Computer Networks 269 (2025) 111382

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Accelerating point cloud analytics on resource-constrained edge devices

Jingzong Li [a], Yik Hong Cai [b], Libin Liu [c], Yu Mao [d], Chun Jason Xue [e], Hong Xu [b,*]

[a] The Hang Seng University of Hong Kong, Hong Kong Special Administrative Region of China
[b] The Chinese University of Hong Kong, Hong Kong Special Administrative Region of China
[c] Zhongguancun Laboratory, Beijing, China
[d] City University of Hong Kong, Hong Kong Special Administrative Region of China
[e] MBZUAI, Abu Dhabi, United Arab Emirates

ARTICLE INFO

Keywords:
Edge computing
Point cloud streaming
Computation offloading
On-device acceleration
Real-time 3D object detection

ABSTRACT

3D object detection is crucial in various applications, particularly in the fields of autonomous driving and robotics. These applications are typically installed on edge devices to quickly interact with the environment and often necessitate nearly instantaneous reaction. Executing 3D detection on the edge using complicated neural networks is daunting due to the constrained computational resources. Conventional methods like offloading tasks to the cloud result in substantial delays because of the extensive volume of point cloud data being transmitted. In order to address the conflict between constrained edge devices and demanding inference tasks, we investigate the potential of empowering rapid 2D detection to extrapolate 3D bounding boxes. To achieve this goal, we introduce Moby, an innovative system that showcases the practicality and promise of our methodology. We propose a lightweight transformation to efficiently and accurately produces 3D bounding boxes using 2D detection results, eliminating the need for heavy 3D detectors. In addition, we develop a frame offloading scheduler that determines the optimal timing to activate the 3D detector in the cloud, preventing the accumulation of errors. Our evaluations conducted on the NVIDIA Jetson TX2 using real autonomous driving dataset show that Moby provides a latency improvement of up to 91.9% with only minimal decrease in accuracy.

1. Introduction

The rapid advancement of Deep Neural Networks (DNNs) has transformed various fields, including object detection, face recognition, and image super-resolution [1–3]. Among these applications, 3D object detection is particularly crucial in domains such as robotics and autonomous driving, where precise environmental perception is essential for ensuring safety [4,5]. This task relies on 3D sensory data, typically captured by LiDAR in the form of point clouds. By analyzing these point clouds, 3D object detection models generate accurate 3D bounding boxes from raw data. An illustration of this process is shown in Fig. 1.

Compared to its 2D counterpart, 3D object detection introduces an additional dimension to capture the spatial location and dimensions of objects in the physical world. However, the increased complexity of this task, combined with the large volume of data to be processed, often necessitates more complex model architectures, leading to higher computational demands. Our experiments (Section 2.2) reveal that the inference latency of a 3D detection model can be up to 41 times longer than that of a 2D model on the same hardware, underscoring the substantial computational overhead of 3D object detection. Moreover, edge computing platforms typically have limited computational resources. For instance, commonly used devices like the TX2 [6] have significantly fewer CUDA cores—17×fewer than desktop-class GPUs such as the RTX 2080Ti [7], and 31×fewer than high-end GPUs like the Tesla A100 [8]. As a result, running 3D detection models on edge devices for near real-time processing remains a significant challenge. While offloading compute-intensive 3D object detection to the cloud alleviates the processing burden on local devices, the end-to-end latency remains impractical due to the heavy communication overhead, which constitutes the primary bottleneck (Section 2.2). Although compression techniques can reduce point cloud size, their computational overhead still hampers real-time streaming, as discussed in Section 2.2.

Traditional approaches either rely entirely on edge computing, which struggles with limited resources, or offload computations to the cloud, introducing excessive communication delays. To address this trade-off, we propose Moby, a novel framework that significantly reduces the need for heavy 3D detection models on edge devices while maintaining high accuracy and efficiency. At the core of Moby

* Corresponding author.
E-mail addresses: jingzongli@hsu.edu.hk (J. Li), 1155141377@link.cuhk.edu.hk (Y.H. Cai), liulb@zgclab.edu.cn (L. Liu), yumao7-c@my.cityu.edu.hk (Y. Mao), jason.xue@mbzuai.ac.ae (C.J. Xue), hongxu@cuhk.edu.hk (H. Xu).

https://doi.org/10.1016/j.comnet.2025.111382
Received 24 July 2024; Received in revised form 16 March 2025; Accepted 13 May 2025
Available online 30 May 2025

Fig. 1. 2D and 3D object detection.

is a lightweight 2D-to-3D transformation technique that enables edge devices to infer 3D bounding boxes using only 2D detection results and point cloud data. This approach minimizes reliance on computationally intensive 3D models, allowing most processing to occur locally while selectively offloading a small subset of frames to the cloud for DNN-based 3D detection. By reducing both computational and bandwidth overhead, Moby makes real-time 3D object detection more feasible for edge computing environments.

However, designing such a system introduces two key challenges:

- **Accurate and Efficient Transformation**: How can Moby effectively transform 2D bounding boxes into 3D ones while maximizing the latency benefits of 2D detection?
- **Error Accumulation and Offloading Strategy**: As transformation errors accumulate over time, how can Moby monitor accuracy degradation and determine the optimal timing for offloading frames to the cloud?

To address these challenges, Moby adopts a holistic system design that integrates several key components. First, a lightweight instance segmentation model runs on the edge device to extract both 2D bounding boxes and segmentation masks. Then, a *2D-to-3D transformation* module leverages these segmentation masks and point cloud data to generate accurate 3D bounding boxes, while avoiding the computational overhead of full 3D detection models. To further enhance accuracy, Moby incorporates a *tracking-based association* mechanism that utilizes prior 3D detection results from the cloud to establish object correspondences across frames, enhancing bounding box estimation. Finally, to mitigate cumulative transformation errors over time, Moby employs a *frame offloading scheduler* that dynamically determines when to offload new anchor frames for cloud-based 3D object detection, ensuring continued accuracy while balancing computational and bandwidth constraints.

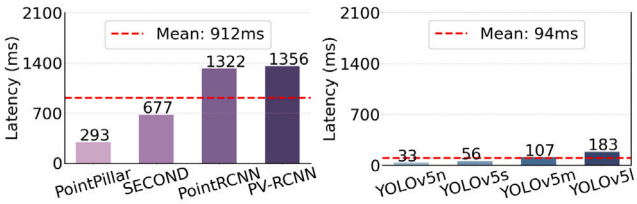
We implement Moby on an Jetson TX2 and a server equipped with RTX 2080Ti GPU. Through evaluation on the KITTI dataset [9], our results demonstrate that Moby outperforms existing approaches by achieving a remarkable latency reduction of up to 91.9%, while maintaining a minimal accuracy loss. Notably, on the Jetson TX2, Moby achieves a real-time processing capability of 10 FPS, which aligns with the scanning frequency of KITTI's LiDAR. Furthermore, our observations reveal that Moby reduces power consumption by 75.7% and memory usage by 48.1%. These reductions are crucial for constrained edge devices that need to save resources for other critical 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 suffer from significant 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.

Table 1
Point cloud-based models we measure. PointPillar [10] and SECOND [11] group points into vertical columns or voxels and apply convolution over these structures; PointRCNN [12] extracts features directly from the points with permutation invariant learning; PV-RCNN [13] combines the point and voxel features. The first two models are one-stage that directly perform object classification and bounding box regression; while the last two models are two-stage with pre-generated region proposals.

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



(a) 3D models (b) 2D models

Fig. 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 [4,5]. 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 [4]. The first approach uses the point cloud data from LiDAR or Radar sensors to generate 3D bounding boxes [10–15]. The second one is image-based [16–20] 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 [21–26] 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 hardwares and deep neural networks [4,5,27–30], performing 3D detection is still a daunting task for resource-constrained edge devices [31]. 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 [6], 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 [9], the most popular one in the field of autonomous driving.

Table 2
Statistics of four cellular bandwidth traces.

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

Table 3
The latency and compression ratio of common compression algorithms. All algorithms are run on the Jetson TX2.

Algorithm	gzip	zlib	bzip2	lzma	draco
Compression Time (ms)	134	238	1007	1179	108
Compression Ratio	1.57	1.57	1.75	1.83	1.6

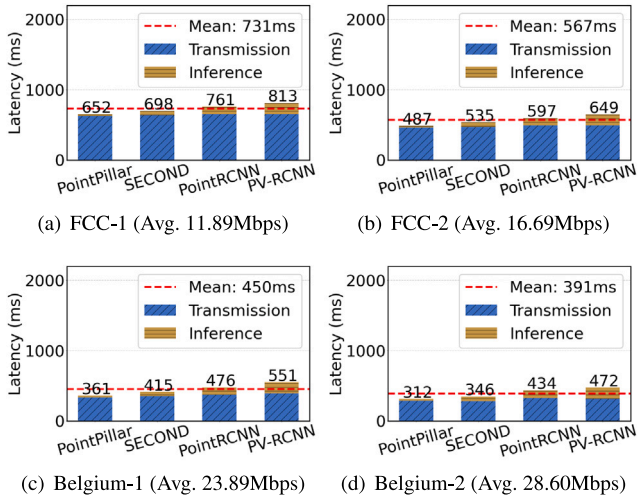


Fig. 3. The end-to-end latency of cloud-ony inference on four real-world network traces.

Edge-only inference. The results, shown in Fig. 2(a), indicate that: (i) The average inference latency on the device reaches 912 ms 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 293 ms and 677 ms, 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 pretrained YOLOv5 [32] variants.[1] The input images also come from KITTI [9], captured by its car camera with a resolution of 1242 × 375. Results in Fig. 2(b) shows that the on-device inference latency of 2D models is much shorter. Notably, YOLOv5n only takes 33 ms 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 [33] and Belgium [34], which are summarized in Table 2. The point cloud data is uploaded to the server using one TCP connection, and Linux TC [35] 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 (~10–30Mbps) [36]. It is likely that the real-world performance of cloud-only inference could be worse than what we report here in certain environments [2].

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 391 ms. This is much better than edge-only latency with 912 ms 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 that of Belgium-2. In addition, we also consider compressing the point cloud data before transmission. We test the performance of four representative lossless compression algorithms: gzip [37], zlib [38], bz2 [39] and lzma [40] on TX2, whose CPU runs at 2 GHz. We use the implementation in Python standard library to run these algorithms on 50 point cloud files and report the average. We also test the performance of the widely-used point cloud compression tool, draco [41]. The results are presented in Table 3. We observe that the compression time is all above 100 ms, and a larger compression ratio generally results in a longer time. With the algorithm gzip, it can reduce the end-to-end latency of the slowest FCC-1 by 78 ms. However, for higher bandwidth traces, compression does not contribute to latency reduction much and can even jeopardize it, due to the non-negligible compression time. 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. The central tradeoff we need to balance here is between accuracy and latency. The more 2D detection we use, the more latency gain we can reap, but the accuracy may degrade over time due to the accumulation of errors introduced during the transformation process.

We then introduce the system design of Moby in detail.

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

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

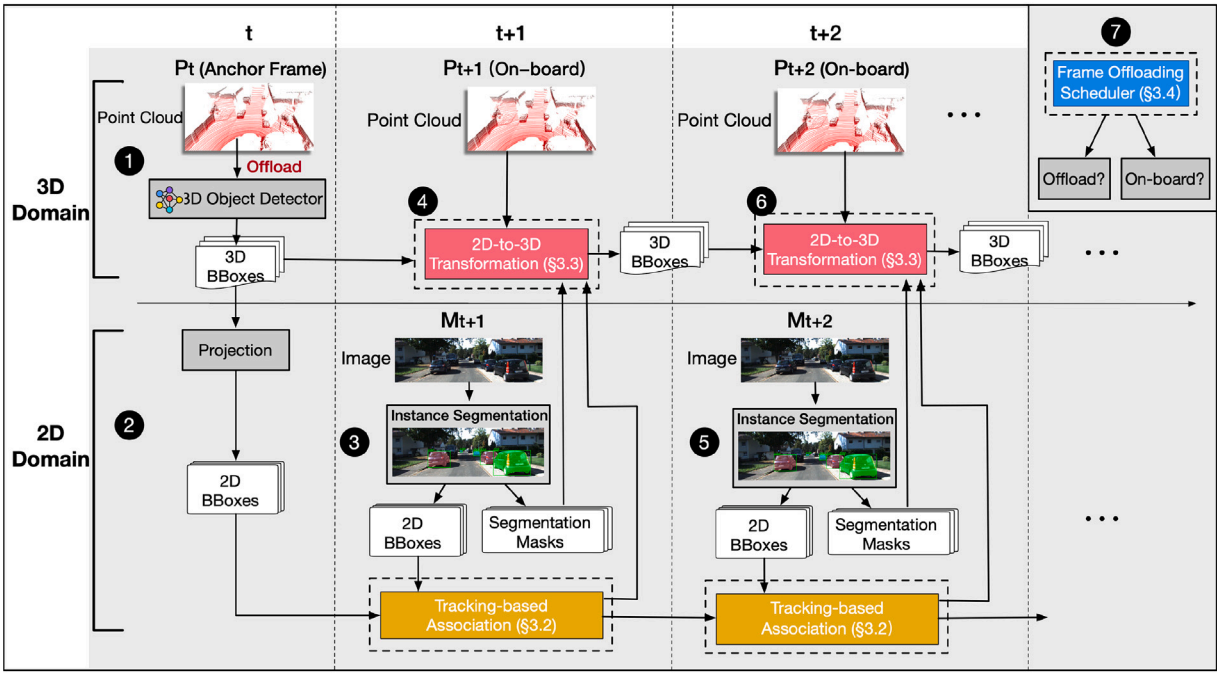


Fig. 4. Moby system overview.

the 3D bounding boxes to initiate the process. 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 the following 2D-to-3D transformation. Fig. 5 shows an overview of tracking-based association.

On-device 2D Inference. To obtain 2D bounding boxes, Moby either projects 3D results to 2D if the current LiDAR frame is selected as the anchor frame, or runs a 2D model directly. In the latter case, we adopt instance segmentation models as they output 2D bounding boxes and segmentation masks simultaneously, with bounding boxes used in the following tracking and segmentation masks needed for the 2D-to-3D transformation in Section 3.3.

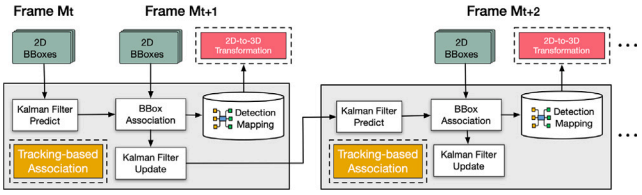


Fig. 5. Overview of tracking-based association.

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 [42], 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 [43] and Intersection-over-Union (IoU) criterion, with association rejected if the IoU is below a threshold. The Kalman filter then updates trajectory predictions using matched detections.

Specifically, the Kalman Filter has an initial state vector for box i in the 2D results of frame M_t: $S_{t,i} = (u, v, s, r)$, including location of the object center in 2D image (u, v), area of the bounding box s, and the aspect ratio r. To predict the new state $S'_{t+1,i}$, we use a constant velocity model independent of camera ego-motion to approximate the inter-frame displacement of objects. After Kalman Filter prediction, $S'_{t+1,i}$ will be updated to a more accurate state $S_{t+1,i}$ by taking in the associated bounding box as the observation.

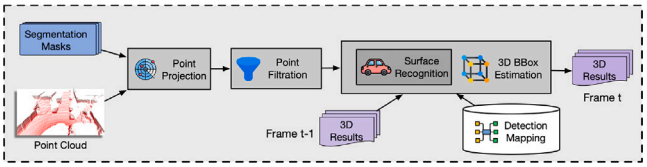


Fig. 6. Overview of 2D-to-3D transformation.

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.

Point Projection. As discussed in Section 2.3, the 3D and 2D frames captured at the same time have closely related semantic information, and the camera and LiDAR normally have fixed positions once they are installed, which makes it possible to leverage the knowledge in the 2D domain to extract meaningful semantics in the 3D domain.

To transfer the semantic information contained in the segmentation masks to 3D, we 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. We again take the KITTI dataset [9] as an example. The transformation is as follows:

$$B = T_t * T_r * A. \tag{1}$$

Here A is points in 3D coordinate, where each point is a vector $[x, y, z]$. They are first projected to a reference camera by the matrix T_r, and then projected to the target camera by T_t to obtain the corresponding pixels B. 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. Specifically, the output of instance segmentation are stacked masks that identify each object in 0 and 1 values. That is, by squeezing the stacked masks along the channel dimension, we can assign each pixel a number that indicates which object it belongs to. Once the LiDAR points are projected to the image plane, the number is concatenated to the point in a form of $[x, y, z, n]$. 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 as parts of vehicles.

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 [5].

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

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

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 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 [5]. (see Fig. 7).

To tackle the above challenges, we implement a robust and efficient 3D bounding box estimation technique that integrates previous detection results with the spatial distribution of the available point cloud data. Specifically, after point filtration, we use a combination of RANSAC and a bounding box fitting algorithm to iteratively adjust the dimensions and orientation of the 3D bounding box to best encompass the available points. Our method will leverage previous 3D detection result of the same object as reference to provide better estimation.

First, we need to determine the object size. For each point cluster that has been associated with an object from the previous LiDAR frame by the association module, Moby can easily 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 [44]

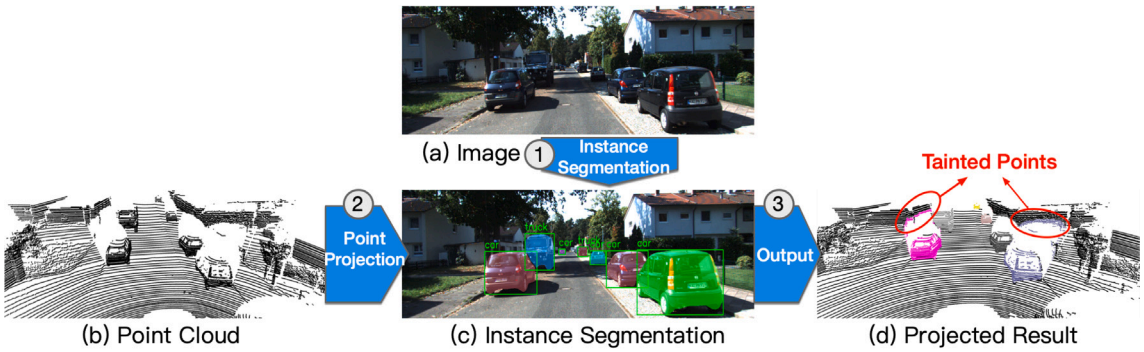


Fig. 7. Example of point projection on the KITTI dataset.

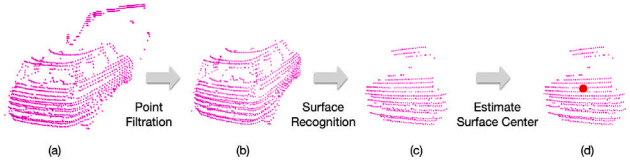


Fig. 8. Key steps of point filtration and 3D bounding box estimation for a point cluster.

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:

$$h^t = \begin{cases} v^t, & \text{if angle between } h^{t-1} \text{ and } v^t \text{ is less than } \xi, \\ -v^t, & \text{if angle between } h^{t-1} \text{ and } v^t \text{ is greater than} \\ & \pi - \xi \ (v^t \text{ in the opposite direction}). \end{cases} \tag{2}$$

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

$$Q = \begin{cases} [a, b, c] + [0.5 * l * \cos\theta, 0.5 * l * \sin\theta, 0], \text{if angle bet-} \\ \text{ween } h^{t-1} \text{ and } v^t \text{ is less than } \xi \text{ or greater than } \pi - \xi, \\ [a, b, c] + [0.5 * w * \cos\theta, 0.5 * w * \sin\theta, 0], \text{otherwise.} \end{cases} \tag{3}$$

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. (3), and the one that fits more points inside is the final output as shown in Fig. 10.

3.4. Frame offloading scheduler

Unlike existing systems [1,45] that send every frame to the server, we design a frame offloading scheduler that only offloads necessary frames whose results are required by the 2D-to-3D transformation, thus drastically reducing bandwidth consumption. As mentioned earlier, 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. Thus from time to time, Moby needs to offload a new anchor frame to the server for 3D detection to assist the subsequent transformation and maintain overall accuracy. To efficiently schedule frame offloading, our design must satisfy two requirements: (1) introduce minimum overhead, and (2) effectively monitor error accumulation.

For this purpose, a simple yet efficient solution is to send a test LiDAR frame to the cloud every N_T frames. Detection on the test frames happens in parallel with Moby's on-device processing. We deem the 3D detection results on the server 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 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 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.

[2] Theoretically, 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 the RANSAC algorithm.

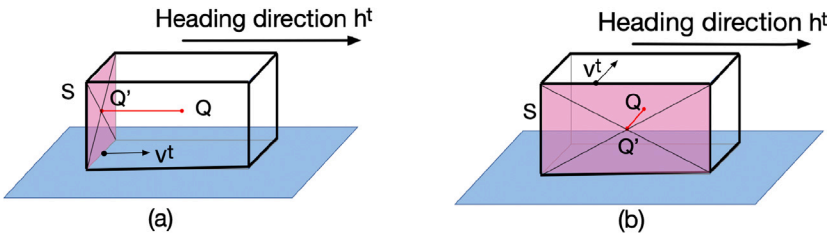


Fig. 9. Two possible situations in bounding box estimation.

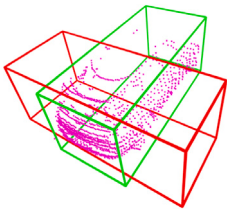


Fig. 10. Compare the number of points inside both situations to determine the correct box.

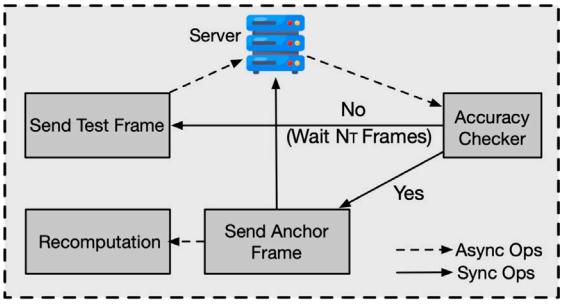


Fig. 11. Overview of frame offloading scheduler.

3.5. Network-aware controller

To address the challenges posed by network deterioration, we build an optional module named as the *network-aware controller*. Its purpose is to detect bandwidth drop in network conditions. Given the volatile nature of cellular networks, severe drops in network quality are not uncommon, a fact that has been discovered in several prior studies [46–50]. In such case, relying solely on running anchor frames on the server may not be a sensible solution, as the increased transmission time would significantly impact latency. Consequently, in the Moby framework, a decision can be made to execute anchor frames locally on the edge device only if the latency is considerably lower compared to offloading. The decision-making process consists of two steps:

Step 1: Available bandwidth prediction. For every frame received on the server, the network-aware controller performs continuous measurements of the total time taken to receive the frame ($T_{received}$) as well as the size of the frame ($S_{received}$). The observed throughput of the frame can be calculated by $S_{received}/T_{received}$. Instead of relying on a single past value, we compute the harmonic mean of the past five observed throughputs to predict the available bandwidth $B_{available}$. This prediction is then transmitted back to the edge device to assist in the decision-making process for offloading.

Step 2: Frame delivery time estimation. On the edge side, before sending the anchor frame, Moby leverages the predicted available bandwidth to determine whether or not to offload the anchor frame. The rationale behind this decision is intuitive: if the network bandwidth experiences a significant drop, it is not practical to offload the frame to the server and endure the resulting increased delivery time. To make this decision, we consider the size of the anchor frame (S_{anchor}) and the available bandwidth ($B_{available}$) and apply the following criteria:

$$Offloading = \begin{cases} True, & \text{if } S_{anchor}/B_{available} < \alpha * T_I, \\ False, & \text{otherwise,} \end{cases} \tag{4}$$

where T_I represents the on-board inference time of the 3D object detection model, which is pre-measured. α is a hyperparameter to adjust the preference. Essentially, we estimate the delivery time of the current anchor frame by using S_{anchor} and $B_{available}$. We then compare this estimated delivery time with the on-board inference time to determine the optimal approach for running the anchor frame. It is important to note that the network-aware controller has no impact on the on-board processing and does not introduce any additional latency.

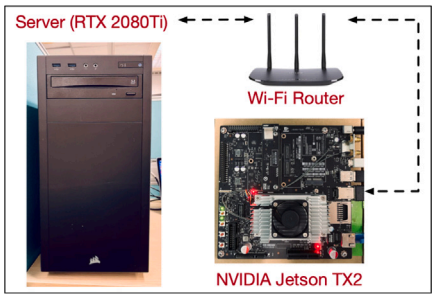


Fig. 12. Our evaluation hardware platform.

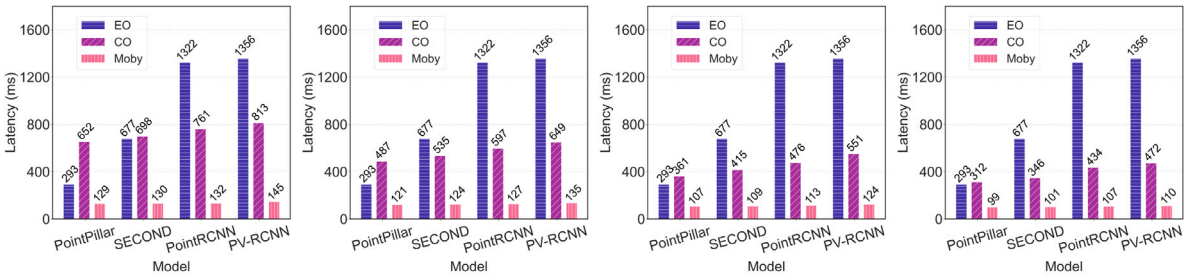
4. Implementation

We implement Moby in an edge-server architecture with around ~4K lines of Python code, as shown in Fig. 12. All major components of Moby are running on-board. Only the 3D object detectors are deployed on the server to serve the edge's inference requests.

Hardware Setup. We use NVIDIA Jetson TX2 [6] as the edge device, which is a commodity mobile module for edge computing. TX2 is equipped with one 256-core NVIDIA Pascal GPU, one Dual-Core Denver CPU, one Quad-core ARM A57 CPU, and loaded with 8 GB LPDDR4 memory. The server program runs on a desktop with Ubuntu 20.04, which has one NVIDIA GeForce RTX 2080Ti GPU and Intel Core i7-9700K CPU. The edge device and the desktop server are physically connected to a router through Ethernet cables and located in the same local network.

Edge Side. We implement the edge side of Moby on the Jetson TX2 with installed JetPack SDK 4.6.2. To maximize its performance, we turn on Max-N power mode [51], which fully uses all 6 CPU cores and operates GPU at the highest clock frequency. The edge device communicates with the server via a TCP connection. We use Linux TC [35] to control the uploading bandwidth by taking the real-world network traces described in Table 2 as input.

Server Side. We deployed OpenPCDet [52] on the server as the inference engine, which is a general PyTorch-based framework for 3D object



(a) FCC-1 (Avg. 11.89Mbps) (b) FCC-2 (Avg. 16.69Mbps) (c) Belgium-1 (Avg. 23.89Mbps) (d) Belgium-2 (Avg. 28.60Mbps)

Fig. 13. The latency comparison of Moby, edge-only, and cloud-only deployment approaches.

detection and currently supports a variety of cutting-edge 3D detection models.

Parameter Settings. There are a few parameters in Moby to be determined. 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. In the 3D bounding box estimation, we choose 30 as the default iteration number in RANSAC [44]. As for the offloading scheduler, the accuracy threshold Q_T is set to 0.7. For the gap of test frame N_T, A high value leads to delayed 3D results from server and accumulated local transformation errors, while a low value increases bandwidth consumption due to frequent offloading. Based on empirical evaluation, we found $N_T = 4$ offers a suitable balance. We will investigate the sensitivity of key parameters and their impact on the performance in Section 5.4.

5. Evaluation

We now evaluate Moby using a series of testbed experiments. Our evaluation answers the following questions:

- How well does Moby work compared to existing approaches in terms of latency and accuracy?
- How much does each key design choice contribute to the overall performance gain?
- How sensitive is Moby to key parameters and 2D models?
- How efficiently does Moby utilize the edge resources in terms of energy and memory consumption?

5.1. Experimental setup

Datasets. All experiments are conducted on the KITTI dataset [9], one of the most popular real-world autonomous driving benchmarks. It consists of synced LiDAR scans and front-view camera images with a frequency of 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 described in the Table 2 to emulate the network condition between edge and server.

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 [1,47,53–55].

Models. We adopt YOLOv5n from the YOLOv5 codebase [32] as Moby's default instance segmentation model, primarily because it is one of the most popular and widely-used 2D models. 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, as we will demonstrate this property in Section 5.2 and Section 5.4.

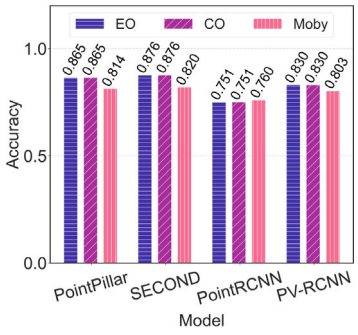


Fig. 14. The 3D detection accuracy of Moby and compared schemes.

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.

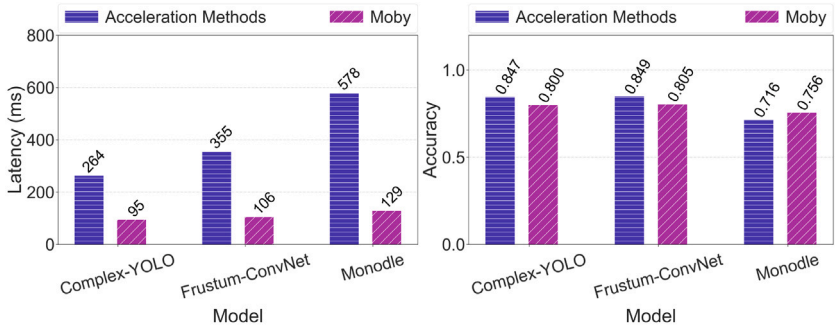
5.2.1. Deployment approaches

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 99 ms 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. 14 reports overall accuracy for all schemes. Note that as EO and CO both



(a) Latency (b) Accuracy

Fig. 15. Comparison of Moby and alternative acceleration methods.

Table 4
Impact of each design component.

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

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 between 0.027 to 0.056 due 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* [14]: 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 [14].
- *Frustum-ConvNet* [21]: A fusion-based approach that utilizes 2D region proposals to narrow down the 3D space for acceleration.
- *Monodle* [56]: 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. 15(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 [5,20]. We also tried another two image-based methods, Deep3DBox [18] and Pseudo-LiDAR++ [20]. However, they take 2834 ms and 5889 ms, respectively, which are too slow to execute on edge devices.

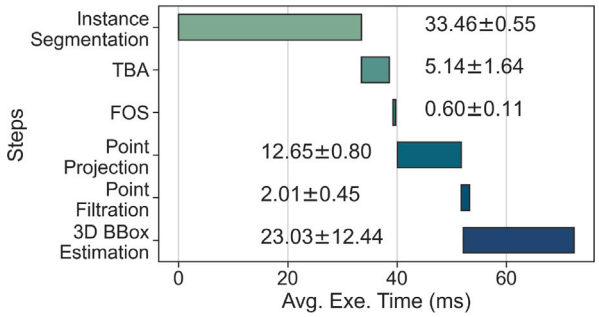


Fig. 16. The avg. execution time of key steps over 300 runs.

Table 5
Bandwidth savings of Moby.

	PointPillar	SECOND	PointRCNN	PV-RCNN
Bandwidth Savings (%)	75.59	73.80	74.64	79.42

5.3. Ablation study

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).

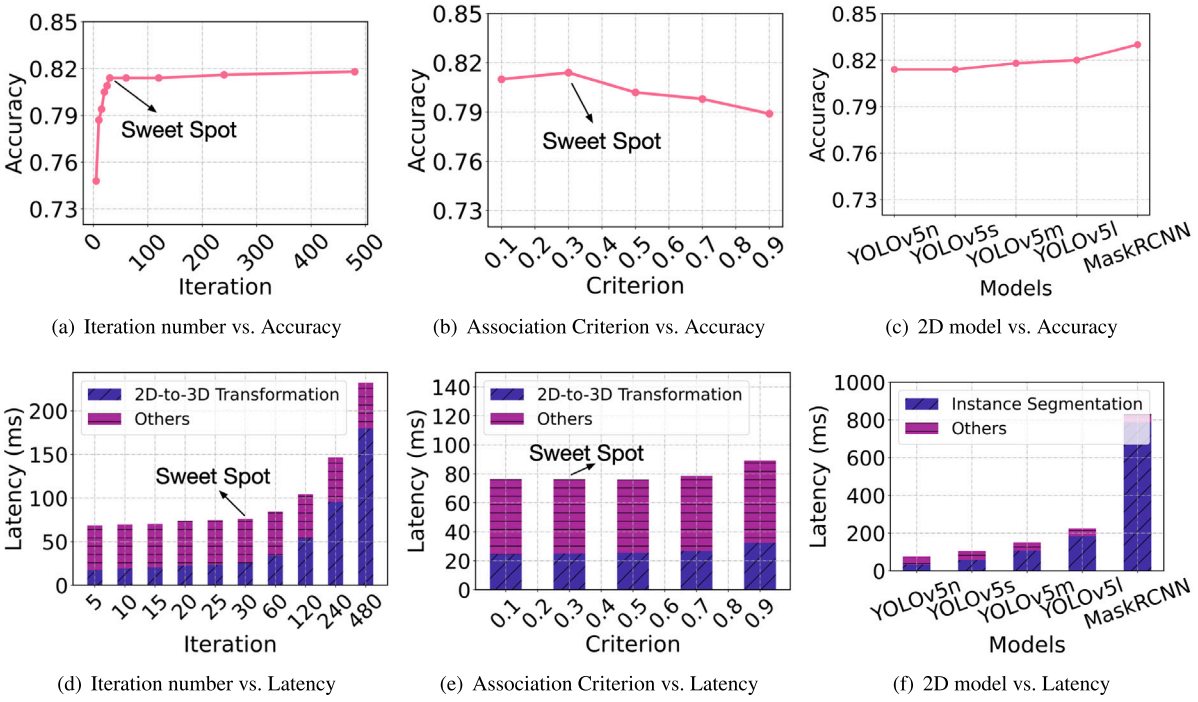
Table 4 shows the results. We observe that TRS can already achieve decent accuracy, proving the effectiveness of 2D-to-3D transformation. With FOS, accuracy is improved as some frames are offloaded to server for inference. Offloading increases the end-to-end latency while on-board latency remains unchanged. 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. 16. 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. For TBA and FOS, they only take 5.14 ms and 0.60 ms respectively, which are negligible compared to the total latency. Notably, Point filtration has only 2.01 ms, proving the efficiency of Algorithm 1.

Bandwidth Savings. Bandwidth efficiency are crucial for data analytics pipelines that involve transmitting data over wireless networks. Since Moby only needs to send a subset of frames to the server for

Table 6
Latency improvement with network-aware controller.

	PointPillar		SECOND		PointRCNN		PV-RCNN	
	w/o	w/	w/o	w/	w/o	w/	w/o	w/
Latency (ms)	205.06	97.96	208.86	136.36	213.06	200.86	218.36	204.26



(a) Iteration number vs. Accuracy (b) Association Criterion vs. Accuracy (c) 2D model vs. Accuracy

(d) Iteration number vs. Latency (e) Association Criterion vs. Latency (f) 2D model vs. Latency

Fig. 17. The impact of key parameters and instance segmentation models on the performance of Moby. Note that here we only report the on-board latency for clearer comparison.

processing, our approach can significantly reduce the amount of transmitted point cloud data. To quantify bandwidth savings, we record the total size of transmitted frames ($SIZE_t$) and the total size of all frames ($SIZE_a$). The bandwidth savings are then calculated as ($SIZE_a$-$SIZE_t$)/$SIZE_a$. Note that both anchor frames and test frames are included in the count of transmitted frames. The results are summarized in Table 5. Notably, we observe bandwidth savings exceeding 70% across all four models evaluated in our experiments. The savings stem from Moby's ability to generate 3D bounding boxes with 2D-to-3D transformation locally. These results also underscore the effectiveness of frame offloading scheduler in reducing the data transmission size, thereby enhancing bandwidth efficiency in point cloud analytics pipelines.

Effectiveness of Network-aware Controller. We conduct a separate investigation to evaluate the effectiveness of the network-aware controller, which is an optional module in our system. Table 6 presents the improvements achieved by integrating the network-aware controller into Moby compared to the version without it. In this evaluation, we utilize a low-bandwidth network trace with an average of 5Mbps and focus on reporting the end-to-end latency, as the network-aware controller only impacts the running time of anchor frames without affecting accuracy. Our findings demonstrate that the network-aware controller can effectively reduce latency, simply because it mitigates long offloading time in the presence of slow networks. Specifically, we observed a latency improvement of 52.2% for PointPillar. The extent of latency reduction diminishes for larger models like PV-RCNN under the same network conditions, primarily due to the latency gains achieved through server-side inference.

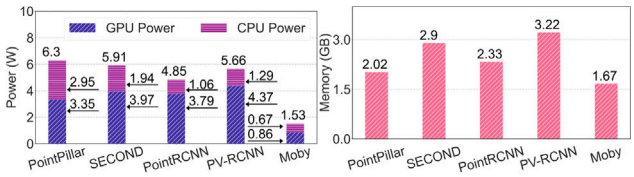
5.4. Sensitivity analysis

We now conduct microbenchmarking experiments to assess Moby's performance under varying key parameters, as well as evaluate its sensitivity to 2D models.

Sensitivity to the Iteration Number in RANSAC. The iteration number in RANSAC is a key parameter in the 2D-to-3D transformation process, specifically in performing plane fitting to identify surfaces from point clusters. To evaluate its impact on Moby, we conduct experiments and analyze the resulting accuracy and on-board latency, as depicted in Figs. 17(a) and 17(d). Our analysis reveals that a smaller iteration number leads to reduced latency but may result in premature or inaccurate surface recognition, consequently affecting the estimation of the heading angle and object center. While a larger iteration number makes the plane fitting more robust and meanwhile induce more computation overhead. We experimentally find that 30 is an appropriate iteration number to balance between accuracy and computation cost.

Sensitivity to Association Criterion. Figs. 17(b) and 17(e) test the impact of association criterion, i.e., the IoU threshold used to associate two bounding boxes in 2D space as described in . A larger criterion makes it harder to associate bounding boxes and cannot effectively utilize previous 3D results. We empirically choose 0.3 as the association criterion, as the accuracy gain diminishes after it is greater than 0.3.

Sensitivity to 2D Models. To verify the impact of 2D models, we test Moby with other instance segmentation models from YOLOv5 [32] codebase, including YOLOv5s, YOLOv5 m, and YOLOv5l. We also evaluate Moby using MaskRCNN [57], a classic instance segmentation model that employs a distinct feature extractor and network architecture, distinguishing it from YOLOv5. The results are shown in Figs.



(a) Energy consumption (b) Memory footprint

Fig. 18. System efficiency.

17(c) and 17(f). We observe that: (i) Moby's accuracy is generally stable when using various instance segmentation models, with a slight improvement when using a larger model; (ii) the on-board latency grows substantially with the increase of model size. The results indicate that a small instance segmentation model is good enough to achieve a decent performance. Hence, we adopt YOLOv5n as Moby's default 2D model to reap the benefit of low latency.

5.5. System efficiency

Finally, as efficient resource utilization is crucial for applications running on constrained edge devices, we evaluate the system efficiency of Moby.

Energy Consumption. We monitored the instantaneous energy consumption of Jetson TX2 using the built-in Tegrastats Utility [58] for a duration of 2 min, with a sampling rate of 10 Hz. The average consumption is reported in Fig. 18(a). Our proposed system, Moby, demonstrates the lowest energy consumption and significantly outperforms the baseline methods. For instance, Moby achieves a power consumption that is only 24.2% of PointPillar. Furthermore, Moby reduces GPU power and CPU power by 74.3% and 77.3%, respectively. Overall, Moby achieves an average power savings of 73% against the four 3D object detection models.

Memory Footprint. We conducted measurements to assess the memory footprint of Moby, and the results are presented in Fig. 18(b). The overall memory reduction achieved by Moby ranges from 17.3% to 48.1%. This reduction is anticipated since Moby only requires loading a smaller 2D detection model into memory, compared to the larger memory requirements of 3D object detection models. This advantage of Moby is particularly noteworthy when deploying on wimpy edge devices, such as Jetson Nano [59], which possess only 4 GB of memory shared by both the CPU and GPU.

6. Related work

3D Object Detection. 3D detectors can be divided into three categories based on input modality. The first and the most common type of 3D detection models use point cloud as input. The key challenge for these models is how to extract features from the sparse point cloud. Voxel-based methods like SECOND [11] first group points into voxels and extract features on voxel level; PointRCNN [12] directly takes the raw point cloud as input and computes a pointwise feature vector. Due to the high cost of LiDAR sensor, image-based 3D detection models have been proposed with only RGB images as input. To compensate for the lack of depth information, they generally require complex DNN structures to extract useful information. Pseudo-LiDAR++ [20] synthesizes LiDAR data based on the image before feeding it to a point cloud-based model. CaDNN [16] leverages a categorical depth distribution to aid 3d detection. The third category, fusion-based methods, leverages both point cloud and image inputs to extract information from both modalities [21–24,26,60]. For example, Rcfusion [60] employs an end-to-end deep learning model that directly extracts features from images and radar points to generate 3D detection results. This approach involves complex DNN architectures, necessitating the use of high-performance GPUs. In contrast, Moby avoids heavy 3D object detectors by using a lightweight 2D object detection model combined with a 2D-to-3D transformation, enabling efficient and accurate 3D bounding box estimation without the need for compute-intensive models. The key distinction between Moby and fusion-based methods lies in the fact that Moby is a systematic framework rather than a specialized DNN model. Moby is designed to be model-agnostic, allowing for the integration of any 3D object detection model into its framework.

On-device Inference Acceleration. Deploying DNN models on edge devices with constrained hardware resources poses significant challenges due to their high computational demand. Considerable endeavors have been dedicated to expediting DNN inference on edge devices, encompassing techniques like model compression, hardware-based approaches, and software-based approaches. Model compression techniques reduce the model size and accelerate computation by network pruning [61–63], quantization [64,65], and knowledge distillation [66,67]. In addition to model compression, various work has explored hardware-based approaches that design specialized accelerators for hardwares, such as FPGA [68,69] or ASIC [70,71]. The last line of work is software-based approaches [2,55,72–74]. For example, TorchSparse [72] proposes a grouping algorithm to smartly batch workload to reduce irregular computation for sparse point cloud, while other approaches leverage task-specific optimizations [2,55,73–76]. Moby belongs to the software-based category and is orthogonal to the existing work. It can be combined with existing acceleration methods, such as model compression and hardware acceleration, to achieve even better performance in terms of latency.

Edge–Cloud Collaboration. Besides on-device inference, the emergence of edge devices has also sparked research on edge–cloud collaboration. Filter-based methods, such as Elf [77], Reducto [78], and Geryon [79], focus on addressing the challenge of handling large data volumes during transmission. These methods employ different strategies to reduce the amount of data that needs to be transmitted. For instance, Elf proposes a region proposal prediction algorithm that partitions frames into smaller slices, enabling the offloading of only partial inference tasks to multiple servers for parallel processing. On the other hand, Reducto utilizes an on-camera filtering technique to identify and filter out frames that lack relevant information for the given query. Partition-based methods [80–82] explore partitioning the DNN models across edge and cloud. They automatically divide a DNN model into two partitions and balance the computation-network tradeoff on the edge device; SPINN [83] further incorporates early-exit into DNN partitioning to allow the inference to exit early based on the input complexity. Feedback-based methods [1,55,84] rely on the feedback from server to assist local processing on the edge. For example, Glimpse [55] periodically sends trigger frames to the server to obtain the object recognition hints to assist local tracking; DDS [1] sends the low-quality video at first and determines where to re-send with high quality driven by the server-side DNN feedback.

Compared to prior research that primarily focuses on the 2D domain, our work delves into the realm of edge–cloud collaboration for the computationally demanding task of 3D object detection. This particular task presents more formidable challenges when it comes to deploying on edge devices.

7. Discussion and limitations

Constant Velocity in Kalman Filter. Although the constant velocity model in the Kalman Filter provides a good trade-off for designing the tracking module, incorporating an adaptive mechanism to adjust the process noise covariance based on the ego-vehicle's motion patterns can enhance performance. Specifically, onboard sensors can monitor the ego-vehicle's acceleration and angular velocity. When significant changes are detected, such as sharp turns or sudden stops, the process

noise covariance is increased to account for the heightened uncertainty in object motion. This adaptive adjustment helps maintain tracking accuracy in dynamic and variable motion conditions.

Occlusion in Images. Although Moby brings significant latency gain to running 3D object detection models, we notice that the accuracy performance of Moby still has room to improve. One problem that haunts existing image-based 3D object detection and also Moby is the occlusion in 2D images [4,5]. As we know, an important characteristic of driving environment is occlusion or truncation present in crowded scenes where vehicles can block the view of other agents and themselves [4]. Since 2D-to-3D transformation in Moby is based on the visual perception of images, its accuracy is likely to be affected by the occlusion effect. A possible solution is to integrate an occlusion reasoning module into Moby to handle object missing. It is a promising direction for Moby to advance its performance, and we leave it as the future work.

Generalization to Other Tasks. While Moby is mainly evaluated on the 3D detection task of cars, it can be generalized to other 3D detection tasks, such as pedestrians and cyclists. For these tasks, minor design modification is needed in the 2D-to-3D transformation module to estimate the bounding box accurately. For example, because pedestrians and cyclists are smaller than cars, their object centers can be simply calculated by averaging the value of all points in the point cluster. Other task-specific design for the pipeline is also needed to maximize its benefit. We expect Moby achieves similar performance gains in these scenarios, as it can be applied similarly to leverage 2D models for 3D object detection to accelerate inference.

8. Conclusion

We have presented Moby, an innovative framework designed for efficient and accurate 3D object detection on constrained edge devices. Moby introduces a novel 2D-to-3D transformation technique that significantly reduces the computational demands of 3D object detection. Moreover, we design an offloading scheduler that judiciously offloads frames to the cloud for inference. This scheduler not only ensures accuracy but also effectively orchestrates the computational resources of the edge and the cloud in a collaborative manner. Extensive evaluation using a prototype system running on Jetson TX2 shows that Moby reduces end-to-end latency by up to 91.9% with minimal accuracy drop. It achieves a real-time processing speed of 10 FPS, matching the scanning frequency of LiDAR in the KITTI dataset. Notably, Moby also exhibits energy efficiency, reducing power consumption and memory footprint by an average of 72.8% and 34.1%, respectively.

CRediT authorship contribution statement

Jingzong Li: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Yik Hong Cai:** Software, Data curation. **Libin Liu:** Writing – review & editing. **Yu Mao:** Writing – review & editing, Writing – original draft. **Chun Jason Xue:** Supervision. **Hong Xu:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by a Faculty Development Grant from the Hang Seng University of Hong Kong (SDSC-SRG066), and funding from Natural Science Foundation of Shandong Province, China (ZR2022QF070).

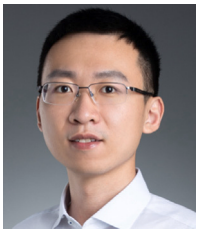
Data availability

Data will be made available on request.

References

[1] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, J. Jiang, Server-driven video streaming for deep learning inference, in: Proc. ACM SIGCOMM, 2020.
[2] J. Yi, S. Choi, Y. Lee, EagleEye: Wearable camera-based person identification in crowded urban spaces, in: Proc. ACM MobiCom, 2020.
[3] C. Dong, C.C. Loy, K. He, X. Tang, Image super-resolution using deep convolutional networks, Proc. IEEE TPAMI 38 (2) (2015) 295–307.
[4] E. Arnold, O.Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, A. Mouzakitis, A survey on 3D object detection methods for autonomous driving applications, IEEE Trans. Intell. Transp. Syst. 20 (10) (2019) 3782–3795.
[5] R. Qian, X. Lai, X. Li, 3D object detection for autonomous driving: A survey, Pattern Recognit. (2022).
[6] NVIDIA Jetson TX2 developer kit, 2023, https://developer.nvidia.com/embedded/jetson-tx2.
[7] Jetson TX2 GPU vs GeForce RTX 2080 Ti, 2023, https://technical.city/en/video/GeForce-RTX-2080-Ti-vs-Jetson-TX2-GPU.
[8] Jetson TX2 GPU vs Tesla A100, 2023, https://technical.city/en/video/Tesla-A100-vs-Jetson-TX2-GPU.
[9] A. Geiger, P. Lenz, R. Urtasun, Are we ready for autonomous driving? The KITTI vision benchmark suite, in: Proc. IEEE/CVF CVPR, 2012.
[10] A.H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, O. Beijbom, Pointpillars: Fast encoders for object detection from point clouds, in: Proc. IEEE/CVF CVPR, 2019.
[11] Y. Yan, Y. Mao, B. Li, Second: Sparsely embedded convolutional detection, Sensors 18 (10) (2018) 3337.
[12] S. Shi, X. Wang, H. Li, Pointrcnn: 3d object proposal generation and detection from point cloud, in: Proc. IEEE/CVF CVPR, 2019.
[13] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, H. Li, Pv-rcnn: Point-voxel feature set abstraction for 3d object detection, in: Proc. IEEE/CVF CVPR, 2020.
[14] M. Simony, S. Milzy, K. Amendey, H.-M. Gross, 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, 2018.
[15] Y. Zhou, O. Tuzel, Voxelnet: End-to-end learning for point cloud based 3D object detection, in: Proc. IEEE/CVF CVPR, 2018.
[16] C. Reading, A. Harakeh, J. Chae, S.L. Waslander, Categorical depth distribution network for monocular 3d object detection, in: Proc. IEEE/CVF CVPR, 2021.
[17] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, R. Urtasun, Monocular 3D object detection for autonomous driving, in: Proc. IEEE/CVF CVPR, 2016.
[18] A. Mousavian, D. Anguelov, J. Flynn, J. Kosecka, 3D bounding box estimation using deep learning and geometry, in: Proc. IEEE/CVF CVPR, 2017.
[19] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, K.Q. Weinberger, Pseudo-lidar 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, 2019.
[20] Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, K.Q. Weinberger, Pseudo-lidar++: Accurate depth for 3D object detection in autonomous driving, 2019, arXiv preprint arXiv:1906.06310.
[21] Z. Wang, K. Jia, Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3D object detection, in: Proc. IEEE/RSJ IROS, 2019.
[22] C.R. Qi, W. Liu, C. Wu, H. Su, L.J. Guibas, Frustum pointnets for 3D object detection from RGB-D data, in: Proc. IEEE/CVF CVPR, 2018.
[23] Y. Chen, Y. Li, X. Zhang, J. Sun, J. Jia, Focal sparse convolutional networks for 3D object detection, in: Proc. IEEE/CVF CVPR, 2022.
[24] X. Chen, H. Ma, J. Wan, B. Li, T. Xia, Multi-view 3D object detection network for autonomous driving, in: Proc. IEEE/CVF CVPR, 2017.
[25] S. Vora, A.H. Lang, B. Helou, O. Beijbom, Pointpainting: Sequential fusion for 3D object detection, in: Proc. IEEE/CVF CVPR, 2020.
[26] S. Pang, D. Morris, H. Radha, CLOCs: Camera-lidar object candidates fusion for 3D object detection, in: Proc.IEEE/RSJ IROS, 2020.
[27] Y. Mao, Y. Cui, T.-W. Kuo, C.J. Xue, TRACE: A fast transformer-based general-purpose lossless compressor, in: Proc. ACM WWW, 2022.
[28] Y. Mao, J. Li, Y. Cui, J.C. Xue, Faster and stronger lossless compression with optimized autoregressive framework, in: Proc. IEEE DAC, 2023.
[29] Y. Mao, Y. Cui, T.-W. Kuo, C.J. Xue, Accelerating general-purpose lossless compression via simple and scalable parameterization, in: Proc. ACM MM, 2022.
[30] Y. Mao, W. Wang, H. Du, N. Guan, C.J. Xue, On the compressibility of quantized large language models, arXiv preprint (2024) arXiv:2403.01384.
[31] J. Li, Y.H. Cai, L. Liu, Y. Mao, C.J. Xue, H. Xu, Moby: Empowering 2d models for efficient point cloud analytics on the edge, in: Proc. ACM WWW, 2023.
[32] G. Jocher, et al., ultralytics/yolov5: v7.0 - YOLOv5 SOTA realtime instance segmentation, 2022, http://dx.doi.org/10.5281/zenodo.7347926.
[33] Federal Communications Commission, Measuring broadband America, 2022, https://www.fcc.gov/general/measuring-broadband-america.

[34] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P.R. Alface, T. Bostoen, F. De Turck, HTTP/2-Based adaptive streaming of HEVC video over 4G/LTE networks, IEEE Commun. Lett. 20 (11) (2016) 2177–2180.
[35] Linux traffic control (TC), 2023, https://linux.die.net/man/8/tc.
[36] Average 4G speed: How fast is 4G LTE compared to 4G+? 2020, https://commsbrief.com/average-4g-speed-how-fast-is-4g-lte-compared-to-4g/.
[37] gzip — Support for gzip files, 2023, https://docs.python.org/3/library/gzip.html.
[38] zlib — Compression compatible with gzip, 2023, https://docs.python.org/3/library/zlib.html.
[39] bz2 — Support for bzip2 compression, 2023, https://docs.python.org/3/library/bz2.html.
[40] lzma — Compression using the LZMA algorithm, 2023, https://docs.python.org/3/library/lzma.html.
[41] draco — A library for compressing and decompressing 3D geometric meshes and point clouds, 2023, https://github.com/google/draco.
[42] A. Bewley, Z. Ge, L. Ott, F. Ramos, B. Upcroft, Simple online and realtime tracking, in: Proc. IEEE ICIP), 2016, pp. 3464–3468, http://dx.doi.org/10.1109/ICIP.2016.7533003.
[43] H.W. Kuhn, The hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1–2) (1955) 83–97.
[44] M.A. Fischler, R.C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Commun. the ACM 24 (6) (1981) 381–395.
[45] L. Liu, H. Li, M. Gruteser, Edge assisted real-time object detection for mobile augmented reality, in: Proc. ACM MobiCom, 2019.
[46] L. Liu, J. Li, H. Xu, K. Xue, J.C. Xue, Efficient real-time video conferencing with adaptive frame delivery, Proc. Elsevier Comput. Networks (2023).
[47] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, E.A. Lee, Awstream: Adaptive wide-area streaming analytics, in: Proc. ACM SIGCOMM, 2018.
[48] L. Liu, J. Li, Z. Niu, W. Zhang, J.C. Xue, H. Xu, Efficient time-series data delivery in IoT with Xender, IEEE Trans. Mob. Comput. (2023).
[49] J. Li, H. Xu, M. Ma, H. Yan, C.J. Xue, Alfie: Neural-reinforced adaptive prefetching for short videos, in: Proc. IEEE ICME, 2022.
[50] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, L. Sun, Stick: A harmonious fusion of buffer-based and learning-based approach for adaptive streaming, in: Proc. IEEE INFOCOM, 2020.
[51] NVIDIA Jetson TX2 NVP model, 2023, https://developer.ridgerun.com/wiki/index.php/NVIDIA_Jetson_TX2_NVP_model.
[52] OpenPCDet Development Team, OpenPCDet: An open-source toolbox for 3D object detection from point clouds, 2020, https://github.com/open-mmlab/OpenPCDet.
[53] X. Shuai, Y. Shen, Y. Tang, S. Shi, L. Ji, G. Xing, milliEye: A lightweight mmwave radar and camera fusion system for robust object detection, in: Proc. ACM IoTDI, 2021.
[54] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, M.J. Freedman, Live video analytics at scale with approximation and delay-tolerance, in: Proc. USENIX NSDI, 2017.
[55] T.Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, H. Balakrishnan, Glimpse: Continuous, real-time object recognition on mobile devices, in: Proc. ACM SenSys, 2015.
[56] X. Ma, Y. Zhang, D. Xu, D. Zhou, S. Yi, H. Li, W. Ouyang, Delving into localization errors for monocular 3D object detection, in: Proc. IEEE/CVF CVPR, 2021.
[57] K. He, G. Gkioxari, P. Dollár, R. Girshick, Mask R-CNN, in: Proc. IEEE ICCV, 2017.
[58] Tegrastats utility, 2023, https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/DRIVE_OS_Linux_SDK_Development_Guide/Utilities/util_tegrastats.html.
[59] NVIDIA Jetson nano developer kit, 2023, https://developer.nvidia.com/embedded/jetson-nano.
[60] L. Zheng, S. Li, B. Tan, L. Yang, S. Chen, L. Huang, J. Bai, X. Zhu, Z. Ma, Rcfusion: Fusing 4-d radar and camera with bird's-eye view features for 3-d object detection, IEEE Trans. Instrum. Meas. (2023).
[61] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015, arXiv preprint arXiv:1510.00149.
[62] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proc. IEEE/CVF ICCV, 2017.
[63] Z. Liu, M. Sun, T. Zhou, G. Huang, T. Darrell, Rethinking the value of network pruning, 2018, arXiv preprint arXiv:1810.05270.
[64] B. Jacob, et al., Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proc. IEEE CVPR, 2018.
[65] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1, 2016, arXiv preprint arXiv:1602.02830.
[66] G. Hinton, O. Vinyals, J. Dean, et al., Distilling the knowledge in a neural network, 2015, arXiv preprint arXiv:1503.02531.
[67] R.T. Mullapudi, S. Chen, K. Zhang, D. Ramanan, K. Fatahalian, Online model distillation for efficient video inference, in: Proc. IEEE/CVF ICCV, 2019.
[68] Y. Umuroglu, N.J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, K. Vissers, Finn: A framework for fast, scalable binarized neural network inference, in: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017, pp. 65–74.
[69] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based accelerator design for deep convolutional neural networks, in: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2015, pp. 161–170.
[70] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al., Dadiannao: A machine-learning supercomputer, in: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE, 2014, pp. 609–622.
[71] H. Wang, Z. Zhang, S. Han, SpAtten: Efficient sparse attention architecture with cascade token and head pruning, in: Proc. IEEE HPCA, 2021.
[72] H. Tang, Z. Liu, X. Li, Y. Lin, S. Han, TorchSparse: Efficient point cloud inference engine, Proc. MLSys (2022).
[73] J. Li, L. Liu, H. Xu, S. Wu, C.J. Xue, Cross-camera inference on the constrained edge, in: Proc. IEEE INFOCOM, 2023.
[74] S. Jiang, Z. Lin, Y. Li, Y. Shu, Y. Liu, Flexible high-resolution object detection on edge devices with tunable latency, in: Proceedings of the 27th Annual International Conference on Mobile Computing and Networking, 2021, pp. 559–572.
[75] B. Fang, X. Zeng, M. Zhang, NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision, in: Proc. ACM MobiCom, 2018.
[76] A. Mathur, N.D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, F. Kawsar, DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware, in: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, 2017, pp. 68–81.
[77] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, Y. Zhang, Elf: Accelerate high-resolution mobile deep vision with content-aware parallel offloading, in: Proc. ACM MobiCom, 2021.
[78] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G.H. Xu, R. Netravali, Reducto: On-camera filtering for resource-efficient real-time video analytics, in: Proc. ACM SIGCOMM, 2020.
[79] K. Deng, D. Zhao, Q. Han, S. Wang, Z. Zhang, A. Zhou, H. Ma, 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.
[80] S. Zhang, Y. Li, X. Liu, S. Guo, W. Wang, J. Wang, B. Ding, D. Wu, Towards real-time cooperative deep inference over the cloud and edge end devices, Proc. ACM IMWUT 4 (2) (2020) 1–24.
[81] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, L. Tang, Neurosurgeon: Collaborative intelligence between the cloud and mobile edge, ACM SIGARCH Comput. Archit. News 45 (1) (2017) 615–629.
[82] H.-J. Jeong, H.-J. Lee, C.H. Shin, S.-M. Moon, IONN: Incremental offloading of neural network computations from mobile devices to edge servers, in: Proc. ACM SoCC, 2018, pp. 401–411.
[83] S. Laskaridis, S.I. Venieris, M. Almeida, I. Leontiadis, N.D. Lane, SPINN: synergistic progressive inference of neural networks over device and cloud, in: Proc. ACM MobiCom, 2020.
[84] V. Nigade, R. Winder, H. Bal, L. Wang, Better never than late: Timely edge video analytics over the air, in: Proc. ACM SenSys, 2021.



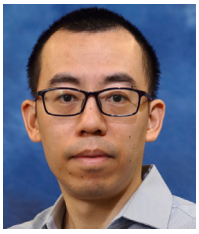
Jingzong Li is currently an assistant professor in the Department of Computer Science, The Hang Seng University of Hong Kong. He received his B.Eng. degree from University of Electronic Science and Technology of China in 2019, and his Ph.D. degree in Computer Science from City University of Hong Kong in 2024. His research interests include multimedia systems and edge computing. He is a member of ACM and IEEE.

Yik Hong Cai is an undergraduate student at the Chinese University of Hong Kong, majoring in computer science. His research interests include machine learning systems, distributed systems, and edge computing.

Libin Liu received his Ph.D. degree from the Department of Computer Science, City University of Hong Kong and his B.E. degree in software engineering from Shandong University. He is currently an Assistant Researcher with the Zhongguancun Laboratory. His current research interests include data analytics systems and machine learning for networking. He received the best paper award of ACM SIGCOMM 2022. He is a member of ACM and IEEE.

Yu Mao received the BS degree from the South China University of Technology, China, and the MSc degree from the Beijing University of Post and Telecommunication, China, in 2016 and 2019, respectively. She is currently working toward the PhD degree with the Department of Computer Science, City University of Hong Kong. Her research interests include machine learning in data engineering and light-weight neural networks.

Chun Jason Xue received the BS degree in computer science and engineering from the University of Texas at Arlington, in 1997, and the MS and PhD degrees in computer science from the University of Texas at Dallas, in 2002 and 2007, respectively. He is currently a professor at MBZUAI. His research interests include memory and parallelism optimization for embedded systems, software/hardware codesign, real time systems and computer security.



Hong Xu is an Associate Professor in CSE, CUHK. His research area is computer networking and systems, particularly big data systems and data center networks. From 2013 to 2020 he was with City University of Hong Kong. He received his B.Eng. from CUHK in 2007, and his M.A.Sc. and Ph.D. from University of Toronto in 2009 and 2013, respectively. His work has received best paper awards from ACM SIGCOMM 2022 and IEEE ICNP 2015, among others. He was the recipient of an Early Career Scheme Grant from the Hong Kong Research Grants Council in 2014. He is a senior member of IEEE and ACM.