Cross-Camera Inference on the Constrained Edge

Jingzong Li*§, Libin Liu†§, Hong Xu¶, Shudeng Wu‡, Chun Jason Xue*
*City University of Hong Kong, †Zhongguancun Laboratory, ¶The Chinese University of Hong Kong, ‡Tsinghua University

***Abstract*—The proliferation of edge devices has pushed computing from the cloud to the data sources, and video analytics is among the most promising applications of edge computing. Running video analytics is compute- and latency-sensitive, as video frames are analyzed by complex deep neural networks (DNNs) which put severe pressure on resource-constrained edge devices. To resolve the tension between inference latency and resource cost, we present Polly, a cross-camera inference system that enables co-located cameras with different but overlapping fields of views (FoVs) to share inference results between one another, thus eliminating the redundant inference work for objects in the same physical area. Polly's design solves two basic challenges of cross-camera inference: how to identify overlapping FoVs automatically, and how to share inference results accurately across cameras. Evaluation on NVIDIA Jetson Nano with a real-world traffic surveillance dataset shows that Polly reduces the inference latency by up to 71.4% while achieving almost the same detection accuracy with state-of-the-art systems.**

I. Introduction

With the explosive development of deep neural networks (DNNs) and extensive deployment of devices with cameras, video analytics proliferates to support a wide spectrum of services [16, 25, 30], including traffic monitoring [8], safety surveillance [44], and anomaly detection [10]. These applications usually run on the edge for two reasons: (1) Unreliable and limited upload bandwidth from edge to cloud makes video transmission the bottleneck of the entire analytics pipeline, and is detrimental to many latency-sensitive analytics tasks [16, 38, 42]; (2) Videos often contain private information that cannot or should not be exposed to the cloud [4, 38].

The tension between edge devices' inherently constrained resources and analytics tasks' ever-growing demand for compute is impeding the full potential of edge-based video analytics. This tension exists and is exacerbating for three reasons. First, DNN-based inference is inherently compute-intensive, especially for object detection models with complex convolutional layers [19, 26, 37]. Second, today's cameras are able to output 4K and even 8K videos which require more processing capabilities [27, 39]. Lastly, the compute cost also grows linearly with the number of cameras which has been growing steadily. For example, when one camera cannot cover the full view of a location such as road intersections, restaurants, etc., usually multiple cameras are deployed around the same spot to cover different perspectives [2, 25].

Much work has been done in trying to ease this tension between constrained supply and growing demand for compute resources. For example, Remix [27] partitions a frame into non-uniform blocks according to historical object distribution and adopts a cheaper DNN for blocks that contain fewer objects. FlexPatch [43] uses a vehicle-tracking-aware patching technique to reduce object detection frequency. Also, many systems choose to judiciously offload part of workloads to the cloud to relieve the burden at the edge while satisfying latency requirement [14, 17, 17, 32, 50, 53].

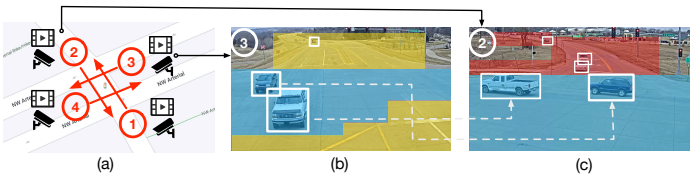


Fig. 1: An illustration of cross-camera inference for vehicle detection at an intersection. Video footages are from the NVIDIA AI City Challenge (AICC) [1]. Given inference results of camera 3 in (b), the detected bounding boxes in the overlapping FoVs (blue) are shared and mapped to camera 2 in (c). The DNN model only needs to run on the unique patches (red) of camera 2. Results are merged to generate the final output of camera 2 in (c).

Different than these prior approaches, we propose a new and complementing angle to curb the resource demand and improve the latency of video analytics. Our key insight is that multiple cameras placed at the same point of location usually have a (large) part of overlapping fields of views (FoVs), and independent repeated inference on these parts of the video frames should be avoided as much as possible. Fig. 1 shows an intersection where four cameras are set up to monitor traffic from all directions. Thus their FoVs naturally overlap: For instance, Fig. 1(b) and 1(c) show snapshots of cameras 3 and 2, respectively, taken at the same time, where both capture the two vehicles in the center of the frame. The blue zones in Fig. 1(b) and (c) highlight the overlapping FoVs.

We design Polly, a new system that carries out cross-camera inference by sharing the inference results of the overlapping FoVs from the reference camera to the target camera, so inference only needs to be done on much smaller images on the target camera. Building such a system faces two fundamental challenges: (1) how to identify the overlapping FoVs of a camera pair automatically, and (2) how to share or transfer the inference results across cameras effectively.

To cope with the first issue, Polly takes advantage of vehicle Re-identification (ReID) algorithms in computer vision that identify the same vehicle appearing in multiple cameras to correlate the overlapping FoVs. Polly also builds a fine-grained position mapping method based on multi-output regression tree to accurately map bounding boxes between cameras. Then

§Co-first authors.

This work is supported in part by funding from CUHK (4937007, 4937008, 5501329, 5501517) and a Microsoft gift fund (6906276).

in resolving the second challenge, for better accuracy performance, we find that objects only partially in the overlapping FoVs might result in false detection at the target camera when being directly shared and should not be used. The corresponding parts of these objects in the target camera's frames, which we call *low-confidence patches*, need to go through dedicated inference along with the unique patches that contain objects beyond the overlapping FoVs. Last, Polly merges the results of these two separate pipelines by addressing erroneous cases such as duplicated bounding boxes.

To summarize, our work makes the following contributions:

- We explore the overlapping FoVs of co-located cameras as a new dimension to optimize edge-based video analytics. It saves resources and accelerates inference by directly mapping the detected objects from the reference to the target camera instead of running the DNN model on the overlapping FoVs redundantly.
- We build Polly, the first system that enables such cross-camera inference. Polly recognizes and solves a number of technical issues, such as robust fine-grained position mapping between cameras, extracting and assembling patches that do require dedicated inference, and merging results of different pipelines gracefully.
- We implement Polly on a commodity edge device, NVIDIA Jetson Nano. Evaluation on a real-world dataset [1] shows that Polly improves the end-to-end latency by up to 71.4% over state-of-the-art systems almost without impairing accuracy.

II. Background and Motivation

In this section, we first introduce the background of video analytics on edge devices (§II-A). Then, we use a real-world dataset to explain the overlapping fields of view across cameras and quantitatively show how common they are in practice in §II-B. We also empirically demonstrate the relationship between model inference latency and input size. These observations motivate our idea to exploit cross-camera inference sharing as a new means to accelerate inference on the constrained edge.

A. Video Analytics on Edge Devices

Nowadays, cameras are widely deployed for various purposes, such as security surveillance [44], traffic monitoring [8], anomaly detection [10], etc. Video analytics applications usually run on edge devices to analyze the video contents, since operators may not want to share videos to the cloud for privacy reasons [4, 38], and the wide-area bandwidth from edge to cloud is also scarce [38, 42].

Due to cost and energy constraints, the computation power of edge devices is limited [27, 38]. A large body of work [12, 14, 17, 26, 32, 50, 51, 53] strives to improve the efficiency of edge-based video analytics.

Software Acceleration. Some work adopts software techniques to improve the inference latency on edge devices. For example, FlexPatch [43] uses a vehicle-tracking-aware patching technique to reduce object detection frequency while preserving accuracy. Liu et al. [32] design a system that employs low-latency techniques and decouples the rendering pipeline from the traditional vehicle tracking process while using a fast-tracking method to maintain detection accuracy. CICO [12] employs a context-aware image compression approach to achieve low latency.

Edge-Cloud Collaboration. Another line of work explores partitioning the DNNs over the edge and cloud resources to enable collaboration between them. CEVAS [17] builds a serverless infrastructure to facilitate fine-grained partitioning of DNNs and accelerate inference. ANS [50] uses online learning to automatically learn the optimal DNN partitioning on-the-fly.

Semantic Image Partitioning. In addition, some work considers using semantic information to accelerate inference. Elf [53] dispatches video frames to multiple servers for parallel processing, and does it in a content-aware manner so frames of the same objects are sent to the same server. Remix [27] partitions a frame into multiple non-uniform blocks according to historical object distribution and adopts different DNNs for different blocks.

Different than these prior approaches, we propose a new angle to reduce edge inference cost, by sharing inference results between co-located cameras at the same location that have different but overlapping fields of views (FoVs). In the following we empirically quantify the degree of overlapping with real-world traffic cameras in order to show that our idea is feasible and promising.

B. Overlapping FoVs across Cameras

As shown before in §I, very often multiple cameras are used to cover the same location and have overlapping FoVs. We now quantify the degree of overlapping for two intersections in our dataset, including the four cameras in the Intersection 1 in Fig. 1(a), and another four cameras in the Intersection 2. The camera footages are from the NVIDIA AI City Challenge (AICC) [1] dataset which covers these particular intersections. We use each camera in turn as the reference, and obtain the average ratio of overlapping FoVs with respect to the remaining cameras as shown in Table. I. The overlapping ratio ranges from 35.4% to 55.8% which shows the substantial overlapping FoVs across the cameras.

Thus, instead of performing inference independently on all cameras, a better idea is to do it only once on one camera, and share the inference results with other cameras for the overlapping FoVs. The non-overlapping parts of the frames still need to be processed separately as before. This can greatly reduce the amount of input data for inference and improve latency.

In order to show the promising potential of our idea, we conduct a simple experiment to measure inference latency with varying input data sizes. We run the YOLOv5 models [19] for object detection on a NVIDIA Jetson Nano GPU [5] using the same AICC dataset [1]. The Low-resolution inputs are obtained by bicubic downscaling from 1080p images. The model information is shown in Table II. Note that we

Reference Camera	Cam. 1	Cam. 2	Cam. 3	Cam. 4
Intersection 1	41.6%	35.4%	55.8%	47.4%
Intersection 2	36.7%	45.5%	39.8%	36.6%

TABLE I: The average overlapping ratio when different cameras are used as the reference for the intersection 1 in Fig. 1(a) and another intersection 2. The two intersections both have 4 cameras, and each column represents a reference camera. The videos we use are from NVIDIA AICC [1].

DNN	YOLOv5n	YOLOv5s	YOLOv5m	YOLOv5l	**YOLOv5x6**
Param. (M)	1.9	7.2	21.2	46.5	140.7
Size (MB)	3.9	14.1	40.8	89.3	270.0

TABLE II: Different YOLOv5 models used.

choose YOLOv5x6, the largest variant in YOLOv5, as an oracle to generate ground truth for calculating accuracy. Fig. 2 shows the results. We observe that as the frame size increases, inference latency increases dramatically (notice the y-axis's log scale) for each model. Notably, compared to the 240p, 1080p images take 17× longer to obtain an inference result. The result implies that significant latency savings can be achieved when the total number of pixels going into the model is reduced, which has also been observed in recent work [27, 53]. Now one naturally wonders about accuracy loss as a result of sharing inference results across cameras. As will be shown in §V-B, with a good design, accuracy loss is very small since objects in the overlapping FoVs are largely similar in the first place.

III. System Design

To make cross-camera inference work, two fundamental questions have to be addressed: (1) how to identify the overlapping FoVs automatically, and (2) how to share or transfer the inference results across cameras effectively, so that the overall inference accuracy loss is minimal? To this end, we now present the design of our system Polly.

A. System Overview

Fig. 3 illustrates Polly's overall design. Polly works in two phases. In the offline phase, it addresses the first design question and finds an accurate FoV mapping between cameras. This is done offline since traffic cameras are stationary, and once profiled their overlapping FoVs can be continuously used until the cameras are moved. The online phase addresses the second design question and performs cross-camera inference sharing on-the-fly at the edge. Note we assume that cameras are synchronized in time, and inference sharing is done on a per-frame basis across cameras.

Identifying Overlapping FoVs. We take advantage of vehicle Re-identification (ReID) algorithms [28, 36, 40, 45] in computer vision that identify the same vehicle appearing in multiple cameras to find the overlapping FoVs (§III-B). The ReID algorithm outputs bounding boxes of the same vehicle in

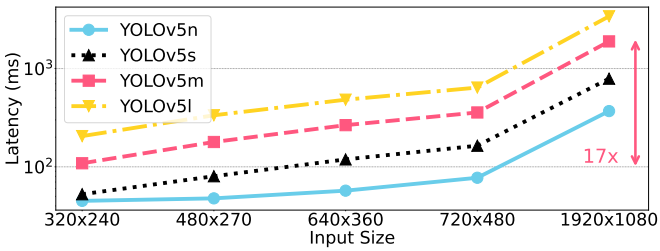


Fig. 2: Impact of input size on inference latency using YOLOv5 on a NVIDIA Jetson Nano GPU. PyTorch is used as the inference engine with the AICC dataset [1]. We repeat one hundred runs for each model and input size, and report the average.

frames from different cameras, which taken together can tell us the overall overlapping FoVs of two cameras (e.g. blue areas in Fig. 1). Polly further performs fine-grained position mapping using regression methods trained over the ReID results, such that a vehicle's position in the target camera can be directly obtained using its position in the reference camera in the online phase. The obtained overlapping FoVs are used for other auxiliary tasks, including background removal and reference camera selection to optimize system performance.

Online Inference Sharing. In the online phase, given inference results from a reference camera and knowledge of overlapping FoVs, Polly outputs the corresponding bounding boxes of the vehicles for the target camera. In the straightforward case when the reference results are given with high confidence for sharing, they are directly mapped to the target camera based on the fine-grained position mapping module introduced before (§III-C). In case the reference results have low confidence, Polly cannot use them directly. Instead, it obtains their corresponding areas in target camera's images (based on position mapping with additional fine-tuning for robustness), and crops the contents to produce the so-called *patches*. These low-confidence patches, together with the unique patches that correspond to non-overlapping and non-background areas of the frame, are piled into a new image by the patch assembler and then fed to the DNN model for inference (§III-C). Results from these two pipelines are merged to give the complete inference result of the target camera (§III-D).

B. Identifying Overlapping FoVs

We now present the design details of Polly's offline phase that automatically creates an association between overlapping FoVs of different cameras.

Overlap Profiling. As discussed earlier, we utilize the positions of the same vehicle in different cameras to profile the overlapping FoVs and establish fine-grained mapping for them. We leverage existing vehicle re-identification (ReID) algorithms, which have been well-studied in computer vision [28, 36, 40, 45]. Specifically, using time-synchronous footages from cameras, ReID assigns an ID and a bounding box to every vehicle in each frame. The same vehicle appearing in multiple cameras is identified with the same ID. The ReID results of each row has the following form, [*frame_id*, *object_id*, *left*, *top*, *width*, *height*]. *left* and *top* are coordinates of the top-left

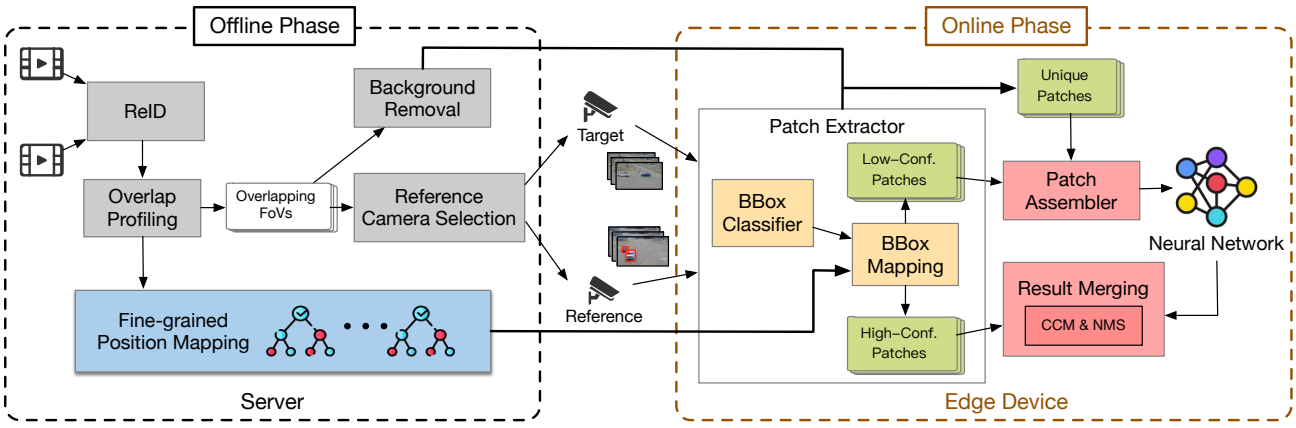


Fig. 3: Polly System Overview.

(a) Cam. 2 (b) Cam. 3

Fig. 4: Example of overlap profiling

corner of the bounding box, along with the width and height measured in pixels.

Polly then uses Algorithm 1 to find the overlapping FoVs. It iterates ReID results of two synchronous frames from two cameras. As shown in Fig. 4, the image is divided into $W \times H$ small grids. Vehicles identified as the same are marked with bounding boxes of the same color. Take the vehicle in the red bounding box as an example. Grids 1–3 and 16–20 are identified as overlapping, while grids like 4–8 and 21 are filtered out to avoid ambiguity as only a small part of them are occupied by the red bounding box, which is controled by a threshold F_T. Additionally, detailed positional information of vehicles is recorded (lines 9 and 22) to achieve a fine-grained mapping described in the next section. At the end, the overlapping FoVs such as the blue zone in Fig. 1 is established.

Fine-grained Position Mapping. Based on the FoVs from Algorithm 1, Polly can build a fine-grained position mapping for effective inference sharing. Essentially, since the cameras cover the same intersection, their perspectives are different 2-D projections of the same 3-D physical scene, and there clearly exists a stationary transformation between any given pair of these 2-D projections. We plot some trajectories of randomly selected vehicles from cameras 3 and 4, as shown in Fig. 5. We can see that vehicles with similar trajectories in one camera appear in the other one with also very similar trajectories.

Since we do not know cameras' precise location coordinates and angles, we resort to machine learning to learn this transformation or mapping using positional data S we have. The input and output are both a triplet, i.e. $[x, y, l]$ in the target camera as input, and $[\tilde{x}, \tilde{y}, \tilde{l}]$ in the reference camera as output. Here x and y represent the coordinates of a vehicle's centroid, and l is the average side length of the bounding box. The reason why we do not directly use the width and length is that a vehicle's shape varies, making it difficult to effectively map. Thus we simplify it by assuming the bounding box is a just square.

Algorithm 1: Overlapping FoVs Identification

Input :
- R_A, R_B: ReID results of cameras A and B.
- F_T: The filtering threshold.
- K: Size of each grid an image is divided into.

Output:
- O_A, O_B: The overlapping FoVs of cameras A and B.
- S: Coordinates of the centroids of the same vehicle and average side lengths of their bounding boxes.

1 **Function** `overlap_profiling`(R_A, R_B, F_T, K)**:**
2 **for** r_A *in* R_A *and* r_B *in* R_B **do**
3 **for** row_A *in* r_A **do**
4 $centroid = [\,]$;
5 $obj\_id \leftarrow$ get vehicle ID from row_A;
6 **if** obj_id *not in* r_B **then** continue ;
7 **else**
8 select row_B in r_B where vehicle ID = obj_id
9 $centroid$.extend($[x, y, l]$);
10 $box\_list \leftarrow$ split bounding box in row_A according to the grid;
11 **for** box *in* box_list **do**
12 **if** $(area\ of\ box/K) > F_T$ **then**
13 convert box to grid;
14 **if** $grid$ *not in* O_A **then**
15 insert grid into O_A;
16 $box\_list \leftarrow$ split bounding box in row_B according to the grid;
17 **for** box *in* box_list **do**
18 **if** $(area\ of\ box/K) > F_T$ **then**
19 convert box to grid;
20 **if** $grid$ *not in* O_B **then**
21 insert grid into O_B;
22 $centroid$.extend($[\tilde{x}, \tilde{y}, \tilde{l}]$);
23 S.append($centroid$);
24 **return** O_A, O_B, S;

Method	R^2	Training Time (s)	Inference Time (ms)
MLP	0.950	631	0.38
Multi-output Regression Tree	0.996	5	0.27

TABLE III: Performance of different mapping methods

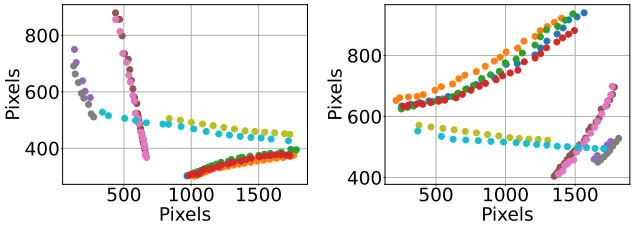


Fig. 5: Trajectories of randomly selected vehicles from cameras 3 (left) and 4 (right). Dots with the same color represent the centroids of the same vihicle.

(a) Before (b) After

Fig. 6: Effect of applying data augmentation. Red boxes are the inferred results by position mapping.

For the ML method, we compare a black-box approach with a multilayer perceptron (MLP) [18], and a white-box approach with a multi-output regression tree [34]. Here we use a MLP with three fully connected layers (with sizes of 64-64-3) and Relu activation functions. The results are shown in Table. III, we can see that regression tree works better in our case, with higher accuracy, and faster training and inference time. Thus, we adopt it as Polly's default mapping approach.

Additionally, when training the regressor, we find that the number of positional samples S are limited for a robust model, as it is impractical to collect all possible positions. So we apply **data augmentation** to enrich the training samples and enhance its accuracy. The critical insight that motivates us is that traffic movement commonly follows particular trajectory as depicted in Fig. 5. The more samples we collect, the more robust the regressor is. Thus, under limited training samples, Polly augments them by adding minor random noise to the triplet, so that the generated dots generally follow the original trajectory. Fig. 6 illustrates the benefits of data augmentation.

Background Removal. As shown in Fig. 1, each camera's view has three parts: overlapping FoVs, unique patches, and background areas, such as grass fields, bushes and trees, etc. Running vehicle detection on the background areas where vehicles do not exist at all is a waste of resource. Since cameras are stationary and the background area seldom changes, we manually identify it for each camera offline and remove it online. Automatic background removal can be explored if this assumption does not hold well in other scenarios.

Reference Camera Selection. We also need to decide which camera to serve as the reference camera. Here we adopt a simple method, that is, the camera that has the largest average overlapping FoVs with the remaining ones is selected as the reference camera. We find, as will be shown in §V-C, this method achieves the lowest latency, while being simple and easy to implement.

C. Online Inference Sharing

With everything we have thus far, in the online phase, directly mapping the overlapping FoVs from the reference camera to the target camera can still lead to unsatisfactory results due to the low quality of shared inference result. One main cause is the **perspective effect** [52]. As shown in Fig. 7, two cameras have overlapping FoVs. Because of the perspective effect, objects closer to the camera appear larger in the image. Although the truck is not physically in the overlapping area, it still appears in part of the overlapping FoV in the left camera, whereas it is completely outside the right camera's view. In this case, sharing inference would identify a truck in the right camera which is clearly wrong.

Intuitively, when some parts of the reference camera's frame has low sharing confidence, it is no longer suitable to apply fine-grained position mapping to obtain the inference result for the target camera. Our solution is then to just run the DNN model on the corresponding parts of the frame from the target camera. To do this, we first need to identify the low-confidence parts of the reference camera's frame.

Patch Extractor. We design the patch extractor to do this precisely. Specifically, the detected bounding boxes in the overlapping FoVs are classified as high- or low-confidence. The high-confidence ones are those completely in the overlapping FoVs while low-confidence ones are only partially in the FoVs. Leveraging the regression tree trained in the offline phase, low-confidence boxes are mapped to patches (with additional margins) in the target camera for further inference, while high-confidence ones are directly shared.

Besides low-confidence patches, the target camera's unique FoVs (e.g. red parts in Fig. 1) are also extracted as small rectangle patches for subsequent processing as shown in Fig. 3.

Patch Assembler. Both the low-confidence and unique patches need to be processed by the DNN for object detection, and running the model on each patch individually is inefficient as a small patch cannot efficiently utilize the GPU cores. Thus it is vital to tile these patches into one image without overlapping.

The goal of the patch assembler is then to minimize the size of the tiled rectangle so as to reduce inference latency (e.g. 1888ms for 1920×1080 full frame, 265ms for a 640×360 tiled rectangle as shown in §II-B). The problem can be treated as a variant of bin packing problem where items of varying sizes are to be packed into one 2-D bin such that its total size is minimized [48]. The problem is NP-hard and difficult to solve [21]. To simplify it, we fix the width W and orientation of the bin, where W is the width of the largest patch. Polly then adopts a simple but effective guillotine bin packing algorithm [48] to solve the problem.

To further improve latency, we observe that the object size is roughly linearly relative to the Y coordinate of the object's centroid as shown in Fig. 8. This is an intrinsic characteristic of camera view: objects closer to the camera appear larger [27]. Thus before tiling, we **down-sample** patches by factor f with Y coordinates greater than a threshold Y_T. This has very

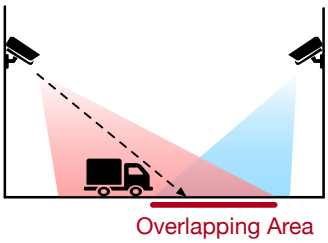
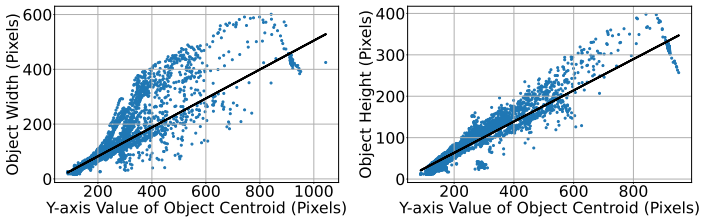


Fig. 7: An illustration of the perspective effect.



(a) Y-axis value vs. width. (b) Y-axis value vs. height.

Fig. 8: Y-axis value of object centroid versus object size.

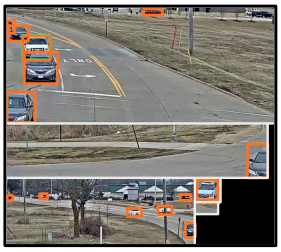


Fig. 9: An example of the tiled rectangle with detection results.

minor impact on accuracy because large objects can still be detected correctly even if they are down-sampled to a smaller rectangle, which we will verify in §V-D.

Putting everything together, Fig. 9 depicts an example of the tiled rectangle from the patch extrator. Notice the car on the right-bottom corner is down-sampled and can still be detected.

D. Accurate Result Merging

As the last step, Polly needs to combine the inference results from the two pipelines, inference sharing from the reference camera, and direct inference of the titled image. However, directly merging the results does not work due to duplicated bounding boxes. Specifically, as demonstrated in Fig. 10, there are two cases of duplicates: (1) The bounding boxes are connected on one side, as in Fig. 10(a), when the vehicle is separated into different patches; (2) The boxes are overlapping, such as in Fig. 10(c), as one vehicle may be detected in both the low-confidence patch and unique patch when it is in both areas.

For the first case, we design a method called Connected Component Merging (CCM) to effectively join two connected boxes into one. While the details are described in Algorithm 2, we illustrate the basic idea using Fig. 11. First, all boxes are projected to the two axes. CCM checks if the projections on one axis are connected (lines 7&14), and the projections on the other axis overlap (lines 11&18). When both are true, CCM merges the two connected boxes such that the merged box is minimized (lines 21-26). We use two parameters M_T and N_T to control the threshold of connection and overlap gracefully. Fig. 10(b) shows an example of the effect of CCM. For the second case, we design a variant of the non-maximum suppression (NMS) algorithm [35] to remove redundant predictions by calculating the ratio of intersection against both of the two boxes. If either one is greater than a threshold P_T, the two boxes will also be merged as lines 21-26 in Algorithm 2.

To quickly summarize, the high-level idea here is to combine the predicted bounding boxes and remove redundancy, thus improving the quality of the final result. This is an indispensible step and logically similar to how most object detection models internally merge results (e.g. [19, 37, 41]).

Algorithm 2: Connected Component Merging (CCM)

Input :
- box_1, box_2: coordinates of bounding box 1 and boudning box 2.
- M_T, N_T: the threshold of connection and overlap, respectively.

Output:
- $merged\_box$: the merged bounding box.

1 **Function** CCM (box_1, box_2) **:**
2 $x_1, y_1, w_1, h_1 \leftarrow$ the coordinates of top-left corner, and the width and height of box_1;
3 $x_2, y_2, w_2, h_2 \leftarrow$ the coordinates of top-left corner, and the width and height of box_2;
4 $flag \leftarrow False$;
5 $D_x, D_y \leftarrow$ distances of centroids along the x- and y-axis;
6 /*check if the projections on the x-axis connect*/
7 **if** $(w_1 + w_2)/2 - M_T <= D_x <= (w_1 + w_2)/2 + M_T$ **then**
8 a $\leftarrow$ max (y_1, y_2);
9 b $\leftarrow$ min $(y_1 + h_1, y_2 + h_2)$;
10 /*check if the projections on the y-axis overlap*/
11 **if** $(b - a)/h_1 >= N_T$ *or* $(b - a)/h_2 >= N_T$ **then**
12 $flag \leftarrow True$
13 /*check if the projections on the y-axis connect*/
14 **else if** $(h_1 + h_2)/2 - M_T <= D_y <= (h_1 + h_2)/2 + M_T$ **then**
15 a $\leftarrow$ max (x_1, x_2);
16 b $\leftarrow$ min $(x_1 + w_1, x_2 + w_2)$;
17 /*check if the projections on the x-axis overlap*/
18 **if** $(b - a)/w_1 >= N_T$ *or* $(b - a)/w_2 >= N_T$ **then**
19 $flag \leftarrow True$
20 /*merge two boxes*/
21 **if** $flag == True$ **then**
22 $\hat{x}$ = min (x_1, x_2);
23 $\hat{y}$ = min (y_1, y_2);
24 $\hat{w}$ = max $(x_1 + w_1, x_2 + w_2)$ - $\hat{x}$;
25 $\hat{h}$ = max $(y_1 + h_1, y_2 + h_2)$ - $\hat{y}$;
26 $merged\_box$ = $[\hat{x}, \hat{y}, \hat{w}, \hat{h}]$;
27 **return** $merged\_box$;

IV. Implementation

We implement Polly with ∼2K lines of Python on NVIDIA Jetson Nano Kit, as shown in Fig. 12. Polly uses Pytorch as the inference engine.

DNN Model. We adopt YOLOv5 [19], one of the state-of-the-art object detection techniques, as Polly's default model. Note that the design of Polly is model-agnostic and any object detection model can be used, as we will demonstrate this property in §V-F. Automatic resizing is disabled for the DNN to maintain the complete visual information.

Parameter Setting. The multi-output regression tree for fine-grained position mapping is implemented and trained with Scikit-learn [6]. We use mean squared error (MSE) as the training criterion; the maximum depth of all trees is set to 500. The filtering threshold F_T in the overlap profiling is set to 0.88. The down-sampling threshold Y_T and the down-sampling

(a) Before (b) After (c) Before (d) After

Fig. 10: Effect of result merging. The detection accuracy of the red boxes is improved after applying result merging.

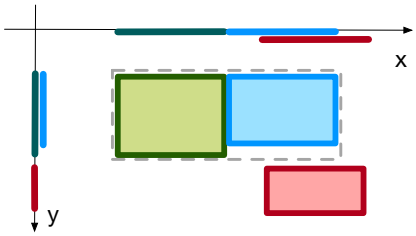


Fig. 11: Illustration of CCM. The green and blue boxes are merged after CCM.

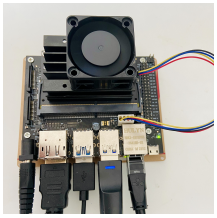


Fig. 12: NVIDIA Jetson Nano Kit.

factor f are 432 and 4 empirically. The threshold of connecting bounding boxes and overlapping, i.e. M_T and N_T, are set to 5 pixels and 0.8 respectively; P_T is also 0.8.

V. Evaluation

We now experimentally evaluate Polly in various aspects by answering the following questions:

- How well does Polly work overall in terms of latency and accuracy? (§V-B)
- How well does Polly work using different reference cameras? (§V-C)
- How much does each key design choice of Polly contribute to the overall performance gain? (§V-D)
- How efficiently does Polly utilize the compute resource? (§V-E)
- How robust is Polly in terms of detection models beyond YOLOv5? (§V-F)

A. Experimental Settings

Testbed. We run our experiments on Jetson Nano[5], which is a commonly used edge device equipped with one 128-core NVIDIA Maxwell GPU, one Quad-core ARM A57 CPU, and 4GB memory. It runs Ubuntu 18.04 LTS with kernel 4.15.0.

Dataset. We evaluate Polly using a real-world multi-camera traffic dataset from the AICC [1]. We use the video footages of the four cameras from the same intersection as shown in Fig. 1 to evaluate Polly. The original video length ranges from 195 to 211 seconds. We carefully align them to be synchronized within a time period of 180 seconds. We use the first 60-second video clips for offline profiling, and the last 120-second videos to evaluate our system. All four cameras have a frame rate of 10 FPS and a resolution of 1920×1080.

Compared Schemes. We compare Polly against the following schemes:

- ***Overlapping-Agnostic* (OA)**: This is the current approach that runs the detection model on each camera independently, without exploiting the overlapping FoVs.
- ***Naive-Merging* (NM)**: This realizes inference sharing by directly combining the shared inference results of overlapping FoVs from the reference camera with the detection results of unique FoVs.
- ***Remix-Mimic* (RM)**. We also compare to Remix [27], a state-of-the-art object detection framework on edge devices. Remix runs cheaper DNN models on portions of the frame that have fewer objects. Due to the lack of open-source code, we reproduce their work faithfully and named as *Remix-Mimic* (*RM*). We emphasize that Polly is in fact complementary to Remix and other prior work, and they can work together to achieve even better results.

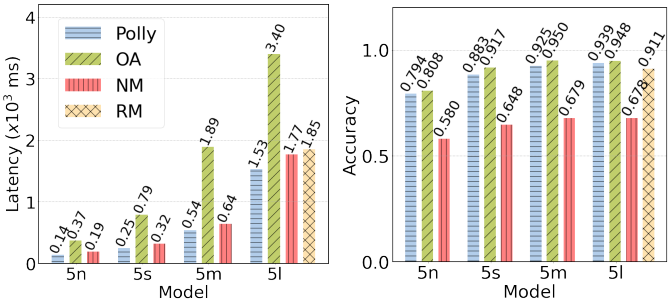
Detection Models. We use four YOLOv5 variants with different sizes [19] as detailed in Table II. Note that *RM* uses YOLOv5l as the large model when objects are small and dense, and YOLOv5n as the small one when objects are large and sparse.

Metrics. We use the average end-to-end latency and accuracy as the main performance metrics. The end-to-end latency is defined by the time between the input of image and the output of detection results. The AICC dataset [1] does not provide labels for object detection, and manually labeling all the video frames is a daunting task. Thus we choose YOLOv5x6, the largest YOLOv5 model, as the oracle to generate the pseudo-labels as ground truth, which is consistent with previous works [20, 27, 30]. Detection accuracy is measured by the commonly used F1 score, i.e. the harmonic mean of precision and recall [16, 47, 49], where 0 and 1 represent the lowest and highest accuracy, respectively.

B. Overall Performance

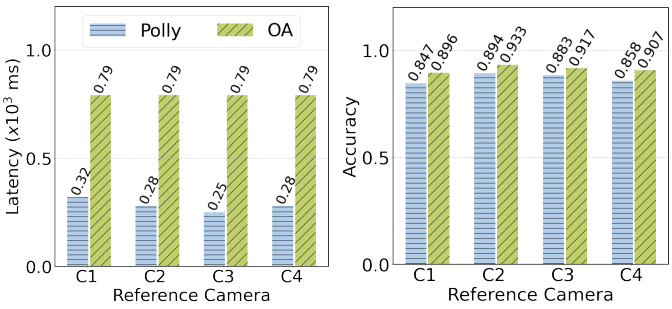
We first look at the overall performance benefits of Polly. Fig. 13 shows the results using the four YOLOv5 models. Note that because *RM* uses both YOLOv5n and YOLOv5l, we plot its bar under YOLOv5l.

As shown in Fig. 13(a), Polly's overall latency reductions range from 55.00% to 71.43% compared against *OA*. For *RM*, Polly with YOLOv5l reduces latency by 17.30%, and the reduction even reaches 70.81% with a smaller model YOLOv5m. The latency improvement of Polly attributes to three main reasons: (i) inference sharing significantly reduces the input size on which the detector needs to run; (ii) Polly's background removal component filters out pixels that never



(a) Latency (b) Accuracy

Fig. 13: The overall performance comparison of Polly and compared schemes using the YOLOv5 models. Note that because *RM* uses both YOLOv5n and YOLOv5l, we plot its bar under YOLOv5l.



(a) Latency (b) Accuracy

Fig. 14: The performance of Polly with different reference cameras. Note that the latency and accuracy results are the averages of corresponding target cameras.

contain objects; (iii) the down-sampling technique for large objects further brings more latency reduction gains.

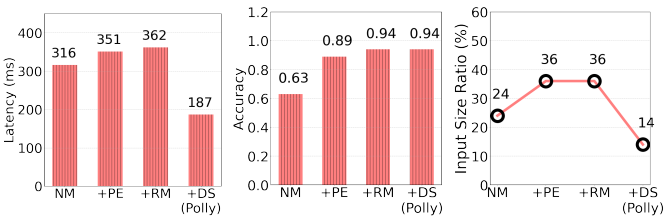
At the same time, Fig. 13(b) shows that Polly's accuracy is almost on par with *OA* across models, with at most 3.7% loss. Note that the accuracy of *OA* is the upper bound of Polly. The accuracy loss is only 0.9% for the YOLOv5l model. Moreover, combining the results of Figs. 13(a) and 13(b), we observe that Polly can use a larger model to achieve better accuracy than *OA* with a smaller model, while still being faster than *OA*. For example, Polly with YOLOv5s achieves 9.28% accuracy improvement and reduces 32.43% latency than *OA* with YOLOv5n.

Besides, both Polly and *OA* work better than *NM* in accuracy. This is because *NM* directly combines the shared inference results without careful patch extraction and consolidated result merging. Polly with YOLOv5m works similarly as *RM*, and slightly better than *RM* if it uses YOLOv5l. This is expected because *RM* uses the large YOLOv5l for the indiscernible patches and YOLOv5n for others.

C. Impact of Reference Camera Selection

We then investigate how the choice of reference camera affects performance of Polly. Fig. 14 presents the latency and accuracy that Polly achieves with different reference cameras. We only show the results using YOLOv5s here for brevity.

We observe that reference camera has a greater impact on latency compared to accuracy. The latency bias of different



(a) Latency (b) Accuracy (c) Input size ratio

Fig. 15: The latency and accuracy breakdown and corresponding average input size ratio across three key design components. The input size ratio indicates the ratio of the actual input frame size to the original frame size. Note that each component is added incrementally (e.g. +DS=*NM*+PE+RM+DS).

Polly Component	Patch Extractor	Patch Assembler	Running Detector	Result Merging
Avg. Exe. Time (ms)	4.38	3.54	168.11	11.32

TABLE IV: The average execution time of Polly's each component.

references cameras, defined as $(l_{\max} - l_{\min})/l_{\max}$, is 21.88%, while the accuracy bias is only 5.25% and accuracy performance is consistent. This implies that inference sharing is in general effective with different cameras selected as the reference, and for better performance one should use latency as the main criterion for reference selection. Since latency is directly related to the total area of overlapping FoVs, our method described in §III-B that relies on this to select reference camera works well. In this case, Polly picks camera 3, which does deliver the lowest latency.

D. Deep Dive

Next, we conduct an ablation study to investigate the impact of the key design components in Polly. We incrementally add each component to *NM*, namely Patch Extractor (PE), Result Merging (RM), and Down-Sampling (DS). Note that Patch Assembler (PA) is already integrated with *NM*. The results are shown in Fig. 15.

We see that PE brings considerable accuracy gain with 11% latency increment. This is because PE extracts the low-confidence patches that might cause false results when being directly shared, and run the detector on them to get more accurate results. RM further improves the accuracy while incurring little latency. Besides, when DS is added, latency is improved by 48.34% without accuracy loss. The reason is that DS down-samples areas with large objects and significantly reduces the input size by 61.11% on average.

Overheads. Further, we measure the execution time of each component in Polly. We repeat the runs for 50 times and report the average in Table IV. It is expected that the DNN detector takes the longest, accounting for 89.73% of the end-to-end latency. PE and RM take only 4.38ms and 11.32ms, respectively, which are negligible compared to the total latency. Notably, the patch assembler has only 3.54ms latency, which proves the efficiency of Polly's bin packing algorithm.

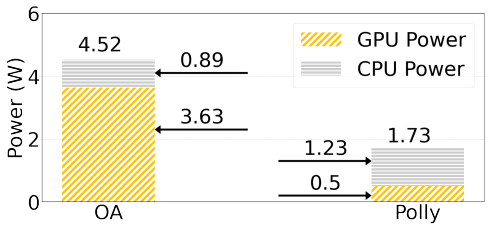


Fig. 16: The average energy consumption comparison between Polly and *OA*. The object detection model is YOLOv5s.

Scheme	Latency (ms)	Accuracy	Input Size Ratio (%)
Polly	670	0.91	12.40
OA	2220	0.93	100

TABLE V: The latency, accuracy, and input size ratio comparisons of Polly and *OA*. Both Polly and *OA* use EfficientDet-D0 [41].

E. Energy Efficiency

Utilizing energy efficiently is critical to the applications run on the constrained edge nodes [13]. We use the built-in Tegrastats Utility [7] of Jetson Nano to monitor the energy consumption over time. The monitoring period is 60s and sampling rate is 10Hz. We report the average of the monitoring period. The results of Polly and *OA* using YOLOv5s are shown in Fig. 16. Polly's GPU power is only 13.77% that of *OA*, due to inference sharing that greatly avoids redundant inference. Polly results in slightly more CPU consumption than *OA*, because of its various extra processing on the CPU. Overall, Polly saves 61.73% power against *OA*.

F. Robustness to Detection Model

To verify Polly's robustness to object detection models, we test Polly with another classic detection model EfficientDet [41]. EfficientDet [41] has different feature extractor and detection architecture compared to YOLOv5. Table V shows the results. We can see that, compared to *OA*, Polly's accuracy degradation is only 2.15%, while cutting latency from 2220ms to 670ms, i.e. a 69.82% saving for each frame. This indicates that Polly's gains are consistent when using various object detection models.

VI. Related Work

A. Cross-Camera Video Analytics

Previous work has considered optimizing cross-camera video analytics from different perspectives. Distream [46] dynamically balances the workload across cameras to fully utilize compute resources and maximize throughput. Chamelon [26] leverages the scene similarities to guide the best configuration search of spatially-related cameras, thus amortizing the profiling cost. Spatula [25] applies the spatial-temporal correlations to prune the search space of a query identity.

The closest work to Polly is CrossRoI [20]. Though both leverage the overlapping FoVs, they are different in three aspects. First, Polly focuses on enabling cross-camera inference sharing while CrossRoI aims at removing redundancy to reduce the communication cost to the cloud. Second, CrossRoI is only optimized for unique vehicle detection and it cannot detect the same vehicle appearing in other cameras, while Polly can output the complete inference results of each camera. Third, CrossRoI only identifies the coarse-grained overlapping FoVs across cameras. In contrast, Polly gains deeper knowledge of the overlapping FoVs and builds a fine-grained position mapping model to achieve effective sharing of detection results. In fact, Polly and CrossRoI are complementary and can be combined together for further improvements.

B. On-Device Inference Acceleration

A large body of work has been proposed to accelerate DNN inference on edge nodes. The first line of work is to compress the DNN model through knowledge distillation [23, 33], pruning [22, 31] or quantization [11, 15]. The second category is hardware-based acceleration [9, 24, 29] that optimizes DNN execution based on hardware architecture. The last category is software-based optimization. For example, Glimpse [14] uses a local tracking approach based on light-weight optical flow calculation to fastly detect objects; and many other efforts exploit task-specific optimization [27, 32, 38, 43, 44]. Polly belongs to the last category and is orthogonal to the above work.

VII. Discussion

A. Generalization to Other Analytics Tasks

Although Polly focuses on object detection, its main idea of leveraging overlaps to reduce inference burden of cameras is still applicable to other tasks on edge devices in the cross-camera scenario. For example, it can be extended to semantic segmentation and face recognition. Task- or domain-specific design for the pipeline is needed to maximize its benefit.

B. Batch Execution

With the continuing development of edge devices, more advanced on-board GPUs are expected to accelerate analytics tasks [30]. Some recent smart camera has an NVIDIA TX2 GPU and 32GB onboard memory [3]. Given this trend, ingesting one frame at a time may not fully utilize the hardware resources. Thus, we consider batching the frames to further boost inference speed. One possible solution is to tile all patches into a fixed-size frame and then feed in batch, while it needs careful tuning without impairing the latency. We leave batching as future work.

VIII. Conclusion

Current video analytics systems run DNN models on each camera independently. In this paper, we argue that repeatably detecting objects that appear in the overlapping FoVs of co-located cameras is clearly a waste of resources for wimpy edge devices. Thus, we present Polly, a cross-camera inference system that enables cameras with overlapping FoVs to share inference results between each other, thus dramatically reducing latency while preserving competitive accuracy. Our evaluation results with a prototype running on NVIDIA Jetson Nano show that Polly achieves almost the same detection accuracy with state-of-the-art detection systems while saving up to 71.4% latency and 61.7% computation power.

REFERENCES

[1] 2022 NVIDIA AI City Challenge. https://www.aicitychallenge.org/.

[2] Can 30,000 cameras help solve chicago's crime problem? https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html.

[3] DNNCam™ AI camera. https://groupgets.com/campaigns/429-%20dnncam-ai-camera.

[4] Enabling Data Residency and Data Protection in Microsoft Azure Regions. https://azure.microsoft.com/en-us/resources/achieving-compliant-data-residency-and-security-with-azure/.

[5] NVIDIA Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[6] Scikit Learn. https://scikit-learn.org/.

[7] Tegrastats Utility. https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/index.html.

[8] Traffic Intelligence from Video. http://www.trafficvision.com/.

[9] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos. Bit-pragmatic Deep Neural Network Computing. In *Proc. IEEE/ACM Micro*, 2017.

[10] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. Real-time Video Analytics: The Killer App for Edge Computing. *Computer*, 50(10):58–67, 2017.

[11] Benoit Jacob, et al. Quantization and Training of Neural Networks for Efficient Integer-arithmetic-only Inference. In *Proc. IEEE CVPR*, 2018.

[12] Bo Chen, et al. Context-aware Image Compression Optimization for Visual Analytics Offloading. In *Proc. ACM MMSys*, 2022.

[13] J. Chen and X. Ran. Deep Learning with Edge Computing: A Review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[14] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan. Glimpse: Continuous, Real-time Object Recognition on Mobile Devices. In *Proc. ACM SenSys*, 2015.

[15] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[16] 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. ACM SIGCOMM*, 2020.

[17] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein. Cracking Open the DNN Black-Box: Video Analytics with DNNs across the Camera-Cloud Boundary. In *Proc. ACM HotEdgeVideo*, 2019.

[18] M. W. Gardner and S. Dorling. Artificial Neural Networks (the Multilayer Perceptron)—A Review of Applications in the Atmospheric Sciences. *Atmospheric Environment*, 1998.

[19] Glenn Jocher, et al. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, Feb. 2022.

[20] H. Guo, S. Yao, Z. Yang, Q. Zhou, and K. Nahrstedt. CrossRoI: cross-camera region of interest optimization for efficient real time video analytics at scale. In *Proc. ACM MMSys*, 2021.

[21] J. Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90, 1982.

[22] Y. He, X. Zhang, and J. Sun. Channel Pruning for Accelerating Very Deep Neural Networks. In *Proc. IEEE/CVF ICCV*, 2017.

[23] G. Hinton, O. Vinyals, J. Dean, et al. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] L. N. Huynh, Y. Lee, and R. K. Balan. Deepmon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications. In *Proc. ACM MobiSys*, 2017.

[25] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, P. Bahl, and J. Gonzalez. Spatula: Efficient Cross-Camera Video Analytics on Large Camera Networks. In *Proc. IEEE/ACM SEC*, 2020.

[26] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica. Chameleon: Scalable Adaptation of Video Analytics. In *Proc. ACM SIGCOMM*, 2018.

[27] S. Jiang, Z. Lin, Y. Li, Y. Shu, and Y. Liu. Flexible High-resolution Object Detection on Edge Devices with Tunable Latency. In *Proc. ACM MobiCom*, 2021.

[28] P. Khorramshahi, A. Kumar, N. Peri, S. S. Rambhatla, J.-C. Chen, and R. Chellappa. A Dual-Path Model with Adaptive Attention for Vehicle Re-Identification. In *Proc. IEEE/CVF ICCV*, 2019.

[29] S. S. Latifi Oskouei, H. Golestani, M. Hashemi, and S. Ghiasi. Cnndroid: GPU-accelerated Execution of Trained Deep Convolutional Neural Networks on Android. In *Proc. ACM MM*, 2016.

[30] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In *Proc. ACM SIGCOMM*, 2020.

[31] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the Value of Network Pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[32] Luyang Liu, et al. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In *Proc. ACM MobiCom*, 2019.

[33] R. T. Mullapudi, S. Chen, K. Zhang, D. Ramanan, and K. Fatahalian. Online Model Distillation for Efficient Video Inference. In *Proc. IEEE/CVF ICCV*, 2019.

[34] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown. An Introduction to Decision Tree Modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 2004.

[35] A. Neubeck and L. Van Gool. Efficient Non-Maximum Suppression. In *Proc. IEEE ICPR*, 2006.

[36] Peilun Li, et al. Spatio-temporal Consistency and Hierarchical Matching for Multi-Target Multi-Camera Vehicle Tracking. In *Proc. IEEE CVPR Workshops*, 2019.

[37] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in neural information processing systems (NIPS)*, 28, 2015.

[38] Romil Bhardwaj, et al. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *Proc. USENIX NSDI*, 2022.

[39] V. Růžička and F. Franchetti. Fast and accurate object detection in high resolution 4K and 8K video using GPUs. In *Proc. IEEE HPEC*, 2018.

[40] Shuting He, et al. Transreid: Transformer-based object re-identification. In *Proc. IEEE/CVF ICCV*, 2021.

[41] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and Efficient Object Detection. In *Proc. IEEE/CVF CVPR*, 2020.

[42] J. Wang, Z. Feng, S. George, R. Iyengar, P. Pillai, and M. Satyanarayanan. Towards Scalable Edge-native Applications. In *Proc. ACM/IEEE SEC*, 2019.

[43] K. Yang, J. Yi, K. Lee, and Y. Lee. FlexPatch: Fast and Accurate Object Detection for On-device High-Resolution Live Video Analytics. In *Proc. IEEE INFOCOM*, 2022.

[44] J. Yi, S. Choi, and Y. Lee. EagleEye: Wearable Camera-based Person Identification in Crowded Urban Spaces. In *Proc. ACM MobiCom*, 2020.

[45] D. Zapletal and A. Herout. Vehicle re-identification for automatic video traffic surveillance. In *Proc. IEEE CVPR Workshops*, 2016.

[46] X. Zeng, B. Fang, H. Shen, and M. Zhang. Distream: scaling live video analytics with workload-adaptive distributed edge intelligence. In *Proc. ACM SenSys*, 2020.

[47] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee. Awstream: Adaptive Wide-Area Streaming Analytics. In *Proc. ACM SIGCOMM*, 2018.

[48] D. Zhang, L. Shi, S. C. Leung, and T. Wu. A Priority Heuristic for the Guillotine Rectangular Packing Problem. *Information Processing Letters*, 116(1):15–21, 2016.

[49] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *Proc. USENIX NSDI*, 2017.

[50] L. Zhang, L. Chen, and J. Xu. Autodidactic Neurosurgeon: Collaborative Deep Inference for Mobile Edge Intelligence via Online Learning. In *Proc. ACM WWW*, 2021.

[51] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang. Towards Cloud-Edge Collaborative Online Video Analytics with Fine-Grained Serverless Pipelines. In *Proc. ACM MMSys*, 2021.

[52] Q. Zhang and A. B. Chan. Wide-area Crowd Counting via Ground-Plane Density Maps and Multi-View Fusion CNNs. In *Proc. IEEE/CVF CVPR*, 2019.

[53] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, and Y. Zhang. Elf: Accelerate High-resolution Mobile Deep Vision with Content-aware Parallel Offloading. In *Proc. ACM MobiCom*, 2021.