predict. To cope with the first issue, Remix learns the long-term spatial distribution of objects from the history, and builds an efficient resource-accuracy profiler which estimates object detection performance of various networks on objects of different sizes. As a result, Remix effectively reduces the search space by approximately 35 orders of magnitude, and generates a group of coarse-grained partition plans within 3 minutes for a 4K video feed on an NVIDIA Jetson device [7] (§3.1). To deal with short-time content variations, Remix features a closed-loop selective execution module. It takes as input the live image and coarse-grained partition plans, and leverages previous detection results as the feedback to dynamically determine which blocks can be skipped, finding the solution that maximizes detection accuracy under the latency budget at runtime (§3.2). In summary, the main contributions are as follows.

- We propose Remix, a flexible high-resolution object detection framework. Remix makes the best of edge compute resources and runs off-the-shelf object detection networks in high-resolution images at a fine-grained level of latency and accuracy.
- We conduct a measurement study and identify several key aspects on the design of high-resolution object detection on edge devices. The adaptive partition and selective execution module as well as a series of practical techniques are designed to efficiently find the best image split plan, handle temporal variations of video content and run various detection models.
- We implement Remix on a Jetson edge device as well as two popular mobile system-on-chips (SoCs). Evaluation results on multiple 4K videos show that Remix achieves 1.7×-8.1× speedup with an accuracy drop of 0.2% from the state-of-the-art (SOTA) object detection networks on devices with different capacities. Compared with SOTA techniques under the same latency constraint, Remix obtained 65.3% detection accuracy improvements on average.

Neural Network	Variant	Backbone	Input Size	Param.(M)	Size (MB)
EfficientDet	D0	EfficientNet-B0	512^2	4.14	16.2
	D1	EfficientNet-B1	640^2	7.07	27.8
	D2	EfficientNet-B2	768^2	8.79	34.6
	D3	EfficientNet-B3	896^2	13.3	52.5
	D4	EfficientNet-B4	1024^2	23.0	90.2
	D5	EfficientNet-B5	1280^2	37.4	146.4
	D6	EfficientNet-B6	1280^2	58.8	230.1
	D7	EfficientNet-B7	1536^2	59.0	232.2
RetinaNet	RN-50-640	ResNet-50	640^2	51.8	202.7
	RN-152-640	ResNet-152	640^2	86.4	337.8
	RN-50-1024	ResNet-50	1024^2	52.2	203.9
	RN-152-1024	ResNet-152	1024^2	86.7	339.0
FasterRCNN	FRCN-152-1024	ResNet-152	1024^2	63.2	497
	FRCN-ICT-1024	Inception	1024^2	60.0	476
SSD-MobileNet	FPNLite-320	MobileNet V2	320^2	29.2	11.4
	FPNLite-640	MobileNet V2	640^2	30.7	12.1
YOLO V3	YOLO-416	DarkNet-53	416^2	62.0	237

Table 1: Selected object detection NNs.

2 MOTIVATIONAL STUDY

Since the majority of object detection networks are designed with low-resolution inputs (*e.g.,* 416×416 for YOLOv3), common practices of high-resolution object detection seek to downscale inputs (*e.g.,* image resizing, partition) or scale up network's input resolution [56, 69, 70]. The former could hurt detection accuracy while the latter results in large networks that take longer to train and will be much slower to infer. In this section, we quantify the trade-off through a measurement study, and identify the key opportunities – in efficiency and accuracy – of adaptive partition and selective execution.

2.1 Experimental Setup

We conduct the experiments on a typical edge device, Nvidia Jetson AGX Xavier [7] with TensorFlow [14] being the inference engine. As listed in Table 1, in total we select 17 object detection NNs based on five typical backbones: EfficientDet [69, 70], RetinaNet [37, 48], Faster-RCNN [64], YOLO [63], and SSD-MobileNet [53, 66].

To evaluate detection accuracy on high resolution inputs, we use the PANDA dataset [9, 74] that contains 555 images and 15 video sequences captured from multiple scenes. In total, it contains 15 million bounding boxes of persons and vehicles. Considering common camera and edge compute capability, we resize images and videos from the original resolution of 26,753×15,052 into 3,840×2,160 (4K), denoted as PANDA 4K dataset. We fine-tune the NNs using 80% of the images and use the rest 20% for testing. We use mean averaged precision (mAP) [6], a widely-used metric in computer vision to evaluate object detection accuracy. In order to reliably profile inference latency, we set the power mode of Jetson to MAXN [8], and execute each NN for 100 times and report the averaged results.

To explore the design space of high-resolution object detection, we measure the performance of three types of solutions: *1) down-sampling inputs; 2) up-scaling networks; and 3) uniform partitioning.*

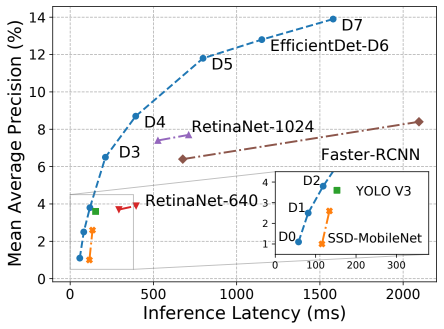


Figure 2: mAP and GPU inference latency of selected NNs on the PANDA 4K dataset with Jetson.

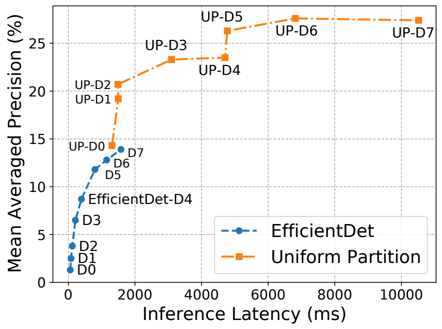


Figure 3: mAP and GPU inference latency of uniform partitioning on the PANDA 4K dataset with Jetson.

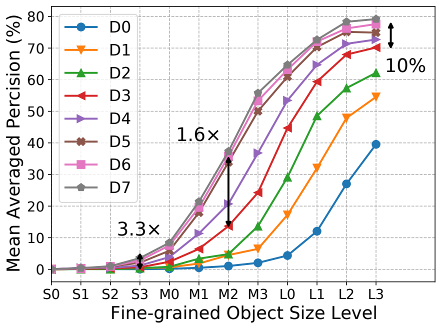


Figure 4: mAP of EfficientDet NNs on objects of different sizes in the PANDA 4K dataset.

2.2 Observations

From the measurement results, we make the following four key observations.

Down-sampling inputs reduces the latency but has a low accuracy. Down-sampling is a widely-used practice that reduces the resolution of images to match a network's input size. Fig. 2 shows the mAP and latency of the 17 networks. Clearly, small NNs are much faster than large ones. For example, SSD-MobileNet and EfficientDet-D0 have a latency of only 134ms (7.4 FPS) and 57ms (17.5 FPS) which are 11.7× and 27.7× faster than EfficientDet-D7 (1,579ms), respectively. However, those small networks have an extremely low mAP, as low as 0.8% for SSD-MobileNet and 1.1% for EfficientDet-D0. Such a low accuracy is caused by the small size of objects after down-sampling. For 4K images in the PANDA dataset, nearly 80% of the objects are larger than 58^2 pixels. However, when down sampled to 320×320, the input size of SSD-MobileNet, nearly half of the objects are smaller than 2.8^2 pixels, which is way too small to be detected by existing neural networks[1].

Up-scaling networks increases the accuracy but has a high latency. Up-scaling networks, *i.e.,* building larger NNs or scaling up existing ones, is another means to deal with high-resolution images [37, 56, 67, 69, 70, 82]. This approach would help accuracy but incur significant latency. We use EfficientDet [70], an efficient network with SOTA detection accuracy as an example. As shown in Fig. 2, with up-scaled EfficientDet from D0 to D7, the inference latency increases exponentially. The input size of EfficientDet-D7 is only 1,536×1,536 (Table 1), but its latency is higher than 2 seconds. It will be less practical to further scale up EfficientDet to process 4K images, due to the unaffordable high latency on edge devices. Furthermore, when the inference latency gets dramatically higher, the accuracy improvement becomes marginal. For instance, D6 has more than 30% inference latency than D5, but the mAP only improves by less than 0.6%.

Uniform partitioning may further increase the accuracy but has an even higher latency. Recent work also seeks to use uniform partitioning to divide a large image into equal chunks, and processes each chunk using the same network [65, 85, 90]. With this approach, the details of objects are kept and more objects in total could be detected, thus improving the overall accuracy. We apply uniform partitioning (UP) strategy with EfficientDet on the PANDA 4K dataset, denoted as UP-D0, UP-D1 etc. As shown in Fig. 3, the strategy significantly improves the accuracy. UP-D0 achieves an mAP of 14.3%, more than 13× better compared with EfficientDet-D0, and it is also higher than the accuracy of EfficientDet-D7 (13.9%). UP-D6 even achieves an mAP of as high as 27.6%. However, due to the cost of running NN for each individual chunk, uniform partitioning also leads to a prohibitively high latency, *e.g.,* more than 10s for UP-07 on Jetson.

Level	Small				Medium				Large			
FG-Level	S0	S1	S2	S3	M0	M1	M2	M3	L0	L1	L2	L3
Min Area	0	8	16	24	32	48	64	80	96	128	160	196
Max Area	8	16	24	32	48	64	80	96	128	160	196	∞

Table 2: Fine-grained (FG) object-size levels and their areas in Pixel^2, *e.g.,* S0 ranges from 0^2 to 8^2 pixels.

The accuracy of NNs varies among objects of different sizes. We also measure the detection accuracy of EfficientDet on objects of different sizes. To do so, we categorize object sizes into 12 fine-grained levels from small to large, as shown in Table 2.

Fig. 4 shows the results. Overall, NNs obtain higher detection accuracy as object size increases. However, the differences diminish with the increase of object size. For instance, small networks like EfficientDet-D3 can still obtain 67.9% and 70% mAP on L2 and L3, while the best mAP achieved by D7 are only 78.3% and 79%, respectively.

More importantly, marginal accuracy gain from different neural networks varies significantly across object sizes. Let's take EfficientDet-D6 and EfficientDet-D3 as an example. EfficientDet-D6 has a latency 5.4× higher than D3. If we run EfficientDet-D6 instead of EfficientDet-D3 on L3 objects, we will end up getting an mAP improvement of approximately 10%, shown in Fig. 4. Alternatively, if we replace D3 with EfficientDet-D6 on S3 and M2 objects, the mAP improvements can be as large as 3.3× and 1.6×, respectively. Such a diverse marginal gain on objects of various sizes calls for careful NN selection, which is absent from existing partitioning-based methods.

[1]Note that even the best mAP achieved by EfficientDet-D7 in Fig. 2 is only 13.9%. It is because many images in PANDA 4K contain way too many objects (>100) than what current NNs can detect.

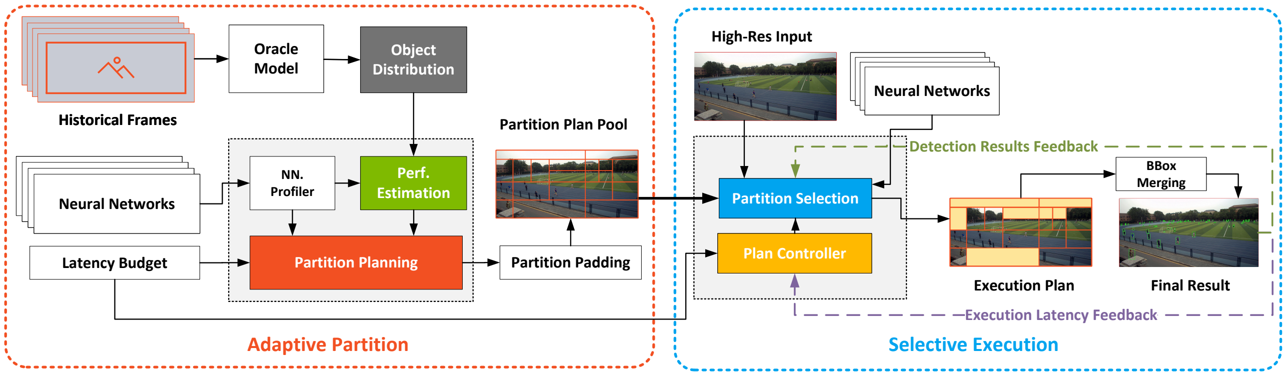


Figure 5: Remix system overview.

2.3 Opportunities and Challenges

The above observations motivate us to design a new system to achieve a better trade-off between accuracy and latency for high-resolution object detection. Specifically, we argue image-partitioning is a promising approach but existing uniform partitioning misses two critical opportunities. First, it doesn't leverage the *spatial distribution of objects*. As shown in Fig. 1, objects of interest are often distributed non-uniformly. A large part of the image does not contain any objects of interest and thus can be ignored to reduce latency without compromising the accuracy. Second, it doesn't explore the *selection of diverse NNs*. As shown in Fig. 4, NNs have diverse characteristics over different object sizes. Intuitively, for large objects in sparse scenes, a small and cheap NN should be used to reduce the latency, while for small objects in crowded scenes, a large and expensive NN is desired to ensure high accuracy. Fundamentally, existing uniform partitioning treats every pixel equally by spending the same amount of computing power on each pixel, which prevents it from achieving a better accuracy-latency trade-off.

Taking advantage of the above opportunities faces several key challenges. First, it is non-trivial to decide an ideal partitioning plan to best balance the trade-off between accuracy and latency considering multiple factors including partition size, object density and size, and NN characteristics. Second, we need to consider short-term content dynamics over long-term object distribution to avoid missing objects caused by ignored partitions. Third, the system must be efficient and lightweight to run on resource-limited edge devices and be flexible to support diverse latency budgets.

To address the challenges, we design Remix to achieve efficient and flexible high-resolution object detection with tunable latency. The key techniques in Remix are *adaptive partitioning* that generates partition plans to maximize the accuracy within a latency budget through non-uniform partitioning; and *selective execution* that deals with short-term content dynamics and runs a partition plan efficiently. Next, we describe how Remix system works in detail.

3 REMIX: SYSTEM DESIGN

Remix aims to run diverse NNs with different execution priorities in different parts of a high-resolution image, so as to maximize the overall detection accuracy within resource and latency constraints. To do so, it solves two critical problems, 1) how to effectively partition the input image, and 2) how to efficiently process each partition. Fig. 5 depicts the design of Remix, where we address these two problems with an adaptive partition module (§3.1) and a selective execution module (§3.2).

The adaptive partition module is designed to obtain a set of effective partition plans. The inputs of this module include a set of candidate NNs, historical frames, and a latency budget set by the user. Remix first obtains the characteristics of the candidate NNs in terms of latency and detection accuracy on objects of different sizes through profiling (§3.1.1). Historical frames are used to extract the object distribution in the input (§3.1.2). Based on the network characteristics and the object distribution, we can enumerate possible partition plans and quickly estimate their detection accuracy and inference latency, without actually executing them (§3.1.3). Finally, Remix builds a partition plan pool by selecting plans with estimated latency close to the budget while achieving high estimated accuracy (§3.1.4).

Partition plan pool is fed into the selective execution module. Initially the selective execution module picks the partition plan which has the best accuracy within the latency budget. According to the selected plan, the high-resolution input is split into blocks, and each block is assigned to a network for inference. Instead of processing all blocks, a subset of them might be skipped if there exists no object. We leverage the previous detection results as the feedback to dynamically determine which blocks can be skipped in the current frame (§3.2.1). Note that skipping leads to a latency strictly lower than the one set by user. As such, Remix adjusts the partition strategy on-the-fly by switching to another plan in the plan pool every a few frames, until the utilization is maximized (§3.2.3). Finally, per-block detection results are merged to produce the whole-image prediction (§3.2.4).

3.1 Adaptive Partition

Given a set of available networks N, the historical frames H and the latency budget T, the goal of this module is to find the partition plan κ, which maximizes the detection accuracy within T, based on the understanding of networks' characteristics and object distribution.

3.1.1 Neural network profiling.
We firstly profile the characteristics of available networks, more specifically, their latency-accuracy trade-off on different sizes of objects. The profiling process is a one-time effort and we can run it in the offline phase.

Profiling networks' inference latency is straightforward. The latency is almost consistent between runs. Thus, we execute the networks on the target device multiple times using different batch sizes. For each network $n \in N$, we obtain its averaged latency L_n^b where b is the batch size.

In terms of accuracy, we evaluate NNs on objects with different sizes defined in Table 2. For each network $n \in N$, we obtain a capability vector AP_n,

$$AP_n = \langle \tau_{S0}, \tau_{S1}, \cdots \tau_{M0}, \tau_{M1}, \cdots \tau_{L2}, \tau_{L3} \rangle, \tag{1}$$

where τ is the detection accuracy at a particular object size level.

Note that AP_n can be obtained using existing public datasets rather than the real data captured by the targeted device. We use MS-COCO dataset [6] in our implementation. Although the obtained AP_n may differ from real data, the relative performance differences between networks are usually consistent (more details are discussed in §4.3.2). We only leverage these relative differences to guide REMIX to choose proper networks.

3.1.2 Object distribution extraction.
In common object detection applications, *e.g.,* smart surveillance, cameras are usually stationary in the fixed angle and position. Thus, we have two observations: 1) for one category of objects, their visible sizes are often similar over time when they are on close positions of the captured view; 2) common objects, *e.g.,* pedestrians and vehicles, trend to appear in certain regions of the captured view. Based on these two observations, we leverage historical frames to learn the distribution of object sizes. Given the captured view V from the targeted camera, we calculate the distribution vector F_V,

$$F_V = \langle \phi_{S0}, \phi_{S1}, \cdots \phi_{M0}, \phi_{M1}, \cdots \phi_{L2}, \phi_{L3} \rangle, \tag{2}$$

where ϕ is the distribution probability of objects on each size level described in Table 2.

For any divided block p in V, we can also get its distribution vector F_p, by counting corresponding objects whose bounding boxes centroids are in p.

Ideally, the distribution vector F_V should be obtained from the ground truth of historical frames H. However, it is not practical to manually label the large amount of frames. Therefore, we use an oracle model to do the labeling, which is a widely-used approach in existing work [86, 87]. The oracle model should be as accurate as possible. We use a modified UP-D7 as the oracle model as detailed in §4.1.1, and our evaluation shows that the end-to-end performance using the ground truth and using the detection results of the oracle model are very close (§4.3.1).

3.1.3 Performance estimation.
In order to find a proper network for a certain part, we need to compare the performance of available NNs. Obviously, it is too time-consuming to execute every network on every possible part. Therefore, we propose to quickly estimate the performance without actually executing the particular network, based on the outputs of techniques above.

Given a network $n \in N$ and a block $p \in V$, the detection accuracy of n on p mainly depends on two factors: 1) the capability of n, specifically its detection accuracy on different object sizes; 2) the object distribution in p, specifically the density of objects in p and their visible sizes.

Algorithm 1: Adaptive Partition Planning

```
function adaptive_partition($V, H, N, AP_N$)
    Input :
        • $V$ : The coordinates of processed view.
        • $H$ : The historical frames.
        • $N$ : The set of candidate networks.
        • $AP_N$ : The capacity vectors of $N$.
    Output:
        • $K$ : The obtained partition plans.
    foreach network $n \in N$ do
        $L_n^1 \leftarrow$ the inference latency of $n$ with 1 as the batch size;
        $\gamma \leftarrow$ scale ratio of the input size of $n$ to the resolution of $V$ ;
        $H_\gamma \leftarrow$ scale down objects in $H$ according to $\gamma$ ;
        $f \leftarrow$ calculate the distribution vector according to $H_\gamma$ ;
        $eAP_v^n \leftarrow$ estimate_AP($f, AP_n$) ;
        $\kappa \leftarrow \langle$V$, n\rangle, eAP_v^n, L_n^1$ ;
        insert $\kappa$ into $K$ ;
        prune redundant plans in $K$ ;
    foreach network $n \in N$ do
        $P \leftarrow$ uniformly partition $V$ by the size of $n$ ;
        Initialize $tmp$ ;
        foreach divided block $p \in P$ do
            $N_s \leftarrow$ networks which are smaller than $n$ ;
            $H_p \leftarrow$ objects located at $p$ ;
            $SK_p \leftarrow$ adaptive_partition($p, H_p, N_s, AP_{N_s}$) ;
            permute and combine $SK_p$ with $tmp$ ;
            prune redundant plans in $tmp$ ;
        insert $tmp$ into $K$ ;
        prune redundant plans in $K$ ;
    return $K$ ;
```

As discussed above, we can use the capability vector AP_n to represent the capability of n, and use the distribution vector F_p to describe the object distribution in p. Therefore, we can quickly estimate the detection accuracy eAP by AP_n and F_p,

$$eAP_p^n = AP_n \cdot F_p, \tag{3}$$

According to a partition plan κ, the captured view is divided into several blocks p. Now we can estimate the overall detection accuracy applying κ on the whole view V, eAP_V^κ by combining all the estimations of every p,

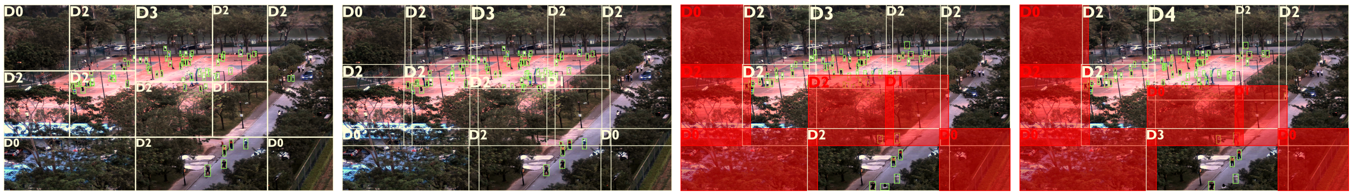
$$eAP_V^\kappa = \sum eAP_p^n \cdot \lambda_p, \quad p \in V, \tag{4}$$

where λ_p is the object density of p relative to the whole view V.

Estimating the inference latency of a partition plan is straightforward. We sum the latency of all the divided blocks p. Some blocks might be assigned by the same network, then we can execute them in batch. Thus, we estimate the inference latency $eLat_\kappa$,

$$eLat_\kappa = \sum L_n^b, \quad n \in N_\kappa, \tag{5}$$

where N_κ is the set of networks used in κ. The proposed performance estimation approach is robust. We evaluate the approach on the PANDA 4K dataset to compare our estimations with the actual measured mAP and latency of different partition plans. The results show the strong correlation (>0.6) and little bias (<5%) for estimated detection accuracy and latency, respectively. More evaluation details would be presented in §4.3.2.



(a) Adaptive partition. **(b) Partition padding.** **(c) Partition selection.** **(d) Plan control.**

Figure 6: Example of partition plans obtained by REMIX on PANDA 4K dataset.

3.1.4 Partition planning.

The goal of partition planning to find a set of candidate partition plans K with high eAP and $eLat$ close to the latency budget T. A partition plan κ that is actually used to process the input is dynamically chosen from K at runtime.

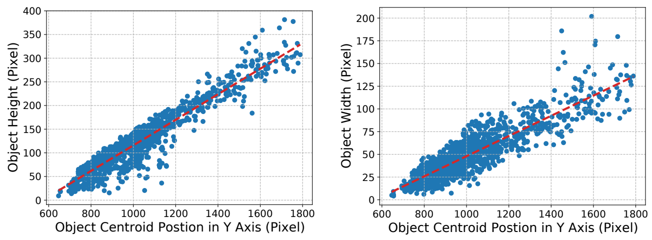
We apply a dynamic programming based algorithm to find such K. Intuitively, the algorithm enumerates every possible partition plan and estimates their detection accuracy and latency. Algorithm 1 shows the pseudo code.

For the capture view of a camera, the algorithm takes its historical frames H and capability vector AP_n of candidate network $n \in N$ as the inputs. The algorithm counts all the ways to process a given frame. Firstly, a frame can be down-scaled and processed by a network directly. Thus, for each network $n \in N$, we calculate the scale ratio between the size of captured view and the required resolution of n (line 4). Next, we scale objects in H accordingly and update the object distribution vector (line 5-6). Then we can estimate $eLat$ and eAP (line 7-8) using Equation 5 and 3, respectively.

A frame can also be split. For each network $n \in N$, we can use it divide the frame uniformly (line 10-11). Every divided block in turn can be processed by this algorithm recursively (line 14-17). The partition plans obtained from the divided blocks p are notated sub-partition plans SK_p (line 17). We permute and combine SK_p of all divided blocks (line 18), and estimate their $eLat$ and eAP, accordingly. The recursive algorithm would stop until the input block cannot be divided any more, *i.e.,* the size of the input block is equal or smaller than the smallest network's size in N.

Enumerating all possible partition plans is time-consuming due to the huge search space. To decrease the search space, we additionally adopt a prune-and-search [60] based method. During the planning, if multiple plans in SK_p have the close $eLat$ within 1 ms, only the one with highest eAP is kept (line 10, 19, 21). We also set a cut-off latency threshold, any partition plan would be dropped if its $eLat$ is higher than this threshold. Empirically we set this threshold to 10 seconds. In this way, the search space can be largely decreased. For instance, we use seven candidate networks, EfficientDet D0-D6 to process one 4K video. The search space is decreased from 4.3×10^{38} to around $9,400$ different partition plans with different latency-accuracy trade-offs.

Among these plans, we first choose the most accurate plan κ_0 within the latency budget T as a candidate plan. However, κ_0's latency might be an over-estimation, since the blocks in κ_0 will be selectively executed at runtime, which would be discussed in §3.2. Thus, we additionally choose several candidate plans with higher $eLat$ that can potentially better utilize the latency budget. In particular, we choose partition plans with $eLat$ ranged from T to



(a) Heights and positions. **(b) Widths and positions.**

Figure 7: Object sizes and object centroid positions in Y axis of the captured view.

$1.5T$. An example partition plan is illustrated in Fig. 6(a). Intuitively we see that for the blocks containing many pedestrians, medium networks like EfficientDet-D3 are used. While lightweight D0 is assigned to the blocks with few or large objects that are easy to detect.

3.1.5 Partition padding.

One problem of partitioning high resolution inputs is that it might cut apart the objects around the boundary, leading to detection failures. To cope with it, a straightforward method is to add overlaps between divided blocks, which introduces the overhead of redundant partitions and higher latency. To minimize the overhead, we propose to adaptively add minimal margins to each divided block. The minimal vertical and horizontal margin sizes can be determined by the potential object's height and width, which happens to locate at the boarder of this block.

Based on the perspective effect, for one category of objects, its visible height and width in pixel is linearly related to its position on vertical axis. Therefore, we can apply the linear regression to predict an object's height and width using its position. Fig. 7 shows an example of such the linear relationship where pedestrians are detected in the PANDA 4K dataset. We leverage the historical detection results to obtain such linear relationships. Based on that, we can add the minimal necessary margin to each divided block. Fig. 6(b) demonstrates a padded partition plan. Every block in Fig. 6(a) is padded by a margin to include the potential pedestrians nearby successfully. Since the margins are often a few pixels, we use the same network to process the padded blocks, which introduces almost zero overhead.

3.1.6 Bootstrap and update.

In order to obtain effective partition plans, the networks' characteristics and the object distributions should be profiled. Thus, REMIX needs a bootstrap stage when deploying. The NNs profiling is an one-time process. To obtain the object distribution, several frames are required. Our evaluation

(§4.3.1) shows that only a very little amount of frames are enough to generate qualified partition plans.

We also notice that object distributions might vary over time. Therefore, in Remix we update the object distribution and re-generate partition plans accordingly. The adaptive partition module is lightweight. It can run on the cloud or on edge devices directly. According to our measurements, the whole update process only costs around 6 minutes in total, to analyze enough historical frames (< 4 minutes) and generate partition plans (< 3 minutes) on a Jetson edge device. Currently we do not detect the semantic changes of scenes and automatically trigger the update, which is handled by users empirically. We would take the automatic update as future work.

3.2 Selective Execution

The generated partition plan pool is then used in the selective execution module. To further deal with the video dynamics at runtime, this module has the following two objectives. 1) Given a sequence of video frame I from the captured view V and the chosen partition plan κ, the module determines which blocks p should be executed, to further speedup the inference. 2) Given multiple partition plans K, it dynamically chooses the proper one to make the execution latency L as close as possible to the latency budget T, to further improve the detection accuracy. Accordingly, we propose the following techniques.

3.2.1 Partition selection. Initially, we select the partition plan κ with the highest eAP_κ while its $eLat_\kappa$ is within the latency budget T. According to κ, the high-resolution frame is divided into blocks. At runtime, some blocks may not contain any target object, *e.g.,* they are covered by irrelevant background or objects may just disappear in them temporarily. Compute resources can be saved by skipping such blocks.

To determine which blocks should be executed, we leverage the previous detection results as the feedback, and deploy a Exploration-and-Exploitation [68] strategy. In order to balance between exploiting and exploring, we make Remix work in this manner conservatively. 1) If no objects can be detected in a block for several inferences, we would skip it in following few frames, to save the latency. 2) If objects are detected in a block, then this block would be processed in the next frame, to ensure the detection accuracy. 3) Objects may still appear sometimes in skipped blocks, so we also try to bring such blocks back, to explore more potential detection.

To achieve this, we introduce a neat yet efficient additive-increase multiplicative-decrease (AIMD) solution, inspired by TCP congestion control algorithm [16]. In particular, for each divided block p we assign a *penalty* window w_p. The value of w_p represents that the block p would be skipped for the following certain inferences. Initially w_p is set as 0, hence every block would be executed. Once the inference is finished, we update w_p by the detection results. 1) If we cannot detect any object in p, we linearly increase w_p according to $w_p = \iota_p - 1$, where ι_p is the consecutive inference times when no object is detected in p. 2) Once objects can be detected in p, instead of multiplicative decrease w_p, we conservatively reset w_p to 0 as well as ι, to ensure p be processed in the next inference. 3) If the block p is skipped, we would decrease its w_p by 1 for every skipped inference. Once its w_p is back to 0, we bring it another opportunity to infer if objects appear.

Fig. 6(c) demonstrates an example of the partition selection output in run time. The blocks with red masks are skipped. Since we apply the conservative strategy above, Remix skips blocks that contain no object, leading to significant latency speedup with minimal accuracy loss.

3.2.2 Batch execution. Next the chosen blocks would be processed by the assigned network. There might be multiple blocks assigned by the same network, and thus we have the opportunity to execute them in batch, which could speed up the inference further.

3.2.3 Plan control. Since several divided blocks might be skipped, the actual inference latency L may be lower than the latency budget T. It suggests that the system still has the compute power left. Therefore, we ask if we can fully utilize the latency budget to further improve the detection accuracy.

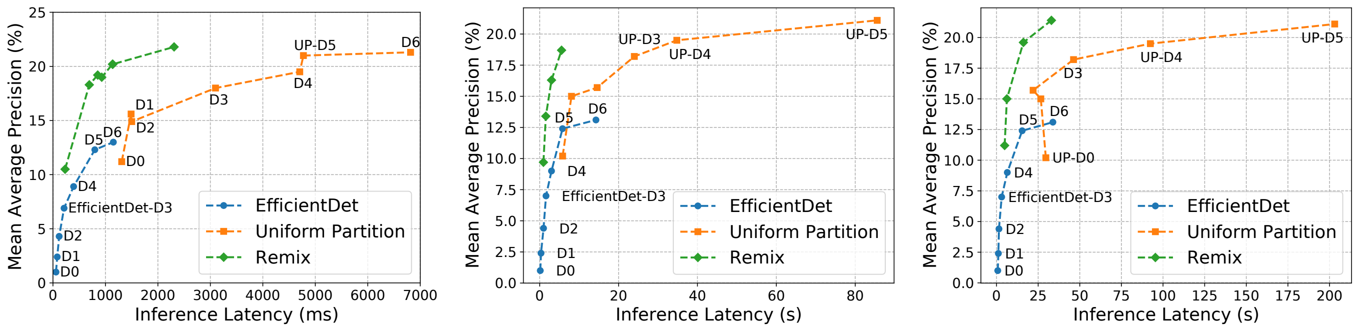
To this end, in the adaptive partition module (§3.1), we not only pick the most accurate partition plan within T, but some other plans beyond T as well. These plans are heavier, with higher latency as well as higher expected detection accuracy. We can re-select another plan from them, letting the actual latency L approximate to T. However, it is not easy to select such another plan because it depends on the video dynamics which blocks are skipped. Thus, the actual latency L is usually time-varying and indeterminate.

To address this issue, we are inspired by the closed-loop control system. Instead of determining the updated plan at once, we use the actual latency L as the feedback and continuously try different plans until its L approximates T. More specifically, we employ a closed-loop controller. The desired setpoint (SP) of the controller is set to the latency budget T. The measured process variable (PV) is L as the feedback. Using SP and PV, the controller would output a control value u. In our case, it is the updated budget used to search plans from the partition plan pool. The most accurate plan within u would be selected. In Remix we implement a proportional–integral-derivative (PID) controller [20]. In each time step t, the PID controller calculates an error value $e(t)$ as the differences between SP and PV, and applies the correction based on proportional, integral, and derivative terms as $u(t)$. Each term is associated with coefficient, K_p, K_i and K_d for the proportional, integral, and derivative terms, respectively.

We tune parameters manually in offline. Generally, we expect the controller can make Remix quickly reach SP, meanwhile be more sensitive and responsive to the disturbances. Therefore, we set relatively higher K_p and K_i. We notice there are several sophisticated tuning methods [18]. We regard the PID parameters tuning as future work.

Fig. 6(d) illustrates an example of the re-selected partition plan. Compared with Fig. 6(c), the blocks containing dense objects are given more compute power by assigning larger networks. Meanwhile its actual latency is closer to the latency budget.

3.2.4 Bounding box merging. As the divided blocks are padded, some objects might be detected repeatedly. When merging detection results from each block, we apply the non-maximum suppression (NMS) algorithm [61] to filter out the duplicated bounding boxes. The threshold of NMS in Remix is set to 0.5.



(a) Jetson AGX Xavier. **(b) Snapdragon 855.** **(c) Kirin 970.**

Figure 8: mAP and mobile GPU inference latency of REMIX on the PANDA 4K dataset with different hardware.

4 EVALUATION

We evaluate the performance of REMIX in terms of the inference latency and the achieved detection accuracy, *i.e.,* mAP with different hardware and under different real scenes. We further take ablation studies to examine the performance of individual units.

4.1 Experimental Setup

4.1.1 Implementation. We implement REMIX on three hardware platforms with different capacities, a Jetson AGX Xavier [7] and two additional popular mobile SoCs, Snapdragon 855 [3] and Kirin 970 [2]. Jetson is equipped with one 512-core Nvidia Volta GPU. On Snapdragon and Kirin, one Adreno 640 and a Mali G72 mobile GPU are equipped, respectively. We leverage such mobile GPUs for the inference. For Jetson, REMIX uses TensorFlow [14] as the inference engine, while TensorFlow Lite (TFLite) [15] [2] is used for Snapdragon and Kirin. We use EfficientDet [70] series as basic network units for partition planning and inference. Specifically, we use EfficientDet-D0 to D6 on Jetson, and only use EfficientDet D0 to D5 considering the limited capabilities of selected mobile SoCs [3]. We use the uniform partition by Efficient-D7 (UP-D7) as the oracle model and set overlap ratio to 50% to avoid objects being cut off during partitioning.

4.1.2 Dataset. We evaluate REMIX on the videos of the PANDA 4K dataset (§2). For each scene in the videos, we use the beginning 10% frames as the historical frames to generate partition plans, and use the remaining 90% for testing.

4.1.3 Baselines. Our baselines include two sets of networks, 1) SOTA EfficientDet and its variants (D0-D7), 2) EfficientDet D0 to D6 with uniform partition, denoted as UP-D0 to UP-D6. To the best of our knowledge, EfficientDet achieves the best detection accuracy on public dataset [6]. We scale down 4K frames to the required input size of the network before feeding them to each individual NN. Uniform partition has also been discussed in several recent studies for detecting objects on high-resolution videos [65, 85].

4.1.4 Latency budget. We design REMIX to achieve the best accuracy within a given latency constraint. Different latency budgets set for REMIX lead to varied performance. To fairly compare against baselines, we adopt multiple latency budgets to show the performance of REMIX. In our evaluation, we select seven latency budgets, which are same with the inference latency of selected baselines, including D4 to D6 and UP-D0 to UP-D3. The selected latency budgets are ranged from 400ms to 3,000ms. We donate them as T_{D4} to T_{D6} and T_{UP0} to T_{UP3}, respectively.

4.2 Remix Performance

4.2.1 Overall performance. We firstly show the overall performance of REMIX. Fig. 8 illustrates the performance of REMIX as well as the baselines. Each point on the curves denotes the achieved averaged mAP with the averaged inference latency for frames of all scenes.

REMIX successfully distinguishes different computation resource requirements for different regions, and assigns with proper networks. Therefore, it can achieve higher detection accuracy. As shown in Fig. 8(a), on Jetson, under the similar latency, REMIX can achieve relative 18%-70% detection accuracy improvements relatively compared with the baselines, around 49% on average. For instance, Efficient-D6 obtains 13% mAP with the latency of about 1,150ms. REMIX is able to achieve 20.2% mAP with similar latency (1,130ms).

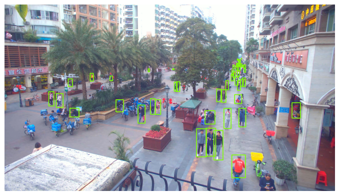
Compared with Jetson, the selected mobile SoCs are more resource-limited. The large networks such as EfficientDet-D5 show fairly poor performance. However, with the proposed techniques, REMIX demonstrates significant performance improvements. In particular, shown in Fig. 8(b), on Snapdragon 855, with the similar latency, REMIX can achieve 85% detection accuracy improvements relatively on average, compared with the baselines. The averaged detection accuracy improvement is about 62% relatively on Kirin as illustrated in Fig. 8(c).

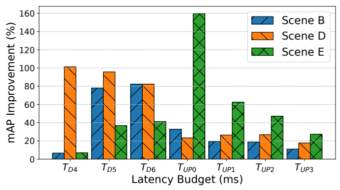
Meanwhile, due to the adaptive partition and selective execution module, REMIX can achieve competitive mAP with much lower inference latency. As shown in Fig. 8(a), on Jetson when obtaining similar mAPs (within 0.2%), REMIX can achieve up to 5.5 × speedup compared with baseline approaches. For instance, UP-D4 achieves approximately 19.5% mAP with a latency of 4700ms. REMIX, on the other hand, is able to obtain 19.3% mAP at a much smaller latency cost of 850ms. The inference speedups are up to 8.1× and 5.1× on Snapdragon and Kirin, shown in Fig. 8(b) and 8(c), respectively.

[2]The inference batch size in TFLite is fixed to 1.
[3]Note that some operators in EfficientDet are not supported by the mobile GPU we use. As a result, they fall back onto CPU.

(a) Scene B: Plaza square.

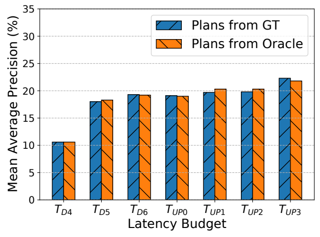
(b) Scene D: Sports ground.


(c) Scene E: Street.

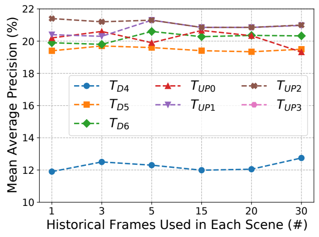


(d) mAP improvements.

Figure 9: Typical scenes in PANDA and corresponding mAP improvements achieved by REMIX.

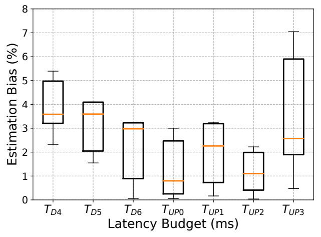


(a) Performance of plans generated from the ground truth and oracle model.

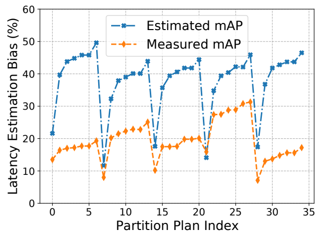


(b) Performance of plans generated using different amounts of historical data.

Figure 10: End-to-end performance of different partition plans under various typical latency budgets.



(a) Bias between the measured and estimated latency on different latency budgets.



(b) Correlation between the measured and estimated mAP of different plans.

Figure 11: Bias and correlation between the measured and estimated latency and mAP.

4.2.2 Performance across different scenes.

We also examine the performance improvements achieved by REMIX across different scenes under various latency budgets. Fig. 9 showcases three typical scenes in the PANDA 4K dataset and their corresponding mAP improvements. We notice that if the scene doesn't have clear spatial structures, the mAP improvement is limited. For instance, Scene B is a plaza square where objects distribute almost uniformly. The average relative mAP improvement is only about 31% compared with the baselines. Conversely, the skewed distribution of object in Scene D and E helps REMIX skip many blocks containing very few objects and distribute more compute power on more important blocks, thus leading to a higher mAP. As shown in Fig. 9(b) and 9(c), the mAP improves by 53% and 52% on average compared with the baselines, respectively.

4.3 Evaluation of Adaptive Partition

Next, we breakdown the system and evaluate each key technique with Jetson, starting from the adaptive partition module.

4.3.1 Quality of partition plans.

We propose to use an oracle model to replace the manual labeled ground truth, to extract the object distribution from the historical frames (§3.1.2). Then based on that, we generate the partition plan pool. We evaluate the oracle model on the PANDA 4K dataset. It achieves about 29% mAP with the latency of nearly 12 seconds per frame.

However, a more critical question is how end-to-end performance differs between using partition plans generated from the labeled ground truth and from detections of the oracle model. Fig. 10(a) answers this question. Leveraging the proposed oracle model, REMIX can achieve a very close detection accuracy to that of using the ground truth. The differences between them are only round 0.1% to 0.7% under various latency budgets. Therefore, we are confident to use the oracle model to extract the object distribution and obtain the partition plan pool base on that.

Another interesting question would be how many history frames REMIX needs to generate qualified partition plans. Thus, we use different volumes of historical frames to search partition plans and evaluate corresponding performance under different latency budgets.

We select three scenes. For each scene we use the first *1, 3, 5, 10, 20* and *30* frames to generate partition plans. Fig. 10(b) illustrates the averaged detection accuracy of scenes under different latency budgets. Overall all the plan pools obtain the similar mAP. It also indicates that a very little amount of historical frames are sufficient enough to generate qualified plan pools. An interesting observation is that even we use only one frame to search plans, we also can get competitive performance. We conclude this as that the object distribution of the sampled historical data might be often consistent during the capturing. But we also notice that the object distribution might be changed, due to external factors, *e.g.,* time and uncertain events. Thus, how to efficiently sample the typical frames from the historical data would be future work.

4.3.2 Performance estimation bias.

In order to perform the partition planning without actually executing a network, we introduce the performance estimation module (§3.1.3). To verify our estimations, we compare the estimated latency and mAP with the measured values after the executions.

Fig. 11(a) shows the bias between the estimated latency with the measured latency when using different latency budgets. All median bias is less than 5%. We also evaluate the estimated detection accuracy *eAP*. We pick different partition plans and compare their *eAP* and the measured mAP. Fig. 11(b) shows the result, which

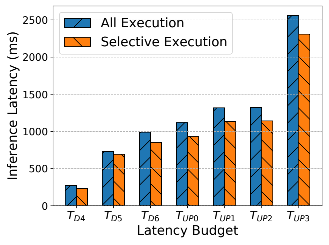


Figure 12: Latency with and without selective execution using different budgets.

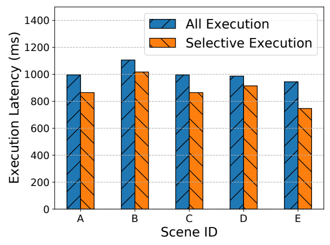


Figure 13: Latency with and without selective execution in different scenes.

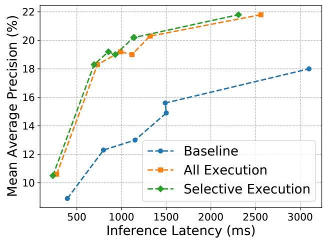


Figure 14: mAP with and without selective execution on PANDA 4K dataset.

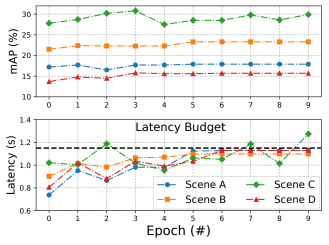


Figure 15: mAP improvements with latency controller across scenes.

Latency Budget	T_{D4}	T_{D5}	T_{D6}	T_{UP0}	T_{UP1}	T_{UP2}	T_{UP3}
Without padding (%)	10.4	17.4	17.8	17.1	19.0	19.1	20.3
With padding (%)	10.6	18.3	19.2	19.0	20.3	20.3	21.8

Table 3: mAP with and without partition padding.

illustrates the correlation between them. The correlation coefficient is about 0.6. We see there always exists a bias between the estimated and measured mAP, since the performance vector AP_n (§3.1.1) used to perform estimations, is obtained from another public dataset. We argue that we can ignore such the bias because we only focus the relative difference among plans to pick the proper one.

4.3.3 Gains from partition padding. We add margins to each divided block to avoid objects being cut-off (§3.1.5). To validate the benefits of the partition padding, we compare the achieved mAP of plans with and without the padding, using different latency budgets in Remix. Table 3 shows the details. We observe that the padded plans consistently outperform the plans without padding, leading to up to 11.3% relative mAP improvement and more 1.2% absolute mAP on average. Meanwhile since we only pad necessary margins adaptively. The margins are so tiny that there is almost no extra overhead.

4.4 Evaluation of Selective Execution

Next, we evaluate each key techniques in the selective execution module with Jetson. This module is introduced to further accelerate the inference by skipping several divided blocks. Thus, we present the saved latency as well as the impacts on the detection accuracy.

4.4.1 Speedup from batch execution. Since we partition high-resolution inputs, Remix naturally has the opportunity to execute blocks assigned with the same network in a batch (§3.2.2). We measure the latency with and without batching. The results show that the batching execution can bring 1.11× to 1.32× speedup across different partition plans, meanwhile it has no impacts on the achieved mAP.

4.4.2 Speedup from partition selection. The selective execution module is designed to skip the blocks containing no object in the run time. To validate its effectiveness, we measure the latency reduction with and without the partition selection technique. Fig. 12 demonstrates the comparison under various latency budgets. With the selective execution module, Remix can achieve about 1.2× to 1.7× speedup.

We also observe that the speedup varies across different scenes. Fig. 13 shows such the variances where we set the latency budget to T_{D6}. The selective execution module can bring the higher latency reduction in the scene that has the clear spatial locality, otherwise the gain is limited. For example, the object distribution of Scene D is significantly well-regulated shown in Fig. 9(b), where the latency reduction is about 14%. Alternatively, the reduction for Scene B, shown in Fig. 9(a), is only about 4%.

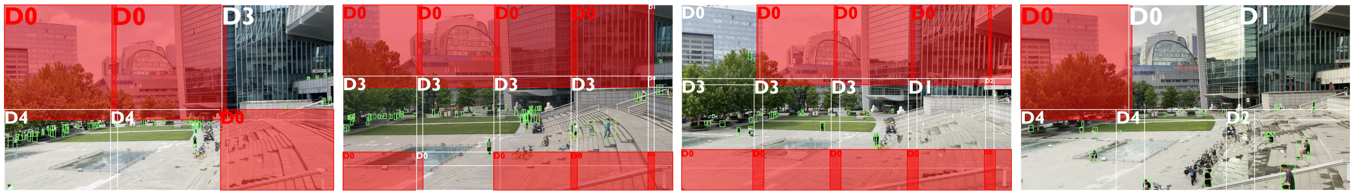
Since some blocks might not be processed, we care about the impacts on the detection accuracy. Fig. 14 demonstrates the mAP loss when applying the selective execution. The mAP loss is less than 0.1% on average. Since we use the very conservative strategy to select blocks, only those blocks containing no object are skipped in most cases. We think such an impact on mAP is negligible.

4.4.3 Gains from plan controller. The saved latency from the selective execution module can be utilized to explore more aggressive partition plans to achieve better detection accuracy (§3.2.3). We test the plan controller if it can help fully utilize the latency budget and achieve the higher mAP. We set the latency budget to T_{D6}. We select 4 scenes and run 10 epochs for each scene. In each epoch we process 240 frames, then the latency controller would re-select the partition plan for the next epoch. Fig. 15 shows the result. After 10 epochs, the averaged utilization of the latency budget improves from 75% to 94%, meanwhile the average mAP across scenes get about 11% relative improvement.

4.5 Case Study

Besides the public dataset PANDA, we also deploy Remix with Jetson in real world, and evaluate 4K videos collected by ourselves from an enterprise campus. We capture multiple videos from the same scene but at different times, *i.e.,* morning, noon and afternoon. Thus, the distribution and density of people slightly differ across video clips. Fig. 16 demonstrates sample frames from the collected videos at 10:00 AM, 13:00 PM and 16:00 PM and 19:00 PM respectively. In total we have nearly 12, 000 frames for the evaluation. We use EfficientDet D0-D5 as the basic network units. We set the latency budget to 1,000ms. For each video clip, we use the first 10% frames to obtain partition plans. Since we don't have labels for these videos, we use the oracle model to generate pseudo-labels for remaining 90% frames as reference.

4.5.1 Detection performance. Compared with the oracle model, overall Remix can achieve 50.2% average mAP on frames. Due to



(a) 10:00 AM. (b) 13:00 PM. (c) 16:00 PM. (d) 19:00 PM.

Figure 16: Sample frames collected from an enterprise campus at different times.

the plan control, Remix would continuously improve mAP by fully utilizing the latency budget. The peak mAP achieved is up to 66%. We also include AP50 [6], which is commonly used in industry and a less stringent metric compared with mAP. The AP50 achieved by Remix is around 76.3% on our own dataset.

Evaluation At	Update At			
	10:00 AM	13:00 PM	16:00 PM	19:00 PM
10:00 AM	**72.8%**	-	-	-
13:00 PM	67.7%	**82.4%**	-	-
16:00 PM	63.1%	63.5%	**71.7%**	-
19:00 PM	78.4%	77.1%	78.3%	**78.5%**

Table 4: AP50 achieved by Remix when applying the partition plans generated at different times.

4.5.2 Performance adaptation.

As discussed in 3.1.6, the object distribution might change over time. Remix adapts to such changes by updating and re-generating the partition plans. We evaluate the achieved detection accuracy at different times by using updated partition plans. We also measure the performance by reusing the previous partition plans. Table 4 details the results.

With the update process, Remix successfully adapts to the scene changes, achieving the highest mAP at each period of time, *i.e.,* 72.8%, 82.4%, 71.7% and 78.5%, respectively. When reusing the partition plans generated previously, the performance is degraded. For example, the AP50 is reduced by up to 14.7% if we apply the partition plans generated at 10:00 AM on the video clip at 13:00 PM. However, we also notice some partition plans might also work well for videos at other time. For instance, when reusing the partition plans generated at 10:00 AM onto the video clip at 19:00 PM, 78.4% can still be obtained, only 0.1% AP50 is reduced compared with using updated partition plans. The observation also indicates that certain scenes might show periodic patterns over time, in terms of object distribution.

4.5.3 Update overhead.

Updating and re-generating partition plans would introduce another overhead. Remix allows the update process run either on the cloud or on the target edge devices. According to our measurements, with Jetson it costs around 4 minutes to analyze enough historical frames (20 frames), search and generate optimized partition plans in 3 minutes. When offloading to the cloud, the historical frames are compressed and transmitted in 2 minutes under a 4G cellular network. The analyzing and plan generation spend around 4 minutes in total on the server with an Intel Xeon-E5-2690 CPU.

4.5.4 Additional inference overhead.

To enable inference on Remix, the system also introduces additional overheads. Generally, Remix introduces four phases to finish the inference procedure in each frame and we measure them according in our case study. 1) Pre-processing, 70ms on CPU. Once the input frame is ready [4], Remix firstly partitions the frame to non-uniform blocks and then resizes them to match the required size of assigned networks. 2) Selection, 0.1ms on CPU. Remix determines which blocks should be processed based on the feedback from previous detection results. 3) Inference, 960ms on GPU. Remix infers blocks by feeding them into networks. 4) Post-processing, 32ms on CPU. Remix merges the detection results from each block.

We notice that the pre-processing dominates the majority of inference overhead. Thus, we analyze this issue in-depth. We find that the partitioning process is relatively lightweight, which only costs around 12ms. However, the resizing process is painful, spending nearly 57ms. The resizing process also varies across different partition plans, *e.g.,* more blocks a partition plan contains, more overhead its pre-processing brings.

Fig. 17 demonstrates the pre-processing overhead distribution of the generated partition plans in our case study. More than 80% of partition plans have the pre-processing overhead less than 130ms. For the picked plans according to the latency budget (1,000ms), the averaged pre-processing overhead is 112ms. What's more, due to the selection execution, several blocks would be skipped and Remix would not resize such blocks. Therefore, the actual pre-processing overhead we measured is about 70ms on average in our case study.

The additional inference overhead can also be hidden. We make use of the multiple heterogeneous computation units on edge devices. According to the workflow of Remix, the pre-processing and post-processing are executed on the CPU and the inference is on the GPU. Therefore, we can pipeline these phases, *e.g.,* starting pre-processing the next frame on the CPU when inferring the current frame on the GPU simultaneously. In this way, the most of pre-processing overhead would be largely hidden.

4.5.5 Memory footprint.

Remix processes each frame using multiple NNs in a mixed way. Thus, we need load and initialize all the candidate NNs, *i.e.,* from D0 to D5, into the memory beforehand. According to our measurements, Remix consumes at maximal 9.0 GB memory during inference on Jetson. The majorly of memory is used by large networks, *i.e.,* the weights and intermediate tensors of EfficientDet-D5 consumes nearly 6.4 GB memory. Currently we

[4]The I/O latency, *i.e.,* loading images from the disk or streaming frames from the camera, is excluded in the pre-processing overhead.

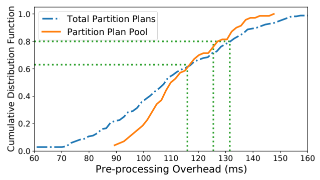


Figure 17: Distribution of pre-processing overhead using various partition plans.

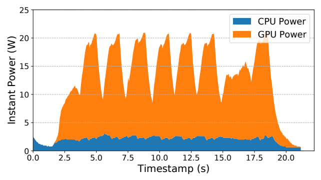


Figure 18: Energy consumption trace when processing 8 frames with Jetson.

don't consider only loading part of networks, we leave the exploration of dynamic NN swapping as future work to optimize the memory footprint.

4.5.6 Energy consumption. We also measure the energy consumption of REMIX. We use the built-in INA3221 power monitor [13] with Jetson to collect the instant energy consumption traces. The sampling rate is $20Hz$. Fig. 18 illustrates a typical trace. Overall the averaged energy consumption per frame is about $12.7W$ in total. The GPU consumes the majority power, $10.6W$ on average specifically. The peak of energy consumption is nearly $21.3W$.

5 RELATED WORK

Object Detection on Edge Devices. Object detection is well studied in computer vision with rich applications, *e.g.,* video surveillance, activity recognition, and object tracking. Today, the SOTA object detectors are based on convolutional neural networks (CNNs) [46] due to their powerful feature extraction ability. Existing object detectors can be classified into two categories, including two-stage detectors and one-stage detectors. The two-stage detectors, *e.g.,* Faster R-CNN [64] and Mask R-CNN [36], reach the highest accuracy, but are typically slower since they typically require many inference steps for each image. Instead, the single-stage detectors such as YOLO [62] and SSD [53] can significantly reduce the latency although the accuracy is much lower. Building upon single frame detectors, video object detection focus on incorporating contextual properties to improve the detection accuracy and speed [19, 52], by linking per-frame detection together [35, 43], extracting and propagating intermediate features across frames via optical flow [88, 89], and designing associated network modules [23], *e.g.,* 3D convolutions [58, 76] or recurrent networks [38, 51].

To push object detection abilities to resource-constrained devices, a typical approach is to adapt detectors optimized for edge [22, 25, 28, 29, 53, 66]. We use the SOTA EfficientDet [69, 70] series in REMIX, they can also be replaced to any other set of NNs. Although much effort has been put into optimizing NNs for edge, existing solutions [50, 65, 85] are not designed to deal with high-resolution inputs on device. Instead of most on-device object detection approaches that attempts to design a more efficient NN, we design REMIX to better utilize these existing diverse NNs by assigning proper networks for different input partitions.

On-device Inference Acceleration. There is also a large line of work focused on accelerating NN on edge devices, including NN compression methods, hardware-based methods and software-based methods.

NN compression mainly aims to reduce the network size to save computation cost and memory overhead. Pruning [33, 57] and quantization [32, 39] are proposed to cut off weights. The optimized NNs can be directly used in REMIX as network candidates.

The hardware-based methods mainly attempt to reduce inference latency by considering hardware characteristics. Some approaches proposed to boost computation by unleashing the potential of existing hardware chips on edge devices, including CPU [72, 84], GPU [40, 42, 45, 79, 81], FPGA [73] as well as domain-specific processors [24, 31, 47, 75, 83]. REMIX does not assume any certain type of edge device, and the emerging AI accelerators and hardware-aware optimization techniques can be incorporated into our framework to achieve higher efficiency.

Existing software-based NN inference acceleration approaches mainly focus on more intelligent and efficient resource scheduling [27, 34, 44, 59, 80], or make use of collective power of edge nodes [55], spatial and temporal locality [41, 78] and redundancy [30] of on-device inference workloads. REMIX belongs to software-based methods and also incorporates acceleration techniques based on resource scheduling and locality. However, our work is tailored for video analytics. The core of REMIX is the adaptive partition method and selective execution strategy, which are significant different from existing software-based NN acceleration approaches.

6 DISCUSSION AND FUTURE WORK

We notice that the performance of REMIX highly depends on the selected networks. On the one hand, the best detection accuracy and the fastest inference are determined by the largest and smallest networks in the NN candidates, respectively. On the other hand, the diversity of selected NNs would enrich the optimization space of REMIX, leading to a better performance. Although we use EfficientDet as examples in this paper, we allow users to apply any other object detection NNs.

While REMIX made the first step in building a flexible high resolution object detection framework using adaptive partitioning, several design aspects of REMIX warrant further investigation. First, it would be interesting to explore accurate and efficient techniques to detect object distribution changes that trigger partition plan update. Second, how to select frames from history for the generation of high-quality partition plans is still an open question. Lastly, more efforts need to be put in reducing the pre-processing overhead and optimizing memory footprint.

7 CONCLUSION

In this paper, we present REMIX, a fixable framework for high-resolution object detection on edge devices. Based on the understandings of object distributions and NNs' characteristics, REMIX intelligently distributes the limited compute power across the high-resolution image, by adaptively partitioning it into non-uniform blocks, and assigning each block a proper network, best balancing the total inference latency and achieved detection accuracy. We also introduce a series of techniques to make REMIX robust, effective and efficient. We evaluate REMIX on real-world videos and the results shows REMIX can either achieve about 65.3% mAP improvements on average for a given latency budget or speed up the inference by up to 8.1× while obtaining competitive accuracy, compared with SOTA object detectors.

REFERENCES

[1] 2019. Are We Ready for AI Security Cameras. https://bit.ly/2OZsT33.
[2] 2019. HiSilicon Kirin 970. https://bit.ly/3waXtWO.
[3] 2019. Qualcomm Snapdragon 855. https://bit.ly/3AeP1cb.
[4] 2019. Qualcomm Vision AI DevKit. https://bit.ly/328LjBF.
[5] 2020. City Brain. https://bit.ly/3lVXpXn.
[6] 2020. Common Objects in Context. https://cocodataset.org.
[7] 2020. Nvidia Jetson AGX Xavier. http://bit.ly/3nVJBM7.
[8] 2020. Nvidia Jetson Developer Guide. http://bit.ly/3oYilxz.
[9] 2020. PANDA Dataset. http://www.panda-dataset.com.
[10] 2020. Samsung Galaxy S20 series. https://bit.ly/39gF4iB.
[11] 2021. Amazon Go. http://amazongo.com/.
[12] 2021. Hikvision 4K camera. https://bit.ly/2NViRiT.
[13] 2021. Software-Based Power Consumption Modeling. https://bit.ly/3jY0eZc.
[14] 2021. Tensorflow. https://www.tensorflow.org/.
[15] 2021. TensorFlow Lite. https://www.tensorflow.org/lite.
[16] Mark Allman, Vern Paxson, and William Stevens. 1999. RFC2581: TCP congestion control. (1999).
[17] Ganesh Ananthanarayanan, Victor Bahl, Landon Cox, Alex Crown, Shadi Nogbahi, and Yuanchao Shu. 2019. Demo: Video Analytics - Killer App for Edge Computing. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*.
[18] Karl Johan Åström and Tore Hägglund. 2004. Revisiting the Ziegler–Nichols step response method for PID control. *Journal of process control* (2004).
[19] Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, and Jonathan Huang. 2020. Context R-CNN: Long Term Temporal Context for Per-Camera Object Detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020*.
[20] S. Bennett. 1993. Development of the PID controller. *IEEE Control Systems Magazine* (1993).
[21] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *USENIX NSDI*.
[22] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *8th International Conference on Learning Representations, ICLR 2020*.
[23] Kai Chen, Jiaqi Wang, Shuo Yang, Xingcheng Zhang, Yuanjun Xiong, Chen Change Loy, and Dahua Lin. 2018. Optimizing Video Object Detection via a Scale-Time Lattice. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*.
[24] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*.
[25] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. 2019. ChamNet: Towards Efficient Network Design Through Platform-Aware Model Adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*. 11398–11407.
[26] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. 2015. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision* (2015).
[27] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*.
[28] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. 2018. MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*. 1586–1595.
[29] Peizhen Guo, Bo Hu, and Wenjun Hu. 2021. Mistify: Automating DNN Model Porting for On-Device Inference at the Edge. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*.
[30] Peizhen Guo and Wenjun Hu. 2018. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*.
[31] Suyog Gupta and Berkin Akin. 2020. Accelerator-aware Neural Network Design using AutoML. *arXiv preprint* (2020).
[32] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint* (2015).
[33] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* (2015).
[34] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*.
[35] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S. Huang. 2016. Seq-NMS for Video Object Detection. *CoRR* (2016).
[36] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*.
[37] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[38] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[39] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* (2017).
[40] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*.
[41] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Victor Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient cross-camera video analytics on large camera networks. In *ACM/IEEE Symposium on Edge Computing (SEC 2020)*.
[42] Shiqi Jiang, Lihao Ran, Ting Cao, Yusen Xu, and Yunxin Liu. 2020. Profiling and optimizing deep learning inference on mobile GPUs. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*.
[43] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, and Wanli Ouyang. 2018. T-CNN: Tubelets With Convolutional Neural Networks for Object Detection From Videos. *IEEE Trans. Circuits Syst. Video Technol.* 28, 10 (2018), 2896–2907.
[44] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.
[45] Seyyed Salar Latifi Oskouei, Hossein Golestani, Matin Hashemi, and Soheil Ghiasi. 2016. Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android. In *Proceedings of the 24th ACM international conference on Multimedia*.
[46] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* (1989).
[47] Wenbin Li and Matthieu Liewig. 2020. A Survey of AI Accelerators for Edge Environment. In *World Conference on Information Systems and Technologies*.
[48] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. 2017. Focal Loss for Dense Object Detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*.
[49] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, and et al. Deva Ramanan. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision - ECCV 2014*.
[50] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19)*.
[51] Mason Liu and Menglong Zhu. 2018. Mobile Video Object Detection With Temporally-Aware Feature Maps. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*. 5686–5695.
[52] Mason Liu, Menglong Zhu, Marie White, Yinxiao Li, and Dmitry Kalenichenko. 2019. Looking Fast and Slow: Memory-Guided Mobile Video Object Detection. *CoRR* (2019).
[53] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In *Computer Vision - ECCV 2016*.
[54] Xiaochen Liu, Yurong Jiang, Puneet Jain, and Kyu-Han Kim. 2018. TAR: Enabling Fine-Grained Targeted Advertising in Retail Stores *(MobiSys '18)*.
[55] Yan Lu, Yuanchao Shu, Xu Tan, Yunxin Liu, Mengyu Zhou, Qi Chen, and Dan Pei. 2019. Collaborative Learning between Cloud and End Devices: An Empirical Study on Location Prediction. In *ACM/IEEE Symposium on Edge Computing (SEC)*.
[56] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The Expressive Power of Neural Networks: A View from the Width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*.
[57] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*.
[58] Wenjie Luo, Bin Yang, and Raquel Urtasun. 2018. Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting With a Single Convolutional Net. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*. 3569–3577.

[59] Akhil Mathur, Nicholas D Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. 2017. Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*.

[60] N. Megiddo. 1982. Linear-time algorithms for linear programming in R3 and related problems. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*.

[61] Alexander Neubeck and Luc Van Gool. 2006. Efficient non-maximum suppression. In *18th International Conference on Pattern Recognition (ICPR'06)*. IEEE.

[62] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[63] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv preprint* (2018).

[64] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[65] Vit Ruzicka and Franz Franchetti. 2018. Fast and accurate object detection in high resolution 4K and 8K video using GPUs. *2018 IEEE High Performance extreme Computing Conference (HPEC)* (2018).

[66] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[67] Or Sharir and Amnon Shashua. 2018. On the Expressive Power of Overlapping Architectures of Deep Learning. In *International Conference on Learning Representations (ICLR)*.

[68] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[69] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*.

[70] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and Efficient Object Detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[71] Panrong Tong, Mingqian Li, Mo Li, Jianqiang Huang, and Xiansheng Hua. 2021. Large-Scale Vehicle Trajectory Reconstruction with Camera Sensing Network. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom '21)*.

[72] Manni Wang, Shaohua Ding, Ting Cao, Yunxin Liu, and Fengyuan Xu. 2021. AsyMo: Scalable and Efficient Deep-Learning Inference on Asymmetric Mobile CPUs. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom '21)*.

[73] Shang Wang, Chen Zhang, Yuanchao Shu, and Yunxin Liu. 2019. Live Video Analytics with FPGA-based Smart Cameras. In *Workshop on Hot Topics in Video Analytics and Intelligent Edges*.

[74] Xueyang Wang, Xiya Zhang, Yinheng Zhu, Yuchen Guo, Xiaoyun Yuan, Liuyu Xiang, Zerun Wang, Guiguang Ding, David J Brady, Qionghai Dai, and Lu Fang. 2020. PANDA: A Gigapixel-level Human-centric Video Dataset. In *2020 IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*,.

[75] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. 2019. Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv preprint* (2019).

[76] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. 2018. Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification. In *Computer Vision - ECCV 2018*, Vol. 11219. 318–335.

[77] Mengwei Xu, Tiantu Xu, Yunxin Liu, and Felix Xiaozhu Lin. 2021. Video Analytics with Zero-streaming Cameras. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 459–472.

[78] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. DeepCache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 129–144.

[79] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. 2017. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*.

[80] Juheon Yi, Sunghyun Choi, and Youngki Lee. 2020. EagleEye: wearable camera-based person identification in crowded urban spaces. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20)*.

[81] Juheon Yi and Youngki Lee. 2020. Heimdall: Mobile GPU Coordination Platform for Augmented Reality Applications. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 14 pages.

[82] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide Residual Networks. In *Proceedings of the British Machine Vision Conference (BMVC)*.

[83] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. 2020. Fast Hardware-Aware Neural Architecture Search. In *Joint Workshop on Efficient Deep Learning in Computer Vision*.

[84] Minjia Zhang, Samyam Rajbhandari, Wenhan Wang, and Yuxiong He. 2018. Deepcpu: Serving rnn-based deep learning models 10x faster. In *2018 USENIX Annual Technical Conference (ATC 18)*.

[85] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: Accelerate High-Resolution Mobile Deep Vision with Content-Aware Parallel Offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom '21)*.

[86] Ganlong Zhao, Guanbin Li, Ruijia Xu, and Liang Lin. 2020. Collaborative Training between Region Proposal Localization and Classification for Domain Adaptive Object Detection.

[87] Y. Zheng, D. Huang, S. Liu, and Y. Wang. 2020. Cross-domain Object Detection through Coarse-to-Fine Feature Adaptation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[88] Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. 2018. Towards High Performance Video Object Detection. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*. 7210–7218.

[89] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Deep Feature Flow for Video Recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. 4141–4150.

[90] F. Ö. Ünel, B. O. Özkalayci, and C. Çiğla. 2019. The Power of Tiling for Small Object Detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.