Fast Polar Attentive 3D Object Detection on LiDAR Point Clouds

Manoj Bhat **Steve Han** **Fatih Porikli**
[1]Qualcomm
{manobhat, shizhan, fporikli}@qti.qualcomm.com

Abstract

3D object detection using LiDAR sensory point-cloud data is widely used for many applications, including autonomous driving and map building. Existing solutions mainly leverage deep learning models; nevertheless, one of the underlying challenges is reducing computational load and latency while maintaining high accuracy for detecting objects in the long-range. Here, we introduce a novel streaming detector utilizing polar space feature representations to provide faster inference for 3D object detection. Our method improves detection performance using pseudo-image features and can support edge devices with limited memory requirements. Comparing with other state-of-art methods along with experimental validations, we show our methods corroborates superiority on Waymo, KITTI dataset. On KITTI validation, it achieves 94.7% AP for cars in BEV detection.

1 Introduction

LiDAR is a crucial sensor for autonomous driving vehicles as it provides essential high fidelity 3D scene information. With the reducing cost of such sensors, they are becoming more available for mainstream systems, motivating perception solutions that directly work with LiDAR data. The point cloud from LiDAR is often used to detect 3D objects, such as cars, pedestrians, cyclists, signs, overhangs, driveable spaces, etc. Compared with a camera, LiDAR can capture precise 3D range i.e distance and relatively higher resolution angular location for the objects even when they are farther away from the sensor in long-range. These advantages make LiDAR the sensor of choice for ground truth annotations of 3D objects.

A majority of research efforts focus on improving the 3D object detection accuracy and simplicity [29; 11; 9] using deep learning methods. Although in recent years, one of the challenges is to reduce the inference latency and memory footprint without sacrificing detection accuracy. As such, the latency is directly related to the safety of autonomous driving since faster detection methods can improve the reaction time of autonomous systems to prevent potential accidents. Furthermore, the decreased computation load allows execution of multiple tasks i.e object detection, segmentation, tracking, path planning. etc. running simultaneously on the same platform from different sensors i.e camera, LiDAR, radar, etc. With these, the overall system must still maintain high accuracy, even for the corner cases such as detecting far small objects or nearby large reflective objects, when reducing the latency and memory due to its strict safety requirements.

Several works attempted to address the above challenges. Earlier approaches proposed to use 3D convolutions [6; 12], which can retain 3D geometry information for improved accuracy; however, they are also computationally intensive and require resources beyond system limitations in run-time. This shortcoming was followed up with efforts using combinations of 3D and 2D convolutions to reduce the latency and computational load [13; 32]. To this end, SECOND [29] proposed 3D sparse convolution to speed up the training and inference time with decreased memory consumption. Nevertheless, 3D

Machine Learning for Autonomous Driving Workshop at the 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

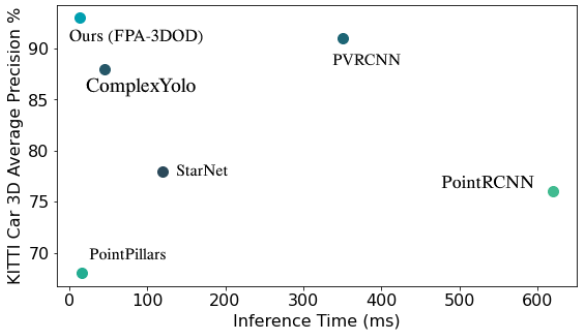

Figure 1: Inference time (indicating average latency) vs. car detection accuracy (in terms of average precision) evaluated on the KITTI validation dataset. As shown, our method achieves a high accuracy while running efficiently.

sparse convolution depends on platform-specific compilers and external implementations, which are not supported by popular deep learning libraries. Compared with 3D sparse convolutions, hybrid 2D convolutions can notably reduce latency and memory consumption. PointPillars [11] and their variants leverage PointNets to extract 2D pseudo-image features, which are pushed through a consecutive 2D convolution architecture, to increase the inference speed but lacks the detection accuracies against the approaches with 3D sparse convolutions. In comparison, our method can reduce the latency and keep the SOTA detection accuracy at the same time as shown in Figure 1.

In this paper, we propose a fast polar attentive 3D object detection (FPA-3DOD) method based on LiDAR point clouds as shown in Figure 2. Specifically, the input point cloud is first converted to a 2D pseudo-image with PointNet in the latent Cartesian space, which is then mapped into the latent polar space. We send a sequence of patches of the pseudo-image in the polar space to a transformer with positional embedding for self-attention. We also introduce a 3D intersection-over-union loss to enhance the detection performance further. The pseudo-image enables faster 2D convolutions. We show that the latent polar space features describe the input point cloud efficiently with higher occupancy rates [34], which allow us to use lower resolution polar space images for additional acceleration. Also, the polar space feature extraction is convenient for extending to stream-wise object detection, which reduces the latency and memory even further.

Our main contributions can be summarized as follows:

- We propose a transformer based 3D object detection architecture that leverage 2D pseudo-image features extracted in the polar space with PointNet and show that our model reduces the latency and memory usage without sacrificing detection accuracy.
- In the inference stage, our model sectors data for a stream-wise processing to make predictions without requiring a complete 360° LiDAR scan.
- We validate that our method outperforms the state-of-the-art methods in the publicly available KITTI and Waymo benchmarks.

2 Related works

Object Detection with Transformers: Vaswani et. al.[25] introduced transformers as attention-based building blocks in neural models. Researchers working towards natural language processing proposed methods [4; 21] improving and extending transformers with different architectures. Wang et. al.[26] applied transformers for video classification. It is shown that self-attention can capture long-range dependencies with favorable classification performance. [2] proposed DETR to use a transformer encoder-decoder for object detection by eliminating anchor boxes and non-maximum suppression. However, self-attention cost scales quadratically with the number of pixels, which is impractical for visual and 3D data. Instead of calculating the correlation of each two pixels, recent works proposed to derive only parts of correlations called sparse transformers [19]. [5] cropped input images into small patches and applied the transformer directly to patch sequences. In this paper, we inspire from sectoring strategy to reduce the load of self-attention for 3D points cloud data.

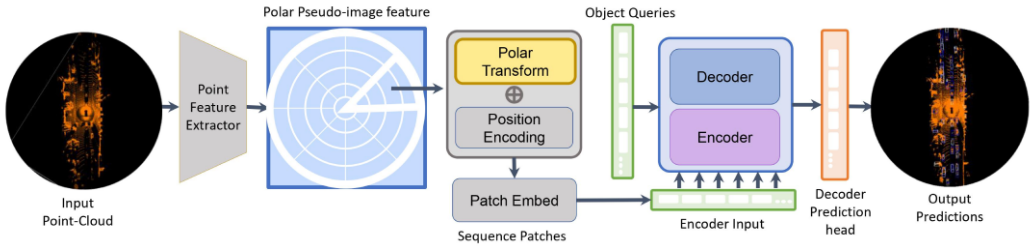

Figure 2: An overview of the proposed method. Point feature extractor derives features from raw points by an MLP [left]. The pseudo-image is transformed to a polar latent feature tensor [middle]. After sector-wise extraction of features, the embedding features are fed to the transformer. The decoding heads generate proposals.

Polar Space Feature Representations: In literature, most methods use the Cartesian space to extract the voxel-wise or point-wise feature for 3D tasks [30; 11; 3]. Increasingly, papers in recent works look into the Polar space in 2D spherical or 3D cylindrical feature extraction. Wu et. al.[27] proposed projection of 3D point cloud to a range/spherical view for feature representation. Darknet [1], SqueezeSegv2 [28], and RangeNet++ [15] use the same spherical feature projections for feature representation. [31] proposed the PolarNet to project the 3D point cloud to bird's eye view in polar coordinates. [34] use the PointNet [20] to extract the 3D cylindrical feature, which can keep the 3D geometry feature. Both[31; 34] claim that the polar space balances the points per grid/voxel compared with the Cartesian space. Motivated by these observations, we adopt PointNet to extract polar space features for 3D object detection.

3D Object Detection: Current CNN-based 3D object detection approaches can be categorized into three groups: 3D convolution methods, 2D convolution methods, and PointNet based methods. It is intuitive to use 3D convolutions for 3D object detection [12]. The pure 3D convolution is computationally expensive. Several works [32; 13] proposed combinations of 3D and 2D convolution on the voxel-wise features. To substitute the 3D standard convolution, [29] used the 3D sub-manifold sparse convolution proposed by Graham [8]. However, the 2D standard convolution is still faster even compared with the 3D spare convolution [11], which is not supported by popular deep learning libraries. In our proposed one-stage network, we utilize PointNet to extract the polar space feature. With the patch-based transformer architecture, we can achieve fast processing without sacrificing detection accuracy.

3 Methodology

Our 3D object detection architecture is minimal in size, and it consists of three lightweight modules: 1) A bird's-eyes-view (BEV) Cartesian space voxelization, 2) A Cartesian-to-polar mapping stage, and 3) A transformer self-attention module with prediction heads. In the inference time, we adopt two types of processing: 1) A full 360° point cloud processing and 2) A streaming sector-wise and ring-wise together called range-wise processing. We discuss the training details and stream-wise processing in further sections.

3.1 Cartesian Space Pseudo Image Extraction

Our motivation to use BEV features is to allow 2D convolutions for improved inference performance as they avoid expensive 3D convolutions and nearest neighbor search. We employ a learnable representation to ensure feature extraction is conditioned on the point cloud sparsity. For this, we first apply a global feature extraction layer that extracts the features directly from the Cartesian space of point clouds. We construct tensors of shape (D, N) from the input point cloud, where D has four dimensions corresponding to the x, y, z, and reflectance for the N number of input points. The learnable feature extractor is a single PointNet in the encoder. We assemble pillars [11] to create a tensor of size (D, P, M), where $P = H \times W$ is the number of voxels and M is the number of points inside each pillar voxel, which is set relative to the average number of points within each pillar voxel in the whole dataset. This voxel discretization is different for KITTI and Waymo. Since KITTI has annotations only in front of the LiDAR, the input is 360° point cloud with only points corresponding

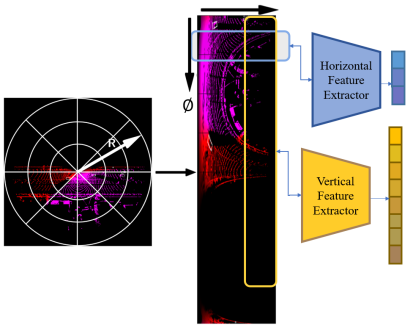

Figure 3: Transformation of the feature space into the polar space and further parsing of the tensor into vectors with row-wise as blue or column-wise as yellow with CNNs. These vectors are fed into the transformer.

to the camera frame. Using PointNet, we build a pseudo-image with the size of (C, H, W) where C is the number of output channels.

3.2 Cartesian to Polar Mapping

To build a pseudo-image representative of polar patterns of the data, we map the feature space into a latent polar pseudo-image using a tensor transformation. This mapping is an element-wise tensor operation as illustrated in the Equation 1.

$$\begin{aligned} \phi = \tan^{-1}\frac{y_l}{x_l},\ r = \sqrt{x_l^2 + y_l^2}, \\ \phi_x = \lfloor \frac{\phi}{S_\phi} \rfloor,\ r_y = \lfloor \frac{r}{S_r} \rfloor, \\ S_\phi = \frac{2\pi}{H_p},\ S_r = \frac{R_{max}}{W_p}. \end{aligned} \tag{1}$$

Here x_l and y_l are Cartesian coordinates or the row, column index in the pseudo image. H and W are the dimensions of the Cartesian feature space, which is then transformed to the polar space (H_p, W_p). R_{max} is the maximum radius in the polar space. Every tensor element in the Cartesian space is mapped to the polar space. ϕ_x is the index over length and r_y is the index over the width. S_ϕ and S_r is the one pixel size in polar space. The tensor values are filled for the missing elements using bilinear interpolation over bijective mapping.

Further, these tensor projections are fed into a collection of $Conv$ operators. The conv operators parse the tensor row-wise, equivalent to parsing each sectors, or column-wise equivalent to parsing each rings over all polar sectors into the same number of vectors as show in Figure 3. After this range-wise processing, the vectors are fed to the encoder and N detections are obtained at the decoder corresponding to the number of queries.

3.3 Transformer

For feeding the transformer sequence embeddings, we discretize the extracted pseudo-image in polar space with shape $x_{\text{polar}} \in \mathbb{R}^{C \times H_p \times W_p}$ is into patches of ring-sector combination. The rings and sectors are equivalent to the columns and rows accordingly in the polar space. We use a CNN to transform the features from these patches into an embedding of shape $1 \times E$, where E is the embedding vector size and the number of rings and sectors can be designed based on required resolution. A single Convolutional Kernel specializes in filtering specific signals for different ranges and sectors as per design.

Since we only need to output the 3D box properties for the corresponding classes efficient transformers [33] can be utilized. We use the standard transformer encoder architecture with inputs as a sequence of vectors (row or column-wise) after addition with positional encoding [5]. In row-wise case with

horizontal feature extraction, the encoder has K_{row} input embedding (whereas K_{col} after vertical feature extractor) for sector-wise processing as illustrated in Figure 3. Thus, a transformer with complete 360° input would have $K_{row} \times K_{col}$ inputs. Our architecture for both range-wise processing and 360° processing has 6 layers of encoder, 4 layers of decoder with N query inputs. N is selected based on the expected maximum number of predictions within a point-cloud frame. The final output feature vectors for the set of predictions are fed to detection heads to regress the box properties.

The prediction head regresses the outputs to Cartesian co-ordinates, size, and $sin\theta$ and $cos\theta$ of azimuth angle θ. This follows the method of training and inference using complex-Euler angles for heading [14]. Figure 2 illustrates the overall components from the Cartesian pseudo image extraction to polar mapping to the transformer to output the predictions back into Cartesian space.

3.4 Loss Function

We denote the ground truth set of objects as y and the set of N predictions as $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$. Assuming the number of objects in the one LiDAR frame is always less than N, we consider y also to be a set of size N padded with $\varnothing$ (no object). Each ground truth object is denoted as a set of parameters of an oriented 3D bounding box. Each box is defined as $(c_x, c_y, c_z, w, l, h, \sin(\theta), \cos(\theta), class)$. The first six parameters are 3-D Cartesian coordinates of object center and 3-D box sizes. The $\sin$ and $\cos$ of the heading angles of the box are following the methodology of finding the imaginary angles as in paper [14]. The last coefficient is the class of the bounding box. To find a bipartite matching between these two sets, we search for a permutation of N elements $\sigma \in \mathfrak{S}_N$ with the lowest cost:

$$\begin{aligned}\hat{\sigma} =& \underset{\sigma \in \mathfrak{S}_N}{\arg\min} \sum_{i}^{N} \mathcal{L}_{\text{match}}\left(y_i, \hat{y}_{\sigma(i)}\right) \\ =& \underset{\sigma \in \mathfrak{S}_N}{\arg\min} \sum_{i=1}^{N} \log\left[-\hat{p}_{\sigma(i)}\left(c_i\right) + 1_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}\left(b_i, \hat{b}_{\sigma(i)}\right)\right]\end{aligned} \tag{2}$$

Here, $\mathcal{L}_{\text{match}}\left(y_i, \hat{y}_{\sigma(i)}\right)$ is a pair-wise matching cost between the ground truth y_i and prediction $\hat{y}_{\sigma(i)}$ with permutation index $\sigma(i)$. $\hat{p}_{\sigma(i)}\left(c_i\right)$ is defined as the probability of class c_i, and $\hat{b}_{\sigma(i)}$ is the predicted box. Still, we need to find one-to-one matching for direct set prediction without duplicates instead of matching the best fit region proposal matches to calculate loss. Hence we compute the Hungarian loss for all pairs matched [7].

We define the loss as the loss of common object detectors including two parts, i. e. a linear combination of a negative log-likelihood for class prediction and a box matching loss as Equation 2. In the first part of matching cost we use probabilities $\hat{p}_{\sigma(i)}\left(c_i\right)$ instead of log-probabilities.

The second part of the matching cost is the Hungarian loss $\mathcal{L}_{\text{box}}\left(\cdot\right)$ also called SetLoss that scores the bounding boxes similarities. Unlike many detectors that perform box predictions as a difference w.r.t. some initial guesses anchors, we make box predictions directly in an anchor-free fashion. The most commonly used ℓ_1 loss will have different scales for small and large boxes even if their relative errors are similar. To mitigate this issue, we use a linear combination of the ℓ_1 loss and the generalized IoU loss [38] called 3D-IOU loss $\mathcal{L}_{\text{iou}}(\cdot, \cdot)$ that is scale-invariant. Overall, our box loss is:

$$\mathcal{L}_{\text{box}}\left(b_i, \hat{b}_{\sigma(i)}\right) = \lambda_{\text{iou}}\, \mathcal{L}_{\text{iou}}\left(b_i, \hat{b}_{\sigma(i)}\right) + \lambda_{\text{L1}} \mathcal{L}_{\text{L1}}\left(b_i, \hat{b}_{\sigma(i)}\right) \tag{3}$$

where λ_{iou} and $\lambda_{\text{L1}} \in \mathbb{R}$ are hyperparameters. Finally, these two losses are normalized by the number of objects inside the batch.

3.5 Sector Processing for Streaming

In literature, the majority of methods for 3D object detection assume availability of complete 360°rotating LiDAR data. Including the total time of preprocessing and postprocessing, such approaches are handicapped for online real-time applications. To address this issue, [9] proposed a streaming object detection by utilizing the sector-wise 3D object detection and stateful-NMS, which takes into account the detection in the previous sectors when determining whether a new detection is unique. The sector-wise object detection can reduce the FLOPS significantly while keeping a

Method names		$AP_{BEV}(IOU = 0.7)$			AP_{3D}			AOS_{AP}
Reference	Input representation	Easy	Mod.	Hard	Easy	Mod.	Hard	-
ComplexYolo [14]	Pseudo-Image	75.32	70.19	62.43	62.11	59.28	55.23	78.45
StarNet [17]	Point-Based	88.53	83.79	77.9	81.94	71.88	66.38	79.20
PointPillars [11]	Pseudo-Image	88.36	86.1	79.83	79.05	74.99	68.35	86.51
PointRCNN [23]	Point-Based	90.34	88.21	86.68	89.71	79.45	75.82	87.98
PV-RCNN [22]	Point-Voxel Fusion	92.24	89.96	78.32	90.52	**83.70**	72.32	89.94
FPA-3DOD	Pseudo-Image	**94.70**	**90.02**	**87.76**	**91.56**	82.35	**79.85**	**93.27**

Table 1: State-of-art comparisons for 3D detection on KITTI validation set for Car class, the results are evaluated by the mean Average Precision with 40 recall positions on the evaluation set. Note the results for the models are results as per checkpoints provided by [OpenPCDet] [MMDetection3D].

	Method Names	AP	
Metric Levels		Vehicle	Pedestrian
Level 1	PointRCNN [23]	61.2	64.8
	PointPillars [11]	59.2	62.1
	PV-RCNN [22]	73.3	60.2
	Ours	**76.3**	**72.7**
Level 2	PointRCNN [23]	59.8	60.2
	PointPillars [11]	56.4	57.9
	PV-RCNN [22]	65.4	68.3
	FPA-3DOD	**69.8**	**70.1**

Table 2: Performance comparison of the state-of-the-art for 3D object detection on Waymo Open Dataset (Version 1.2 with fixed evaluation scripts) validation set.

comparable detection accuracy. Therefore, we leverage on the stateful-NMS [9] in our model when we sector-wise process from the polar space, feeding sector-wise embedding to the transformer. While training, the model learns to extract global features and detect in Polar-Cartesian space. Consequently for inference the same trained parameters can be then utilized to infer on cropped inputs for peculiar sectors introducing streaming between the sectors. Thus we illustrate that same transformer trained for 360° detection in the polar space can be used in inference for streaming for a particular sector input by masking out the remaining sequential inputs. The advantage of this method is to avoid the requirement of separate training of model for streaming inference.

4 Experiments

4.1 Datasets

We use the KITTI dataset with training only Camera-frame pointcloud and 360°range wise processing to evaluate the performance of our model. In addition, the Waymo dataset is utilized for experimenting streaming method with Stateful-NMS in sector-wise processing due to availability of 360°fully annotated upto 80m point-cloud data. Details about both the datasets are as follows.

KITTI dataset provides 7481 training and 7518 testing data for 3D object Detection for cars, cyclists, and pedestrians. The proposed model is only evaluated on Car as most research is on the evaluation of the class. The detection performance is compared with official KITTI evaluation metrics, which are 3D detection average precision and BEV average precision. For each class, there are three difficulty levels; easy, moderate, and hard, based on the size, occlusion level, and truncation.

Waymo open dataset contains 798 training and 202 validation sequences for Vehicle and Pedestrian class. The LiDAR points are 180k per 0.1s. We use the official metrics to evaluate the model's performance: AP and mAP weighted by heading accuracy. The IoU threshold is 0.7 for vehicles and 0.5 for pedestrians. The latest update of the evaluation toolkit provides a breakdown into two levels, $level_1$ for boxes with more than five LiDAR points and $level_2$ for boxes with at least one LiDAR point.

Model	AP_{3D}	Parameters (millions)	FLOPs (Giga)	Average Inference Time
PointRCNN	81.66	2.20	25	620ms
PointPillars	74.11	1.43	32	16ms
FPA-3DOD	84.58	0.59	6	14ms

Table 3: Comparison of Model Parameters, FLOPS for 16 slice streaming model and 3D AP of Car class in KITTI validation split set with AP calculated as per recall positions for Car class

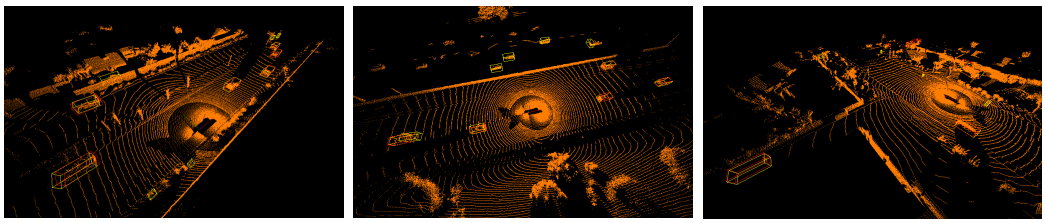

Figure 4: Qualitative description of performance of the model on Waymo dataset, Yellow boxes are model predictions whereas the Red-boxes are the Ground-Truth annotations. The outputs are 360°inference on a 360°trained model illustrating far-range detection.

4.2 Implementation Details

Feature Extractor At the start of the pipeline, Multi layer Perceptron (MLP) is used of PointNet [20] module to parse the voxelized/discretized point-cloud into a pseudo image. The projected pseudo image is of the shape [C, 512, 512], where the C is the number of channels after encoding the features from voxels into a canvas tensor. This pseudo image tensor is transformed into a polar feature dimension of shape [C, 1024, 258] by the conversion formula in Equation 1. The dimensions of the polar space tensor are $(512, 1024)$. This tensor is then cropped to construct 64×32 dimension crops, which is an input to 16 specialized CNNs for sector-wise embedding generation with output shape dimension $E = 512$. A position encoding is added to the embedding from the crops. The feature extractor is minimal and one stage for streaming perception hence reduces memory consumption. A single CNN extracts the current sector features for next phase-shifted sequential input to the transformer along with prior sector inputs.

Transformer The transformer encoder-decoder network is trained using an AdamW [10] optimizer with one-cycle learning curve [24] with initial learning rate of $2e^{-4}$. The batch size is four, and the network is trained for 100 epochs. For the transformers, the number of encoder layers is six, and the number of decoder layers is four. The hyperparameter values are determined heuristically. The whole pipeline is trained from scratch for about 220GPU hours in 4 Nvidia 1080Ti's with Waymo dataset for each model experiment either with Cartesian or polar-cropping.

Streaming Detection The addition of streaming method reduces the peak computational demand due to the divide-and-conquer concept and reduces it to a fraction of $\frac{1}{n}$ where n is the number of slices, from the total. We report the peak FLOPS in table 3.

4.3 Results

We report performance and latency results for the KITTI and Waymo datasets. Table 1 provides a comparison of our performance over the state-of-art methods. Our model outperforms others with a large margin and high recall scores for BEV metrics and the average orientation KITTI metrics

Classes	1-slice	2-slice	4-slice	8-slice	16-slice	32-slice	64-slice	128-slice
Vehicles	76.3	71.59	62.53	61.24	58.91	36.79	24.20	5.78
Pedestrian	72.7	68.47	50.29	48.28	35.16	20.12	5.71	1.92
FLOPs(Giga)	5.2	4.1	2.8	1.1	0.6	0.2	0.08	0.03

Table 4: Comparison of average precision (AP) over the number of sectors/slices within a single rotation for Vehicles and Pedestrian while using stateful-NMS [9] evaluated on Waymo dataset level 1. The network FLOPs for a single forward pass is also calculated. The total flops in inference is for a single frustum f_a where f = Frustum at angular spacing a. For n-slices the spacing is 360°/n

AP_{3D}								
Distance	1-slice	2-slice	4-slice	8-slice	16-slice	32-slice	64-slice	128-slice
0-20m	98.25	97.49	97.12	95.79	94.31	90.15	85.10	78.22
20m-50m	72.4	68.25	65.25	64.11	63.53	58.96	51.26	49.25
50m-100m	29.25	23.71	21.98	19.41	16.68	12.49	10.34	4.86
AP_{BEV}								
0-20m	99.61	98.16	97.67	95.33	94.18	93.11	93.10	91.06
20m-50m	85.96	84.86	84.04	83.52	81.57	78.12	75.10	70.49
50m-100m	58.15	49.16	47.36	45.25	43.18	42.89	42.11	37.30

Table 5: Comparison of average precision (AP) in both 3D and BEV (Birds-eye-view) over the distances comparing the long range detection performance for various streaming evaluated on Waymo dataset level 1

Loss	AP_{BEV}	AP_{3D}	AOS_{AP}
3D-IOU + L2	92.4	89.7	93.4
L2 + SetLoss	90.11	87.43	89.01
3D-IOU + SetLoss	91.25	82.34	90.16
3D-IOU + L2 + SetLoss	94.66	91.05	**94.35**
3D-IOU + L1 + SetLoss	**94.70**	**91.56**	93.27

Table 6: Effect of loss function combination on the Performance of the FPA-3DOD on KITTI dataset only for the Car class. Evaluated on Camera frame field-of-view annotations.

(under the IOU threshold of 0.7). Whereas PV-RCNN [22] is better only for moderate difficulty in KITTI for 3D average precision metric.

Our model outperforms others for the Waymo benchmark for vehicle and pedestrian class AP in both metric levels as shown in Table 2. We also analyze the efficacy of the model in Table 3. These results validate that the proposed method is faster with fewer parameters while maintaining high accuracy.

To improve the inference speed, we utilize the streaming framework. As we already perform the sector-space, i.e., range-based featurization, we artificially slice polar feature space with a sector angle parameter. The performance of the model with corresponding input shapes based on the number of slices in 360° of point cloud data is given in Table 4. It is observed that the performance is consistent for a pipeline until 4-slice for both vehicle and pedestrian classes, and starts slightly degrading after 8-slice of 45°. Which is coherent with [9].

4.4 Ablation Study

In this section, we provide ablation experiments to dissect the effects of individual components of FPA-3DOD and discuss our design choices based on experiments.

Effect of loss function As shown in Table 6, we compare the performance of models after training with different loss configurations. We consider the combinations to analyze the performance characteristics by training a different network with the same initialization. The AP score is the best when our model is trained with 3D-IoU for detection, L1 for 2D bounding box location, shape, and SetLoss for detection and classification between boxes as illustrated in Table 6.

Effect of cropping on feature space For a standard backbone feature extractor, we compare with different cropping (sector processing) strategies within polar and Cartesian input feature spaces. A simple cropping strategy is described by ViT [5], which focuses on cropping sub-pixel spaces and feeding the resulting flattened vector to the transformer. Cartesian cropping does not consider the range-based feature distribution. In contrast, polar cropping stretches the features range-wise as in Figure 3 providing an effective prior for the encoder. The position encoding help in leveraging range dependency. The performance improvement is evident from the results shown in Table 7. In comparison to Cartesian cropping, we find that the polar cropping is much faster, as shown in Table 7. This is speculated due to the higher resolution convolutions used for processing larger patch dimensions [64 $\times$ 64] compared to the polar feature extraction of [32 $\times$ 16] for range-wise streaming. Additionally, comparisons of the number of parameters vs. the inference time and AP performance are shown in Table 7.

Model	Mean Inference Time	AP_{3D}	CNN Parameters
Cartesian	24ms	90.39	524K
Polar	14ms	94.71	65K

Table 7: Effect of cropping strategies on the 360° point-cloud 3D object detection, the AP results are performance in the KITTI validation dataset for front-view detections.(K=Thousand)

5 Conclusion

We proposed FPA-3DOD, a polar sequence-to-sequence feature extraction framework for efficient streaming 3D object detection from the LiDAR point clouds. Our method uses PointNet feature extractor with a few convolution layer operators on each ring/sector to produce embeddings as input to the sequence-to-sequence processing transformer model. The detection head on top of which produces detections without using NMS. Further, it uses stateful-NMS as defined in [9] for processing detection outputs stream-wise. An advantage of our work is no-training streaming inference i.e without any additional training of 360°detection transformer, which could be used on top of any feature extraction from 3D pointclouds or sparse 2-D data. To conclude, FPA-3DOD is a real-time performing 3-D object detection model, and it achieves state-of-the-art performance on the validation set of KITTI and Waymo benchmarks. We hope our contribution advances research in point-cloud scene understanding systems and open doors for robust L4 Autonomous Driving Perception. Future direction on top of this article would involve more matured streaming methods, improving small object detection though efficient transformers [33], self-supervised sequential training for 3D detection and Domain generalization between LiDAR point-cloud Datasets.

References

[1] Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., and Gall, J. (2019). Semantickitti: A dataset for semantic scene understanding of lidar sequences. pages 9297–9307.

[2] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. pages 213–229.

[3] Chen, Y., Liu, S., Shen, X., and Jia, J. (2019). Fast point r-cnn. pages 9775–9784.

[4] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[5] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[6] Engelcke, M., Rao, D., Wang, D. Z., Tong, C. H., and Posner, I. (2017). Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. pages 1355–1361.

[7] Feiyang Chen, Nan Chen, H. M. H. H. (2019). The application of bipartite matching in assignment problem.

[8] Graham, B., Engelcke, M., and Van Der Maaten, L. (2018). 3d semantic segmentation with submanifold sparse convolutional networks. pages 9224–9232.

[9] Han, W., Zhang, Z., Caine, B., Yang, B., Sprunk, C., Alsharif, O., Ngiam, J., Vasudevan, V., Shlens, J., and Chen, Z. (2020). Streaming object detection for 3-d point clouds. pages 423–441.

[10] Ilya Loshchilov, F. H. (2017). Decoupled weight decay regularization.

[11] Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. (2019). Pointpillars: Fast encoders for object detection from point clouds. pages 12697–12705.

[12] Li, B. (2017). 3d fully convolutional network for vehicle detection in point cloud. pages 1513–1518.

[13] Luo, W., Yang, B., and Urtasun, R. (2018). Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. pages 3569–3577.

[14] Martin Simon, Stefan Milz, K. A. H.-M. G. (2020). Complex-yolo: Real-time 3d object detection on point clouds.

[15] Milioto, A., Vizzo, I., Behley, J., and Stachniss, C. (2019). Rangenet++: Fast and accurate lidar semantic segmentation. pages 4213–4220.

[MMDetection3D] MMDetection3D. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection.

[17] Ngiam, J., Caine, B., Han, W., Yang, B., Chai, Y., Sun, P., Zhou, Y., Yi, X., Alsharif, O., Nguyen, P., et al. (2019). Starnet: Targeted computation for object detection in point clouds. *arXiv preprint arXiv:1908.11069*.

[OpenPCDet] OpenPCDet. Openpcdet: An open-source toolbox for 3d object detection from point clouds.

[19] Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. (2018). Image transformer. pages 4055–4064.

[20] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. pages 652–660.

[21] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[22] Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., and Li, H. (2020). Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. pages 10529–10538.

[23] Shi, S., Wang, X., and Li, H. (2019). Pointrcnn: 3d object proposal generation and detection from point cloud. pages 770–779.

[24] Smith, L. N. (2017). Cyclical learning rates for training neural networks.

[25] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

[26] Wang, X., Girshick, R., Gupta, A., and He, K. (2018). Non-local neural networks. pages 7794–7803.

[27] Wu, B., Wan, A., Yue, X., and Keutzer, K. (2018). Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. pages 1887–1893.

[28] Wu, B., Zhou, X., Zhao, S., Yue, X., and Keutzer, K. (2019). Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. pages 4376–4382.

[29] Yan, Y., Mao, Y., and Li, B. (2018). Second: Sparsely embedded convolutional detection. *Sensors*.

[30] Yang, B., Liang, M., and Urtasun, R. (2018). Hdnet: Exploiting hd maps for 3d object detection. pages 146–155.

[31] Zhang, Y., Zhou, Z., David, P., Yue, X., Xi, Z., Gong, B., and Foroosh, H. (2020). Polarnet: An improved grid representation for online lidar point clouds semantic segmentation. pages 9601–9610.

[32] Zhou, Y. and Tuzel, O. (2018). Voxelnet: End-to-end learning for point cloud based 3d object detection. pages 4490–4499.

[33] Zhu, X., Su, W., Lu, L., Li, B., Wang, X., and Dai, J. (2020a). Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*.

[34] Zhu, X., Zhou, H., Wang, T., Hong, F., Ma, Y., Li, W., Li, H., and Lin, D. (2020b). Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. *arXiv preprint arXiv:2011.10033*.