

Moby: Empowering 2D Models for Efficient Point Cloud Analytics on the Edge

Jingzong Li
City University of Hong Kong

Yik Hong Cai
The Chinese University of Hong Kong

Libin Liu
Zhongguancun Laboratory

Yu Mao
City University of Hong Kong

Chun Jason Xue
City University of Hong Kong

Hong Xu
The Chinese University of Hong Kong

ABSTRACT

3D object detection plays a pivotal role in many applications, most notably autonomous driving and robotics. These applications are commonly deployed on edge devices to promptly interact with the environment, and often require near real-time response. With limited computation power, it is challenging to execute 3D detection on the edge using highly complex neural networks. Common approaches such as offloading to the cloud induce significant latency overheads due to the large amount of point cloud data during transmission. To resolve the tension between wimpy edge devices and compute-intensive inference workloads, we explore the possibility of empowering fast 2D detection to extrapolate 3D bounding boxes. To this end, we present Moby, a novel system that demonstrates the feasibility and potential of our approach. We design a transformation pipeline for Moby that generates 3D bounding boxes efficiently and accurately based on 2D detection results without running 3D detectors. Further, we devise a frame offloading scheduler that decides when to launch the 3D detector judiciously in the cloud to avoid the errors from accumulating. Extensive evaluations on NVIDIA Jetson TX2 with real-world autonomous driving datasets demonstrate that Moby offers up to 91.9% latency improvement with modest accuracy loss over state of the art.

CCS CONCEPTS

• Information systems → Data analytics; Data streaming.

KEYWORDS

Point cloud analytics, edge computing, 3D object detection

ACM Reference Format:

Jingzong Li, Yik Hong Cai, Libin Liu, Yu Mao, Chun Jason Xue, and Hong Xu. 2023. Moby: Empowering 2D Models for Efficient Point Cloud Analytics on the Edge. In *Proceedings of the 31st ACM International Conference on Multimedia (MM '23)*, October 29–November 3, 2023, Ottawa, ON, Canada. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3581783.3612158>

This work is supported in part by funding from the Research Grants Council of Hong Kong (11209520, C7004-22G), CUHK (4937007, 4937008, 5501329, 5501517), and Natural Science Foundation of Shandong Province (ZR2022QF070).

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.

MM '23, October 29–November 3, 2023, Ottawa, ON, Canada

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0108-5/23/10...\$15.00

<https://doi.org/10.1145/3581783.3612158>

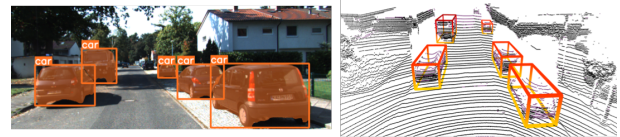


Figure 1: 2D and 3D object detection.

1 INTRODUCTION

The rapid development of Deep Neural Networks (DNNs) has empowered a number of use cases, including object detection, face recognition and image super-resolution [23, 24, 61]. One of the most promising use cases is 3D object detection, which plays a pivotal role in a wide range of applications such as robotics and autonomous driving that require accurate perception of their surrounding environment to operate well [12, 46]. 3D object detection is a fundamental basis in such perception systems, and significant research effort has been devoted to improving its accuracy [36, 48, 49, 51, 59, 60]. It takes 3D sensory data represented by point cloud as input, which is generally captured by LiDAR, to generate 3D bounding boxes. An example is illustrated in Fig. 1. Compared to its 2D counterpart, 3D object detection introduces a third dimension to characterize the location and size of an object in the real world. Due to the task complexity and large amount of data to process, 3D detection models usually have more complicated structures with inflated model sizes, posing a higher demand for computation resources. We experimentally find that with the same hardware, the inference latency of 3D detection model can be up to 41× of the 2D model (§2.2). Meanwhile, edge computing is provisioned with limited computation power. For instance, the widely used TX2 [9] has significantly fewer CUDA cores (17x less) than desktop-class GPU RTX 2080Ti [4], and even fewer (31x less) than high-end GPU Tesla A100 [5]. Thus, it is notoriously difficult to run 3D detection models on edge for near real-time processing.

To address the tension between constrained computing resources and growing demand, the de-facto standard, cloud-only approach offloads compute-intensive 3D object detection to the cloud for inference. Although computation offloading shifts the heavy burden to the cloud and significantly improves inference latency, the end-to-end latency is still unsatisfactory for practical use due to transmission delay, which accounts for the majority of the latency and bottlenecks the entire pipeline (§2.2). Despite compression techniques that can reduce point cloud size, the non-negligible compression overhead still hinders real-time streaming.

To enable 3D object detection on constrained edge, we present Moby, a new framework that addresses the limitations of edge- and cloud-only inference, by using 2D models to accelerate 3D object

detection. Rather than relying on heavy DNN-based 3D detectors, Moby proposes a lightweight transformation method that can run on the edge device, with only a few *anchor frames* offloaded to the server for 3D object detection. Moby’s approach is based on a *2D-to-3D transformation* method that constructs 3D bounding boxes using outputs from the 2D model. This transformation allows Moby to generate 3D detection results on board, without running heavy 3D models that are ill-suited for edge computing.

However, creating such a full-fledged system involves two key challenges: (1) At the frame level, how can Moby accurately and efficiently transform 2D bounding boxes into 3D ones, thus maximizing the latency benefit from 2D detection? (2) Across frames, as the error of transformation accumulates over time, how can Moby monitor the accuracy drop and decide the offloading timing?

Moby addresses these two challenges by introducing a novel system design. First, Moby starts by running a fast on-board instance segmentation model to obtain both 2D detections and segmentation masks. To utilize previous 3D detection results for better transformation, a *tracking-based association* component is designed to establish the association of 2D bounding boxes in adjacent frames. Next, we propose the *2D-to-3D Transformation* component, which is a light-weight geometric method that takes in both 2D results and point cloud to generate 3D bounding boxes efficiently. Specifically, we transfer semantic information contained in the segmentation masks to point cloud to obtain the point cluster of each potential object. Point filtration is designed to eliminate tainted points in each cluster. Then we design a sequence of geometric methods to accurately construct 3D bounding boxes by referring to previous detection results. Finally, as the error of transformation accumulates over time, a *frame offloading scheduler* is proposed to judiciously decide when to offload a new frame to launch 3D detectors for the subsequent transformation to utilize.

We implement Moby on an Jetson TX2 and a server equipped with RTX 2080Ti GPU. Evaluation on KITTI [27] dataset shows that, compared to existing approaches, Moby achieves up to 91.9% end-to-end latency reduction with only modest accuracy loss. It can even achieve 10 FPS on TX2, matching the scanning frequency of KITTI’s LiDAR for real-time processing. Besides, we observe that Moby reduces power consumption and memory usage by up to 75.7% and 48.1%, respectively, which is crucial for wimpy edges that need to save resources for more urgent tasks.

In summary, we make three key contributions:

- We investigate the system challenges of deploying 3D detection models on edge or cloud and reveal that both edge-only and cloud-only inference incur severe latency overheads, making them ill-suited for latency-sensitive tasks.
- We build Moby, the first system that enables 2D-to-3D transformation for robotics and autonomous driving applications. Moby innovatively utilizes previous detection results to enhance the transformation process. Additionally, we introduce a novel approach for coordinating edge and cloud computation using a frame offloading scheduler.
- Our experiments demonstrate that Moby offers a significant latency improvement with only modest accuracy loss against several state-of-the-art 3D detection techniques.

Model	PointPillar	SECOND	PointRCNN	PV-RCNN
Feature Extraction	Voxel based	Voxel based	Point based	Point-voxel based
Network Architecture	One Stage	One Stage	Two Stages	Two Stages

Table 1: Point cloud-based models we measure. PointPillar [36] and SECOND [60] group points into vertical columns or voxels and apply convolution over these structures; PointRCNN [49] extracts features directly from the points with permutation invariant learning; PV-RCNN [48] combines the point and voxel features.

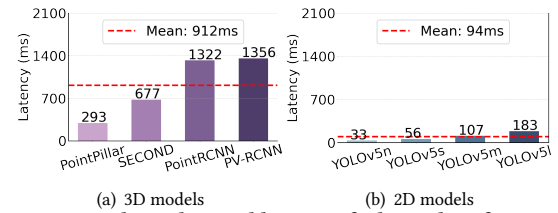


Figure 2: The end-to-end latency of edge-only inference.

2 BACKGROUND AND MOTIVATION

2.1 The Need for 3D Object Detection

State-of-the-art 2D detection techniques lack the depth information to localize objects and estimate their sizes, which is essential for tasks such as path planning and collision avoidance in robotics and autonomous driving [12, 46]. To overcome these limitations, 3D object detection methods are proposed. Generally, 3D object detection falls into one of three approaches based on the input modality [12]. The first approach uses the point cloud data from LiDAR or Radar sensors to generate 3D bounding boxes [36, 48, 49, 51, 60, 68]. The second one is image-based [17, 41, 47, 58, 62] that uses neural networks to recover the depth information from image planes for 3D bounding boxes, which can result in accuracy gaps compared to the point cloud approach. The third approach is fusion-based [18, 19, 44, 45, 56, 59] which uses one neural network to take both image and point cloud as inputs to improve performance. We emphasize that although Moby and fusion-based methods both utilize point cloud and images, the key difference is that fusion-based methods use one DNN to extract features from two modalities, while Moby replaces the 3D detection model with the lightweight 2D detection model for most LiDAR frames by leveraging an efficient and accurate 2D-to-3D transformation.

2.2 Challenges of 3D Detection on the Edge

Despite advancements in hardware and deep neural networks [12, 46], performing 3D detection is still a daunting task for resource-constrained edge devices [15]. Meanwhile, frequent detection is required to track moving objects, necessitating short detection/inference time on the device. To clearly illustrate the challenges, we first measure the inference latency of 3D object detection models on a typical edge device, NVIDIA Jetson TX2 [9], with one 256-core Pascal GPU. Next, to experiment with offloading the 3D object detection models to the cloud, we build a testbed to run the models using a GPU server and replay the real-world network traces to emulate the wide-area network conditions. Table 1 summarizes the four representative point cloud-based models we measure. The point cloud data comes from the KITTI dataset [27], the most popular one in the field of autonomous driving.

Trace (Mbps)	Mean (\pm Std)	Range	$P_{25\%}$	Median	$P_{75\%}$
FCC-1	11.89 (\pm 2.83)	[7.76, 17.76]	9.09	12.08	13.42
FCC-2	16.69 (\pm 4.69)	[8.824, 28.157]	13.91	16.07	19.43
Belgium-1	23.89 (\pm 4.93)	[16.02, 33.33]	19.84	23.46	27.73
Belgium-2	29.60 (\pm 4.92)	[20.17, 37.345]	25.18	30.761	32.76

Table 2: Statistics of four cellular bandwidth traces.

Algorithm	gzip	zlib	bzip2	lzma
Compression Time (ms)	134	238	1007	1179
Compression Ratio	1.57	1.57	1.75	1.83

Table 3: The latency and compression ratio of four common compression algorithms.

Edge-only inference. The results, shown in Fig. 2(a), indicate that: (i) The average inference latency on the device reaches 912ms for all models, which is impractical for real-world applications that require prompt environmental response; (ii) Even the one-stage models (PointPillar and SECOND) that classify bounding boxes in a single step without pre-generated region proposals have an average inference latency of 293ms and 677ms, respectively; it is not surprising that two-stage approaches take longer due to their more complex pipelines.

To gain a better understanding of the high computational cost of 3D models, we compare them against 2D detection models. We use four YOLOv5 [28] variants,¹ and the input images also come from KITTI [27], captured by its car camera with a resolution of 1242 \times 375. Results in Fig. 2(b) shows that the on-device inference latency of 2D models is much shorter. Notably, YOLOv5n only takes 33ms on average; even for the largest YOLOv5l, the inference latency is only 62% of the fastest 3D model PointPillar.

Cloud-only inference. Given the high latency of edge-only inference, naturally one considers the possibility of performing the inference tasks in the cloud with much more powerful hardware. However, with an average size of 6.96Mb per file, streaming point cloud data to the cloud is bandwidth-intensive and offsets the inference latency savings. To estimate the end-to-end latency of cloud-only inference, we simulate network conditions using bandwidth traces from empirical 4G/LTE datasets from the FCC [25] and Belgium [55], which are summarized in Table 2. The point cloud data is uploaded to the server using one TCP connection, and Linux TC [6] is used to throttle the link capacity according to the bandwidth traces. The server performs 3D detection on an RTX 2080Ti GPU. Note that we deliberately select the traces that cover different parts of the bandwidth spectrum, and they are all within a normal 4G/LTE cellular performance (\sim 10–30Mbps) [1]. It is likely that the real-world performance of cloud-only inference could be worse than what we report here in certain environments [61].

As shown in Fig. 3, we observe that: (1) Even for trace with the highest average bandwidth, Belgium 2, the average end-to-end latency across four models reaches 391ms. This is much better than edge-only latency with 912ms mean latency, but it is still not satisfactory for practical use. (2) The transmission delay over the wide Internet accounts for the majority of latency. When the network condition deteriorates, the end-to-end latency grows noticeably. For instance, the inference time with FCC-1 trace is almost twice

¹Here we use YOLOv5’s instance segmentation models, which output detection results and segmentation masks simultaneously, and are consistent with our design.

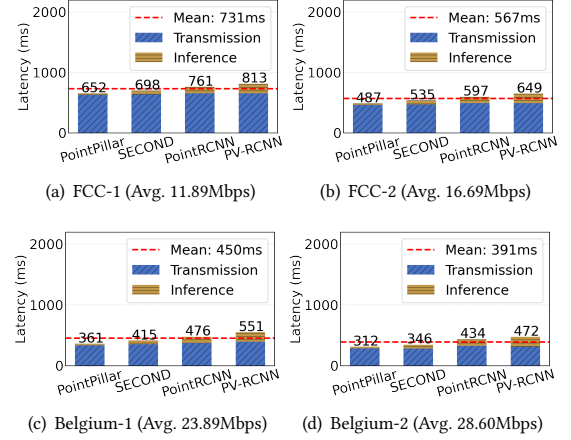


Figure 3: The end-to-end latency of cloud-only inference on four real-world network traces.

that of Belgium-2. In addition, we also consider compressing the point cloud data before transmission. We test the performance of four representative compression algorithms: gzip [3], zlib [11], bz2 [2] and lzma [7] on TX2, whose CPU runs at 2GHz. We use the implementation in Python standard library to run these algorithms on 50 point cloud files and report the average. The results are presented in Table 3. We observe that the compression time is all above 100ms, and a larger compression ratio results in a longer time. With the fastest algorithm gzip, it can reduce the end-to-end latency of the slowest FCC-1 by 78ms. However, for higher bandwidth traces, compression does not contribute to latency reduction much and can even jeopardize it. Based on these results, it is evident that offloading all 3D frames to the cloud for inference is also impractical.

2.3 Using 2D Models for 3D Detection

To sum up, the bottleneck of 3D object detection lies in the sheer amount of data that either needs to be processed by complex models on the wimpy edge, or to be transmitted over cellular networks to the cloud. Motivated by the drastically lower inference time of 2D object detection (recall Fig. 2) and the close correspondence between the 2D and 3D bounding boxes (recall Fig. 1), we cannot help but wonder: what if we use 2D detection models to extrapolate the 3D bounding boxes? Evidently, this approach would require DNN-based 3D detection on an anchor frame to provide information about the third dimension, and 2D detection can be sufficient to effectively infer the 3D results in subsequent frames. Additionally, previous detection results of anchor frame can be incorporated to perform better transformation. To our best knowledge, these have not been explored before.

3 SYSTEM DESIGN

3.1 System Overview

Fig. 4 illustrates the overall design of Moby. We summarize its workflow into three stages as highlighted in the figure: **(1) Preparation.** At time t , the edge device offloads the LiDAR frame P_t (point cloud), i.e. an anchor frame, to the cloud for 3D object detection, and obtains the 3D bounding boxes to initiate the process.

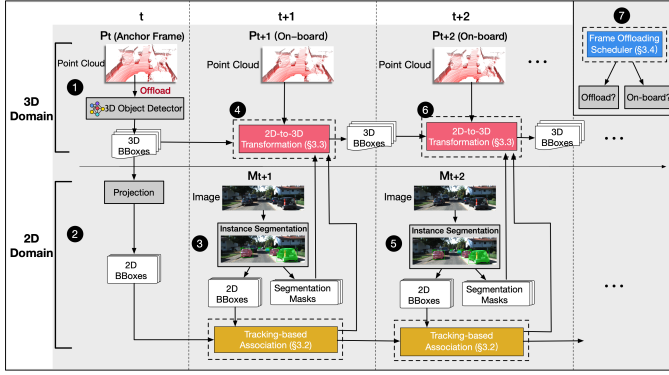


Figure 4: Moby system overview.

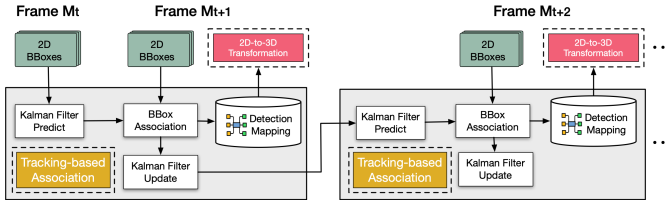


Figure 5: Overview of tracking-based association.

The 3D bounding boxes are then projected to 2D ones on the image plane (steps 1&2 in Fig. 4). **(2) Transformation.** In the next time slot $t + 1$, the edge has both a new LiDAR frame P_{t+1} and image M_{t+1} from the camera. Instead of running 3D detection on P_{t+1} again, Moby runs an instance segmentation model on M_{t+1} and obtains the 2D bounding boxes and segmentation masks. The 2D bounding boxes of both M_{t+1} and M_t are fed into the *tracking-based association* module to build a mapping between the same objects in these two image frames, which serves as the basis for associating the current objects with previous detection results. Then, using 3D bounding boxes from P_t as the reference, the *2D-to-3D transformation* module takes the segmentation masks and point cloud as input to generate 3D bounding boxes on P_{t+1} (steps 3&4). The processing at $t + 2$ is identical by using detection mapping and 3D bounding boxes from $t + 1$ as the reference (steps 5&6). **(3) Scheduling.** Moby’s 2D-empowered 3D detection inevitably causes accuracy drop especially as time goes. Thus it relies on a *scheduler* to efficiently monitor the quality of 2D-to-3D transformation and judiciously decide when to offload a new anchor frame to the cloud, so the subsequent transformations have the latest 3D information to draw upon (step 7).

We now introduce these components in detail.

3.2 Tracking-based Association

The goal of tracking-based association is to utilize tracking in the 2D domain to build the mappings between bounding boxes of the same object in two adjacent frames, which serves as a key basis for 2D-to-3D transformation. Fig. 5 shows an overview of tracking-based association.

On-device 2D Inference. To obtain 2D bounding boxes, Moby either directly projects 3D results if the current LiDAR frame is the anchor frame, or runs a 2D detection model. Instance segmentation

models are chosen as they output bounding boxes and segmentation masks simultaneously, with bounding boxes used in tracking and segmentation masks needed for the 2D-to-3D transformation.

Kalman Filter-based Tracking. Considering the limited computation resources, the tracking module must ensure real-time processing on edge devices and accurate inter-frame tracking. We extensively explore existing object tracking techniques in pixel domain and adopt Kalman filter-based tracking [14], which satisfies both requirements. With bounding boxes of each frame, the tracking pipeline (shown in Fig. 5) uses the Kalman filter to predict trajectories from frame M_t to M_{t+1} and estimate the position of boxes in the next frame. Predicted boxes are associated with 2D detection results on M_{t+1} using the Hungarian algorithm [35] and Intersection-over-Union (IoU) criterion, with association rejected if the IoU is below an threshold. The Kalman filter then updates trajectory predictions using matched detections.

3.3 2D-to-3D Transformation

We now discuss the 2D-to-3D transformation of bounding boxes, which is one of our main technical contributions. Fig. 6 shows an overview of the 2D-to-3D transformation. To incorporate 2D semantic information into 3D, Moby first projects the point cloud to the 2D segmentation masks of the image frame in the same time slot. The point cluster of each object can be identified in 3D; points of the background that are erroneously labeled as objects are filtered out. Moby then uses a light-weight geometric method, leveraging previous detection results as reference, to estimate 3D bounding boxes based on point clusters without heavy 3D models.

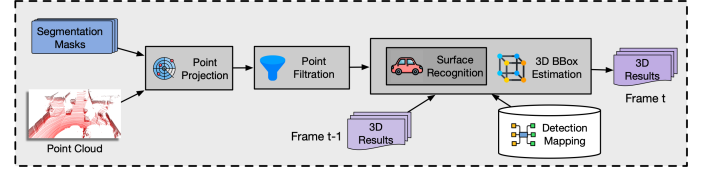


Figure 6: Overview of 2D-to-3D transformation.

Point Projection. As discussed in §2.3, the 3D and 2D frames captured at the same time have closely related semantic information. To transfer the semantic information contained in the segmentation masks to 3D, we naturally choose to project the LiDAR frame to 2D domain. This projection is time-invariant since the camera and LiDAR are both fixed once they are installed, and the multimodal-sensor systems today usually provide the transformation matrices in the sensor calibration file to project points from LiDAR to camera coordinate. With the point projection, we can then mark each 3D point to indicate which object it belongs to based on the segmentation masks, by squeezing the stacked masks along the channel dimension. Then the point cluster of each potential object can be identified. Fig. 7 illustrates the process and Fig. 7(d) shows the result with point clusters highlighted in different colors.

Point Filtration. Directly transferring segmentation masks to the point cloud may cause certain points to be erroneously marked as objects of interest and degrade accuracy. This can also be seen in Fig. 7(d), where some background points are recognized as part of a vehicle since they are projected to the same region of the vehicle in 2D [46].

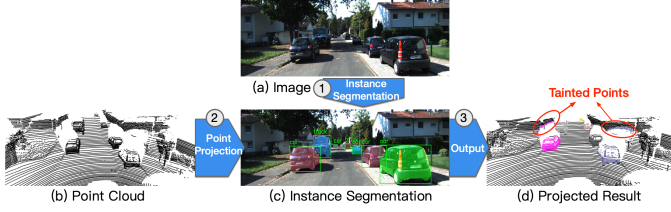


Figure 7: Example of point projection on the KITTI dataset.

Algorithm 1: Point Filtration

Input :

- PC_{old} : LiDAR points of all potential objects after point projection.
- F_T : The filtering threshold of Euclidean distance range.
- M_T : The threshold of minimum points in a potential object.
- S_T : The threshold of step size.

Output : PC_{new} : Filtered LiDAR points.

```

1 Function point_filtration( $PC_{old}, F_T, M_T, S_T$ ):
2   for  $PC_i$  in  $PC_{old}$  do
3      $PC_{new}, idx, iter \leftarrow \{\}, [], 0$ ;
4      $init\_list \leftarrow$  calculate the Euclidean distance from all points to the
5       origin of LiDAR coordinate;
6      $n_j \leftarrow$  get the nearest point to the origin according to  $init\_list$ ;
7     while  $sum(idx) < M_T$  do
8        $dist\_list \leftarrow$  calculate the distance from all points to  $n_j$ ;
9        $idx = dist\_list < F_T$ ;
10       $n_j \leftarrow$  get the point whose distance to origin is at least  $S_T$ 
11        further away;
12       $iter ++$ ;
13      if  $iter == 3$  then break ;
14       $PC'_i = PC_i[idx]$ ;
15       $PC_{new} \leftarrow PC_{new} \cup PC'_i$ ;
16   return  $PC_{new}$ ;

```

To filter out these tainted points, we design an algorithm to “purify” each object’s point cluster for better 3D bounding box estimation. The details are summarized in Algorithm 1. Chiefly, for each point cluster, it first calculates the distance from all points to the origin of LiDAR coordinate (line 4). Next, it searches for the nearest point to the origin, which most likely represents the boundary of this object and is thus called the *critical boundary point* (line 5). Then it calculates the distance from all points to the critical boundary point (line 7). Those points close to the critical boundary point are considered to belong to the object itself (line 8). If too few points are filtered this way, it suggests that the critical boundary point may not be the actual boundary of this object, which can happen when for instance vehicles are close to each other. We then add a small step size S_T and use the nearest point whose distance to the origin is at least S_T further away as the new critical point (line 9). We repeat the process until it finds a cluster that has enough points or the number of iteration exceeds three (lines 6 and 11). Fig. 8 depicts an example of the effect of point filtration. The rationale of the algorithm is based on our observation that the points of potential objects are commonly much nearer to the origin than background points. As we measured quantitatively, our point filtration algorithm removed 98% of tainted points. After point filtration, the point cluster of each potential object is clean enough for the following 3D bounding box estimation.

3D Bounding Box Estimation Moby now strives to estimate each object’s 3D bounding box based on its point cluster. A 3D bounding

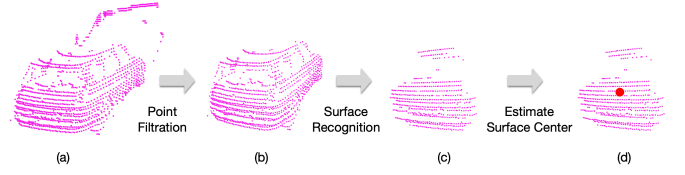


Figure 8: Key steps of point filtration and 3D bounding box estimation for a point cluster.

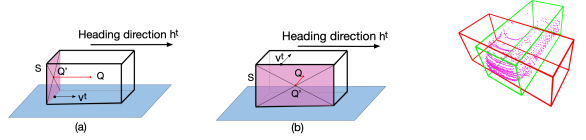


Figure 9: Two possible situations in number of points inside bounding box estimation.

Figure 10: Compare the both situations to determine the correct box.

box is represented by a seven-tuple $[x, y, z, l, w, h, \theta]$, including the object’s center $[x, y, z]$, size $[l, w, h]$, and heading angle θ relative to the x axis on the $x - y$ plane of LiDAR coordinates. It is challenging to estimate all parameters solely based on the values of points, especially the heading and center, for two main reasons: 1) The point cloud is sparse and irregular; 2) Only part of the object has a point cloud because of the self-occlusion problem [46].

We introduce a novel approach for accurately and efficiently estimating 3D bounding boxes. For each point cluster that has been associated with an object from the previous LiDAR frame by the association module, Moby first obtains the object’s size directly from the previous frame’s result (size of the same object does not change). It then estimates the heading angle, and calculates the object center based on size and heading.

Now the heading angle can be obtained from the normal vector of one of the object’s surfaces, if we know which side this surface is. Fig. 9 shows an example of this intuition with different surfaces of an object. Based on this idea, Moby uses the well-known RANSAC algorithm [26] to find a surface first. It iteratively selects three random points to form a plane until a best-fitting plane is found with the most inliers in it, with its normal vector v_t . Fig. 8(c) shows the plane it finds on the point filtration output. This plane can be either the front or rear surface as in Fig. 9(a), or the side surface as in Fig. 9(b).² To determine the correct heading, Moby calculates the angle between the normal vector v^t of the surface and the heading direction h^{t-1} of this object’s associated bounding box in the previous LiDAR frame. If the angle is less than a threshold ξ or greater than $\pi - \xi$, it means the normal vector v^t is nearly parallel to h^{t-1} , and the surface is the rear or front as shown in Fig. 9(a). Further, since the object moves continuously and is physically impossible to change its heading dramatically in one frame (i.e. 0.1s in KITTI), Moby obtains the heading direction h^t directly from v^t :

²Theoretically, the found plane can also be the top surface of a vehicle. Yet in the autonomous driving scenario, LiDAR are installed at the top of the vehicle, and the angle of laser beams makes it very unlikely for the top surface to have more points and then be found by RANSAC. Even when this happens, it can be handled by removing points in the top surface and re-run RANSAC.

$$h^t = \begin{cases} v^t, & \text{if angle between } h^{t-1} \text{ and } v^t \text{ is less than } \xi, \\ -v^t, & \text{if angle between } h^{t-1} \text{ and } v^t \text{ is greater than } \pi - \xi \\ \pi - \xi & (v^t \text{ in the opposite direction}). \end{cases} \quad (1)$$

In case v^t is perpendicular to h^{t-1} as in Fig. 9(b), we can also easily obtain h^t by rotating v^t by either 90 or 270 degrees. The heading angle θ is then directly calculated from h^t .

Lastly, in terms of the object center Q , Moby estimates it based on the surface center Q' , the heading angle θ , and the object size. Note that the surface center $Q' = [a, b, c]$ is obtained by averaging the corresponding value of all points in the plane, as shown in Fig. 8(d). Then the object center Q can be directly calculated by

$$Q = \begin{cases} [a, b, c] + [0.5 * l * \cos \theta, 0.5 * l * \sin \theta, 0], & \text{if angle between } h^{t-1} \text{ and } v^t \text{ is less than } \xi \text{ or greater than } \pi - \xi, \\ [a, b, c] + [0.5 * w * \cos \theta, 0.5 * w * \sin \theta, 0], & \text{otherwise.} \end{cases} \quad (2)$$

Finally we need to consider objects that are not associated with anything in the previous frame after tracking-based association. These are highly likely new objects that first appear in the current frame. Since we no longer have the reliable size information, we use the average length, width and height of all objects to estimate the new object's size. This can be updated after Moby schedules another anchor frame to go through the 3D detector. To estimate the heading angle, Moby uses the same procedure as described above to identify a best-fitting surface based on the point clusters from point filtration. As we no longer have prior knowledge of the heading direction (from the previous LiDAR frame), we adopt an alternative method by calculating the bounding boxes for both possibilities by Eq. (2), and the one that fits more points inside is the final output as shown in Fig. 10.

3.4 Frame Offloading Scheduler

Moby's 2D-to-3D transformation is based on the 3D detection results of the precedent anchor frame. As time goes this transformation becomes less effective simply because the scene and the objects may have changed quite much from the anchor frame. Thus from time to time, Moby needs to offload a new anchor frame to the cloud for 3D detection to assist the subsequent transformation and maintain overall accuracy. To efficiently schedule frame offloading, we must be able to determine when a new anchor frame is needed. For this purpose, a test LiDAR frame is sent to the cloud every N_T frames. Detection on the test frames happen in parallel with Moby's on-device processing. We deem the 3D detection results as ground truth to obtain the error of the 2D-to-3D transformation on the same test frame. When the error is larger than a threshold, the next LiDAR frame is designated as the new anchor frame and also sent to the cloud for inference. The subsequent on-device processing is blocked until results of the new anchor frame is received by the edge device. Fig. 11 depicts the complete process. Our design strikes a fine balance between overheads and effectiveness.

Recomputation. Since the process of waiting anchor frame's result is synchronous, the operational flow of Moby is blocked to wait for the result from server to continue the on-board processing. To

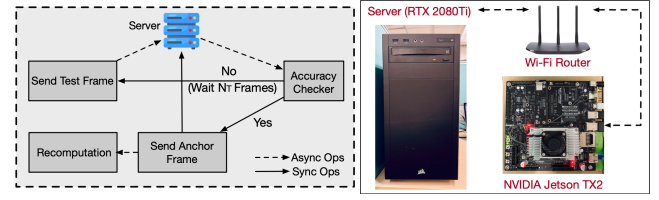


Figure 11: Overview of frame offloading scheduler.

Figure 12: Our evaluation hardware platform.

make use of the idle computation resources on edge device during the waiting time, we propose a mechanism called *recomputation* to recompute the 3D results of past intermediate frames using the results of test frames. Specifically, some intermediate results such as outputs of instance segmentation model are stacked. When an anchor frame is triggered, the intermediate results and the stale output of test frame are utilized to re-run the 2D-to-3D transformation to recompute 3D bounding boxes. We observe that the recomputation time is much less than the offloading time and is completely hidden without inducing extra latency.

4 IMPLEMENTATION

Our framework is implemented with ~4K lines of Python. We use a Jetson TX2 [9] as the edge device and a desktop with Intel i7-9700K CPU and RTX 2080Ti GPU as the server. As shown in Fig. 12, they are physically connected to a Wi-Fi router and communicate via TCP. We deploy OpenPCDet [53] on the server as the inference engine. The filtering threshold F_T , number of minimum points M_T and step size S_T are empirically set to 4.5, 24 and 12 in point filtration. ξ in the box estimation is 30 degrees. As for the offloading scheduler, the accuracy threshold and the gap of test frame, i.e. Q_T and N_T , are set to 0.7 and 4, respectively.

5 EVALUATION

We now evaluate Moby by answering the following questions:

- How well does Moby work in terms of latency and accuracy?
- How much does each key design choice contribute to the overall performance gain?
- How sensitive is Moby to the key parameters?
- How efficiently does Moby utilize the edge resources?

5.1 Experimental Setup

Datasets. All experiments are done on the KITTI dataset [27], a real-world autonomous driving benchmark, with synced LiDAR scans and camera images at 10 FPS. It provides 3D ground truth for cars, pedestrians, and cyclists. Currently we only focus on the most challenging class, car, as it moves the fastest. We use the same empirical bandwidth traces as shown in Table 2.

Performance Metrics. We use the end-to-end latency and accuracy as main performance metrics. The end-to-end latency is the time between the input of point cloud cloud and the output of 3D detection results. We consider the object to be successfully detected if the 3D IoU between detection and ground truth exceeds 40% (i.e., accurately located). F1 score is used to measure accuracy, i.e. the harmonic mean of precision and recall, as widely adopted in existing work [16, 24, 50, 63, 65].

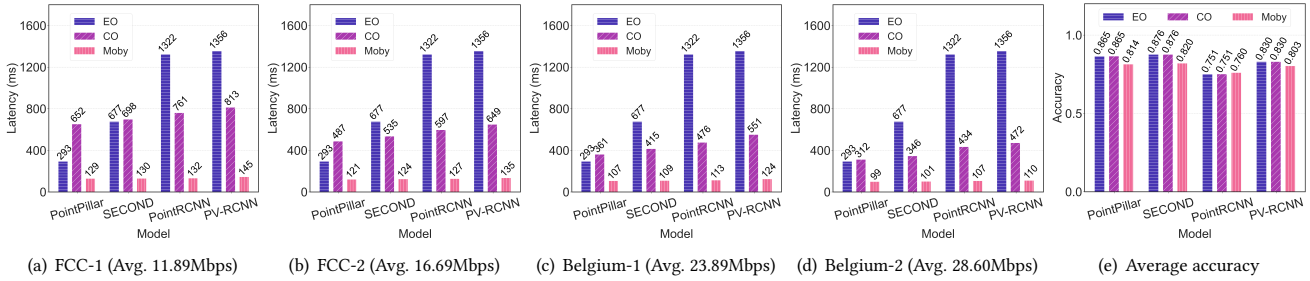


Figure 13: The latency (a-d) and accuracy (e) comparison of Moby, edge-only, and cloud-only deployment approaches.

Models. We adopt YOLOv5n from the YOLOv5 codebase[28] as Moby’s default instance segmentation model. Unless otherwise specified, Moby uses the same 3D object detection model as the baseline systems. Note that Moby is not model-dependent: any 3D object detection and instance segmentation model can be applied.

5.2 Overall Performance

We evaluate Moby comprehensively from two different perspectives: (1) deployment approach. Here we show how well Moby performs compared to two common scenarios of edge inference, i.e., completely running on the edge and offloading to cloud; (2) acceleration method. Here we compare Moby to other methods that accelerate 3D object detection.

5.2.1 Deployment Approaches. We first compare Moby with the following two common deployment approaches:

- **Edge Only (EO):** In this scheme, the 3D models are deployed on the edge device only to run inference.
- **Cloud Only (CO):** The point cloud data is fully offloaded over 4G/5G networks to the server for inference. The end-to-end latency involves both transmission and inference delay.

Fig. 13 shows the latency comparison results using four representative point cloud based 3D detection models, as described in Table 1. Observe that Moby outperforms EO and CO in latency with significant margins ranging from 56.0% to 91.9% across models. Notably, Moby achieves the lowest latency 99ms with PointPillar in Belgium-2 trace, matching the 10 FPS frame rate in KITTI for real-time processing. The latency gain is much more salient for two-stage models, such as PointRCNN and PV-RCNN, as their architectures pose much severer computation burden. Moby delivers speedups across bandwidth traces. Even for the lowest bandwidth trace, FCC-1, Moby reduces latency by 56.0% compared to running the fastest PointPillar onboard. As the bandwidth increases, the latency gain reaches 66.2%. The latency improvement mainly comes from the design choice of replacing the 3D object detector with a much cheaper 2D model that can run on the edge device, which dramatically reduces both the inference time and transmission time (to the cloud).

Moby has little impact on 3D object detection accuracy. Fig. 13(e) reports overall accuracy for all schemes. Note that as EO and CO both run 3D object detectors for all frames, they have the same accuracy with the same model. For PointRCNN, Moby achieves almost the same or slightly higher accuracy (0.760 vs 0.751) due to the effective 2D-to-3D transformation mechanism. For the other three models, accuracy drops slightly from 0.027 to 0.056 due

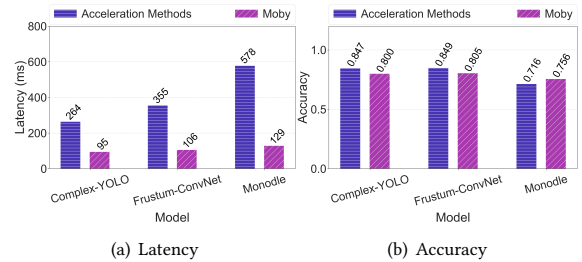


Figure 14: Comparison of Moby and alternative acceleration methods.

to: (i) possible occlusion problem in 2D images and (ii) bounding box estimation errors caused by sparse point clusters.

5.2.2 Acceleration Methods. Next, we compare Moby to three methods that accelerate 3D object detection in different ways.

- **Complex-YOLO** [51]: It converts point cloud data to birds-eye-view RGB maps and uses a light-weight YOLO-based architecture to predict 3D boxes. It is reported to achieve 50 FPS on an NVIDIA TitanX GPU [51].
- **Frustum-ConvNet** [59]: A fusion-based approach that utilizes 2D region proposals to narrow down the 3D space for acceleration.
- **Monodle** [40]: State-of-the-art image-based 3D detection approach that only use monocular images as input.

For a fair comparison, we also run Moby fully onboard for the anchor frames since these methods here do not offload to cloud, and use the same 3D detection method as the respective baseline.

Based on Fig. 14(a), we observe that Moby shows significant latency improvement against the three alternative approaches. Specifically, Moby cuts down latency by 64.0% compared to Complex-YOLO with only minor accuracy loss, and the same advantage still holds against the fusion-based Frustum-ConvNet. Notably, compared with the image-based approach Monodle, Moby reduces the latency by 77.6% while improving accuracy by 5.5%. It is worth mentioning that Moby’s accuracy is hindered by Monodle’s performance compared to other approaches, as its inaccurate output indirectly affects the transformation accuracy of Moby. The relatively low accuracy of Monodle is attributed to its inability to detect locations of distant objects, which is common among image-based acceleration methods for 3D object detection [46, 62]. We also tried another two image-based methods, Deep3DBox [41] and Pseudo-LIDAR++ [62]. However, they take 2834ms and 5889ms, respectively, which are too slow to execute on edge devices.

Components	Accuracy	Latency (ms)	On-board Latency (ms)
TRS	0.762	88.44	88.44
TRS+FOS	0.787	112.06	89.45
TRS+FOS+TBA	0.814	99.23	76.29

Table 4: Impact of each design component.

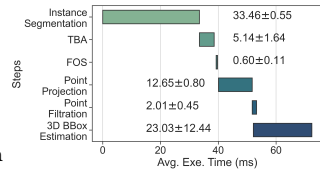
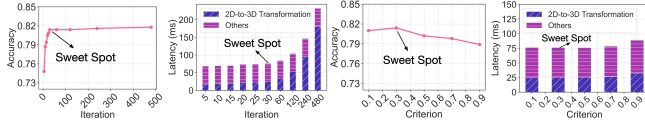


Figure 15: The avg. execution time of key steps over 300 runs.



(a) Iteration number (b) Iteration number (c) Association Criterion vs. Accuracy (d) Association Criterion vs. Latency

Figure 16: The impact of key parameters. Note that here we only report the on-board latency for clearer comparison.

5.3 Microbenchmarks

Contribution of Each Component. We evaluate the contribution of each key design component in Moby. Starting with only 2D-to-3D transformation (TRS) to generate 3D results, we incrementally add each component, namely Frame Offloading Scheduler (FOS), and Tracking-based Association (TBA). Table 4 shows the results. We observe that TRS can already achieve decent accuracy, proving the effectiveness of 2D-to-3D transformation. With FOS, accuracy is improved as some frames are offloaded to server for inference. Offloading increases the end-to-end latency while on-board latency remains unchanged. TBA further improves accuracy by using previous detection results to estimate more accurately. The slightly drop of the end-to-end and on-board latency is because TBA reduces the computation overhead of TRS if boxes are successfully associated.

Overheads. We measure the execution times of key steps of Moby and report the average in Fig. 15. Instance segmentation takes the longest, accounting for 43.9% of the onboard latency, followed by 3D bounding box estimation and point projection with 30.1% and 16.6%, respectively, because these two steps involve multiple matrix multiplications. TBA and FOS have negligible latency, taking only 5.14ms and 0.60ms. Notably, Point filtration has only 2.01ms, proving the efficiency of Algorithm 1.

5.4 Sensitivity Analysis

We also benchmark Moby by changing its key parameters.

Number of Iterations in RANSAC. The number of iterations in RANSAC is crucial in bounding box estimation and affects Moby’s accuracy and on-board latency, as shown in Fig. 16(a) and Fig. 16(b). A small number reduces latency but may lead to premature or inaccurate surface recognition, while a larger number improves robustness at the expense of computation overhead. We find that 30 iterations strike an appropriate balance between accuracy and computation cost.

Sensitivity to Association Criterion. Fig. 16(c) and Fig. 16(d) test the impact of association criterion, i.e., the IoU threshold used to associate two bounding boxes in 2D space as described in §3.2. A larger criterion makes it harder to associate bounding boxes and cannot effectively utilize previous 3D results. We empirically choose

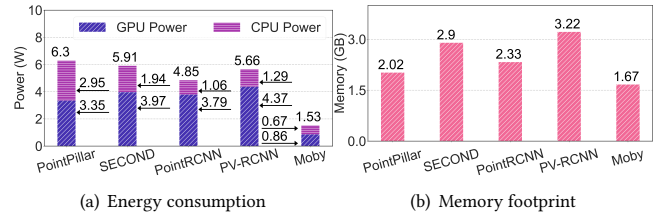


Figure 17: System efficiency.

0.3 as the association criterion, as the accuracy gain diminishes after it is greater than 0.3.

5.5 System Efficiency

Finally, we evaluate the system efficiency of Moby.

Energy Consumption. We measured Moby’s energy consumption using Tegrastats Utility [10] for 2 minutes with a 10Hz sampling rate. Moby consumes the least power compared to baselines with 73% savings on average. Its power consumption is only 24.2% of PointPillar, and reduces GPU and CPU power by 74.3% and 77.3%, respectively.

Memory Footprint. We measure the memory footprint of Moby and the results are shown in Fig. 17(b). Moby’s overall memory reduction ranges from 17.3% to 48.1%, which is expected as it only needs to load a 2D detection model into the memory that is much smaller than 3D object detection models. Moby shows its advantage in deployment on the wimpy edge devices, such as Jetson Nano [8] that only has 4GB memory shared by both CPU and GPU.

6 RELATED WORK

On-device Inference Acceleration. Lots of efforts have been devoted to accelerating DNN inference on edge devices. Model compression techniques reduce the model size and accelerate computation by network pruning [29, 30, 39], quantization [13, 21], and knowledge distillation [31, 42]. Hardware-based methods design specialized accelerators for inference pipelines using FPGA [54, 64] or ASIC [20, 57]. The last line of work is software-based methods [16, 33, 37, 52, 61]. Moby falls under the software-based category and is orthogonal to the existing work.

Edge-Cloud Collaboration. Besides on-device inference, the emergence of edge devices has also sparked research on edge-cloud collaboration. For example, filter-based methods [22, 38, 67] investigate how to prune the large data volume before transmission. Partition-based methods [32, 34, 66] explore partitioning the DNN models across edge and cloud. Feedback-based methods [16, 24, 43] rely on the feedback from server to assist local processing on the edge. Unlike existing work that all targets the 2D domain, our work explores edge-cloud collaboration for more compute-intensive 3D detection task that poses severer challenges to deploy on edge.

7 CONCLUSION

We present Moby, a novel framework for fast and accurate 3D object detection on edge devices. It leverages the 2D-to-3D transformation to significantly reduce the computational resource demand of inference, and judiciously offload the anchor frames to the cloud to maintain accuracy. Extensive evaluation shows that Moby reduces end-to-end latency by up to 91.9% with little accuracy drop.

REFERENCES

- [1] 2020. Average 4G Speed: How Fast Is 4G LTE Compared To 4G+?. <https://commsbrief.com/average-4g-speed-how-fast-is-4g-lte-compared-to-4g/>.
- [2] 2023. bz2 — Support for bz2 compression. <https://docs.python.org/3/library/bz2.html>.
- [3] 2023. gzip — Support for gzip files. <https://docs.python.org/3/library/gzip.html>.
- [4] 2023. Jetson TX2 GPU vs GeForce RTX 2080 Ti. <https://technical.city/en/video/GeForce-RTX-2080-Ti-vs-Jetson-TX2-GPU>.
- [5] 2023. Jetson TX2 GPU vs Tesla A100. <https://technical.city/en/video/Tesla-A100-vs-Jetson-TX2-GPU>.
- [6] 2023. Linux Traffic Control (TC). <https://linux.die.net/man/8/tc>.
- [7] 2023. lzma — Compression using the LZMA algorithm. <https://docs.python.org/3/library/lzma.html>.
- [8] 2023. NVIDIA Jetson Nano Developer Kit. <https://developer.nvidia.com/embedded/jetson-nano>.
- [9] 2023. NVIDIA Jetson TX2 Developer Kit. <https://developer.nvidia.com/embedded/jetson-tx2>.
- [10] 2023. Tegrastats Utility. https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvlib_docs/DRIVE_OS_Linux_SDK_Development_Guide/Utilities/util_tegrastats.html.
- [11] 2023. zlib — Compression compatible with gzip. <https://docs.python.org/3/library/zlib.html>.
- [12] Eduardo Arnold, Omar Y Al-Jarrah, Mehrdad Dianati, Saber Fallah, David Oxtoby, and Alex Mouzakitis. 2019. A Survey on 3D Object Detection Methods for Autonomous Driving Applications. *IEEE Transactions on Intelligent Transportation Systems* 20, 10 (2019), 3782–3795.
- [13] Benoit Jacob, et al. 2018. Quantization and Training of Neural Networks for Efficient Integer-arithmetic-only Inference. In *Proc. IEEE CVPR*.
- [14] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple online and realtime tracking. In *Proc. IEEE ICIP*. 3464–3468. <https://doi.org/10.1109/ICIP.2016.7533003>
- [15] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Nikolaos Karianakis, Yuanchao Shu, Kevin Hsieh, Victor Bahl, and Ion Stoica. 2022. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *Proc. USENIX NSDI*.
- [16] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices. In *Proc. ACM SenSys*.
- [17] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 2016. Monocular 3D Object Detection for Autonomous Driving. In *Proc. IEEE/CVF CVPR*.
- [18] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. 2017. Multi-view 3D Object Detection Network for Autonomous Driving. In *Proc. IEEE/CVF CVPR*.
- [19] Yukang Chen, Yanwei Li, Xiangyu Zhang, Jian Sun, and Jiaya Jia. 2022. Focal Sparse Convolutional Networks for 3D Object Detection. In *Proc. IEEE/CVF CVPR*.
- [20] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadianna: A Machine-Learning Supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 609–622.
- [21] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv preprint arXiv:1602.02830* (2016).
- [22] Kaikai Deng, Dong Zhao, Qiaoyue Han, Shuyue Wang, Zihan Zhang, Anfu Zhou, and Huadong Ma. 2022. Geryon: Edge Assisted Real-time and Robust Object Detection on Drones via mmWave Radar and Camera Fusion. *Proc. ACM IMWUT* 6, 3 (2022), 1–27.
- [23] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image Super-Resolution Using Deep Convolutional Networks. *Proc. IEEE TPAMI* 38, 2 (2015), 295–307.
- [24] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. 2020. Server-driven Video Streaming for Deep Learning Inference. In *Proc. ACM SIGCOMM*.
- [25] Federal Communications Commission. 2022. Measuring Broadband America. <https://www.fcc.gov/general/measuring-broadband-america>.
- [26] Martin A Fischler and Robert C Bolles. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM* 24, 6 (1981), 381–395.
- [27] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. IEEE/CVF CVPR*.
- [28] Glenn Jocher, et al. 2022. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation. <https://doi.org/10.5281/zenodo.7347926>
- [29] Song Han, Huiji Mao, and William J Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149* (2015).
- [30] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *Proc. IEEE/CVF ICCV*.
- [31] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531* (2015).
- [32] Hyuk-Jin Jeong, Hyeon-Jae Lee, Chang Hyun Shin, and Soo-Mook Moon. 2018. IONN: Incremental Offloading of Neural Network Computations from Mobile Devices to Edge Servers. In *Proc. ACM SoCC*. 401–411.
- [33] Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. 2021. Flexible High-resolution Object Detection on Edge Devices with Tunable Latency. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 559–572.
- [34] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.
- [35] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.
- [36] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. 2019. Pointpillars: Fast Encoders for Object Detection from Point Clouds. In *Proc. IEEE/CVF CVPR*.
- [37] Jingzong Li, Libin Liu, Hong Xu, Shudeng Wu, and Chun Jason Xue. 2023. Cross-Camera Inference on the Constrained Edge. In *Proc. IEEE INFOCOM*.
- [38] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In *Proc. ACM SIGCOMM*.
- [39] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the Value of Network Pruning. *arXiv preprint arXiv:1810.05270* (2018).
- [40] Xinzhu Ma, Yinmin Zhang, Dan Xu, Dongzhan Zhou, Shuai Yi, Haojie Li, and Wanli Ouyang. 2021. Delving Into Localization Errors for Monocular 3D Object Detection. In *Proc. IEEE/CVF CVPR*.
- [41] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 2017. 3D Bounding Box Estimation Using Deep Learning and Geometry. In *Proc. IEEE/CVF CVPR*.
- [42] Ravi Teja Mullanpudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. 2019. Online Model Distillation for Efficient Video Inference. In *Proc. IEEE/CVF ICCV*.
- [43] Vinod Nigade, Ramon Winder, Henri Bal, and Lin Wang. 2021. Better never than late: Timely edge video analytics over the air. In *Proc. ACM SenSys*.
- [44] Su Pang, Daniel Morris, and Hayder Radha. 2020. CLOCs: Camera-LiDAR object candidates fusion for 3D object detection. In *Proc. IEEE/RSJ IROS*.
- [45] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. 2018. Frustum Pointnets for 3D Object Detection from RGB-D Data. In *Proc. IEEE/CVF CVPR*.
- [46] Rui Qian, Xin Lai, and Xirong Li. 2022. 3D Object Detection for Autonomous Driving: A Survey. *Pattern Recognition* (2022).
- [47] Cody Reading, Ali Harakeh, Julia Chae, and Steven L Waslander. 2021. Categorical depth distribution network for monocular 3d object detection. In *Proc. IEEE/CVF CVPR*.
- [48] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. 2020. Pv-rcnn: Point-Voxel Feature Set Abstraction for 3D Object Detection. In *Proc. IEEE/CVF CVPR*.
- [49] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. 2019. Pointcrnn: 3d Object Proposal Generation and Detection from Point Cloud. In *Proc. IEEE/CVF CVPR*.
- [50] Xian Shuai, Yulin Shen, Yi Tang, Shuyao Shi, Luping Ji, and Guoliang Xing. 2021. milliEye: A Lightweight mmWave Radar and Camera Fusion System for Robust Object Detection. In *Proc. ACM IoTDI*.
- [51] Martin Simony, Stefan Milzy, Karl Amendey, and Horst-Michael Gross. 2018. Complex-yolo: An Euler-Region-Proposal for Real-Time 3D Object Detection on Point Clouds. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*.
- [52] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. 2022. TorchSparse: Efficient Point Cloud Inference Engine. *Proc. of MLSys* (2022).
- [53] OpenPCDet Development Team. 2020. OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds. <https://github.com/open-mmlab/OpenPCDet>.
- [54] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Visser. 2017. Finn: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*. 65–74.
- [55] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alfai, T. Bostoen, and F. De Turck. 2016. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters* 20, 11 (2016), 2177–2180.
- [56] Sourabh Vora, Alex H Lang, Bassam Helou, and Oscar Beijbom. 2020. Pointpainting: Sequential Fusion for 3D Object Detection. In *Proc. IEEE/CVF CVPR*.
- [57] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. In *Proc. IEEE HPCA*.
- [58] Yan Wang, Wei-Lun Chao, Divyansh Gard, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. 2019. Pseudo-lid From Visual Depth Estimation:

- Bridging the Gap in 3D Object Detection for Autonomous Driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
- [59] Zhixin Wang and Kui Jia. 2019. Frustum Convnet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection. In *Proc. IEEE/RSJ IROS*.
- [60] Yan Yan, Yuxing Mao, and Bo Li. 2018. Second: Sparsely Embedded Convolutional Detection. *Sensors* 18, 10 (2018), 3337.
- [61] Juheon Yi, Sunghyun Choi, and Youngki Lee. 2020. EagleEye: Wearable Camera-based Person Identification in Crowded Urban Spaces. In *Proc. ACM MobiCom*.
- [62] Yurong You, Yan Wang, Wei-Lun Chao, Divyansh Garg, Geoff Pleiss, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. 2019. Pseudo-lidar++: Accurate Depth for 3D Object Detection in Autonomous Driving. *arXiv preprint arXiv:1906.06310* (2019).
- [63] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. 2018. Awstream: Adaptive Wide-area Streaming Analytics. In *Proc. ACM SIGCOMM*.
- [64] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*. 161–170.
- [65] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *Proc. USENIX NSDI*.
- [66] Shigeng Zhang, Yinggang Li, Xuan Liu, Song Guo, Weiping Wang, Jianxin Wang, Bo Ding, and Di Wu. 2020. Towards Real-time Cooperative Deep Inference over the Cloud and Edge End Devices. *Proc. ACM IMWUT* 4, 2 (2020), 1–24.
- [67] 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 *Proc. ACM MobiCom*.
- [68] Yin Zhou and Oncel Tuzel. 2018. Voxelnet: End-to-end Learning for Point Cloud Based 3D Object Detection. In *Proc. IEEE/CVF CVPR*.