Attention-based Proposals Refinement for 3D Object Detection

Minh-Quan Dao, Elwan Héry, Vincent Frémont

Attention-based Proposals Refinement for 3D Object Detection

Minh-Quan Dao, Elwan Héry, Vincent Frémont

***Abstract*— Recent advances in 3D object detection is made by developing the refinement stage for voxel-based Region Proposal Networks (RPN) to better strike the balance between accuracy and efficiency. A popular approach among state-of-the-art frameworks is to divide proposals, or Regions of Interest (ROI), into grids and extract feature for each grid location before synthesizing them to form ROI feature. While achieving impressive performances, such an approach involves a number of hand crafted components (e.g. grid sampling, set abstraction) which requires expert knowledge to be tuned correctly. This paper proposes a data-driven approach to ROI feature computing named APRO3D-Net which consists of a voxel-based RPN and a refinement stage made of Vector Attention. Unlike the original multi-head attention, Vector Attention assigns different weights to different channels within a point feature, thus being able to capture a more sophisticated relation between pooled points and ROI. Experiments on KITTI *validation* set show that our method achieves competitive performance of 84.84 AP for class Car at Moderate difficulty while having the least parameters compared to closely related methods and attaining a quasi-real time inference speed at 15 FPS on NVIDIA V100 GPU. The code is released[1].**

I. Introduction

Object detection is a crucial component of autonomous vehicles because it provides input for downstream tasks such as prediction of other road users motion which essentially influence the motion planning of the ego vehicle. Due to the need for localizing objects in 3D space, object detection for autonomous vehicles is often performed on point cloud collected by 3D LiDAR. The unstructured and sparse nature of point cloud makes it unsuitable for convolutional neural networks (CNNs) to operate. Early works [1], [2] rasterize point cloud to Bird-Eye View (BEV) to enable the use of standard 2D CNNs. Their encouraging results motivate studies on learning BEV representation of point cloud [3], [4], [5]. Their common point is the discretization of point clouds to 3D grids made of voxels, with PointPillars [5] being the extreme case where voxels have infinite size along the vertical direction. Voxel-based methods have excellent inference speed thanks to the regular grid structure brought by the voxelization step. However, their performances are rather limited due to lack of 3D structure in the BEV representation.

Aware of such drawback, there is a number of works, e.g. [6], [7], [8], advocating for operating directly on raw point cloud by using Set Abstraction and Feature Propagation (proposed by PointNet++ [9]) instead of Convolution. The fine grain structure preserved by operating at point level helps point-based methods outperform voxel-based methods in various benchmarks. The drawback of point-based methods is their low frame rate caused by point cloud query operators (e.g. ball query, k-nearest neighbors).

Recently, there has been a renaissance in voxel-based methods. This stems from the observation that voxel-based methods such as SECOND [4] have exceptionally high recall rate (up to 95%) yet only achieves a moderate performance, e.g. SECOND's 78 Average Precision (AP) for Car class in KITTI. The new trend is to develop a refinement stage to unleash the full potential of this class of methods. The key of the refinement stage is how to effectively compute Region of Interest (ROI) features. Early works, e.g. PartA^2 [10], PV-RCNN [11], and VoxelRCNN [12], address this by first dividing ROI into a 3D grid then extracting feature at each grid location before feeding the concatenation of grid point features to a Multi-Layer Perceptron (MLP) to obtain the desired output. Their motivation is that such a grid can recover the 3D structure lost in BEV representation used in the region proposal stage. Arguing that computing grid point features requires several hand crafted components, CT3D [13] devises a variant of the transformer [14] to compute ROI feature directly from points pooled from raw point cloud. Though having less inductive bias, CT3D achieves state-of-the-art performance, demonstrating the benefit of integrating transformer to 3D detection pipeline.

This paper adds to the family of two-stage voxel-based 3D object detectors by making two main contributions. First, we develop a new module, named ROI Feature Encoder (RFE), for computing per-proposal features based on Vector Attention [15]. Together with a detection head made of MLPs, RFE serves as a refinement stage and can be integrated to both voxel-based and point-based region proposal frameworks. Second, based on the observation that strong methods such as PV-RCNN [11] and CT3D [13] employ additional modules to learn pooled point features which results in increasing model size and reducing frame rate, we propose to pool directly from feature maps generated by the backbone during region proposal process. Inspired by [16], our pooling strategy effectively fuses multi-scale features, thus increasing model's ability to detect classes of different sizes. In addition, we carry out extensive experiments to validate the effectiveness of our method as well as validating the design choices we made. In the following, we first highlight our differences compared to closely related works in Section.II, then provide the conceptual details of APRO3D-Net in Section.III. Section.IV presents the implementation details and performance of our method on KITTI [17] and

Authors are with Nantes Université, École Centrale de Nantes and CNRS LS2N, 44300 Nantes, France first-name.last-name@ec-nantes.fr

[1]https://github.com/quan-dao/APRO3D-Net

NuScenes [18] dataset as well as conducts ablation studies. Conclusion and outlook are draw in Section.V

II. Related Works

As mentioned in Section.I, our work belong to the family of two-stage voxel-based detectors which comprises of PartA^2 [10], PV-RCNN [11], VoxelRCNN [12], and CT3D [13]. Compared to the first three methods, we are similar in the interest of using inductive bias to compensate for the lost of 3D structure in BEV representation used in the proposals generation stage. While their inductive bias is to impose a grid structure to ROI, ours takes place in position encoding of pooled points (Section.III-B.3). Specifically, pooled points' coordinates are mapped to ROI's canonical frame then augmented with their displacement vector with respect to ROI's eight vertices. The difference between pooled points' augmented coordinates and that of ROI's center is used as input to an MLP for computing position encoding.

Our pooling strategy shares the same source of VoxelRCNN [12] which is the intermediate feature maps generated by the 3D backbone of the region proposal framework. Our difference is that instead of concatenating features pooled across different scales, we first pool from the highest scale to compute initial ROI features, then update this initial ROI features by pooling from another feature map of lower scale. This is to condition the computing of ROI features at the lower scale on the higher one, thus encouraging the consistency of learned features throughout the architecture.

CT3D [13] is the closest to our proposed approach since we share the method of computing ROI features via attention mechanism. Compared to CT3D, we have two key differences. First, we use a different formulation of the attention mechanism, namely vector attention [15], to assign different attention weights to different channels of one point feature. The motivation for this will be explained in Section.III-B.2. Second, CT3D pools from raw point cloud to enable its integration to virtually any detection framework. Such flexibility comes at the cost of ignoring the valuable intermediate results of the region proposal process. This forces CT3D to recompute features for pooled points before transforming them to ROI features using the self-attention in which a pooled point feature is computed as a weighted sum of others'. Self-attention has a quadratic complexity, thus increasing CT3D memory footprint and inference time. In contrast, our method pools from backbone's intermediate feature maps. As a result, it is no longer necessary to recompute pooled features. Furthermore, by re-using backbone features, our pooling strategy maximizes the use of the information produced in the region proposal process.

It worth to notice that there is a number of works on developing a full-transformer 3D object detectors [19], [20], [21], [22] which are orthogonal to this paper.

III. APRO3D-Net for 3D Object Detection

The overview of APRO3D-Net made by integrating our ROI Feature Encoder (RFE) modules to SECOND [4] is presented in Figure.1. After the first-stage of proposals generation, backbone-generated feature maps are interpreted to point-wise features which are then pooled into ROI. Pooled points are positionally encoded to incorporate ROI 's geometry. The resulted position encoding and their features are transformed into ROI feature via vector attention. ROI feature are mapped to ROI's confidence score and refinement vector by two MLP-made heads.

A. 3D Backbone and Region Proposal Network

The reason we choose SECOND to demonstrate our method instead of a point-based method such as PointRCNN is two folds. First, point-based methods are not as computationally efficient because of their repetitive use of query operations (e.g. ball query and k-nearest neighbor query) which can take up to 80% computational time [23]. More importantly, the final performance is conditioned on how well ROI produced by the Region Proposal Network (RPN) covers the set of ground truth boxes which is measured by RPN's recall rate. [10] has shown that SECOND-like RPN delivers a higher recall rate compared to the bottom-up proposal strategy of PointRCNN.

SECOND first uses a backbone made of Sparse Convolutions to learn a compact representation of the input point cloud. The backbone's final output is a C-channel feature volume $D \times H \times W$. It is then converted into a BEV representation of size $(C \times D) \times H \times W$ by flattening the channel and depth dimension. At each pixel of the resulted BEV image, multiple anchors corresponding to different classes and orientations are assigned. Finally, a RPN which is a standard 2D CNN computes a deeper representation for this BEV image before predicting class probability and offset vector w.r.t associated ground truth for each anchor. After being adjusted by predicted offset vector, anchors become Regions of Interest (ROI). ROI are post-processed by the non-max suppression (NMS) procedure to remove redundant but low confident ROI. The output of this module is a set of ROI $\mathcal{R}$

$$\mathcal{R} = \{([x_r, y_r, z_r, dx_r, dy_r, dz_r, \theta_r], \text{cls}_r)\}_{r=1}^{M} \quad (1)$$

where each ROI is parameterized by the location of its center $[x_r, y_r, z_r]$, its size $[dx_r, dy_r, dz_r]$, its heading direction (i.e. yaw angle) θ_r and its class cls_r.

B. ROI Feature Encoder

RFE has three sub-modules: Feature Map Pooling, Position Encoding and Attention Module. The Feature Map Pooling interprets backbone-generated feature volumes into point features and pools them according to their location and ROIs' bounding box. After being pooled, points are positionally encoded to incorporate ROI geometry. The Attention Module transforms pooled features and their position encoding to ROI feature via the Vector Attention [15] which essentially is a weighted sum of pooled point features.

1) Feature Maps Pooling: Intermediate feature maps generated by the backbone, denoted by $\mathcal{F}^i$ in Figure.1, represent 3D space at different scales in the form of voxels grid. Using these scales (i.e. voxel size), a feature map can be interpreted

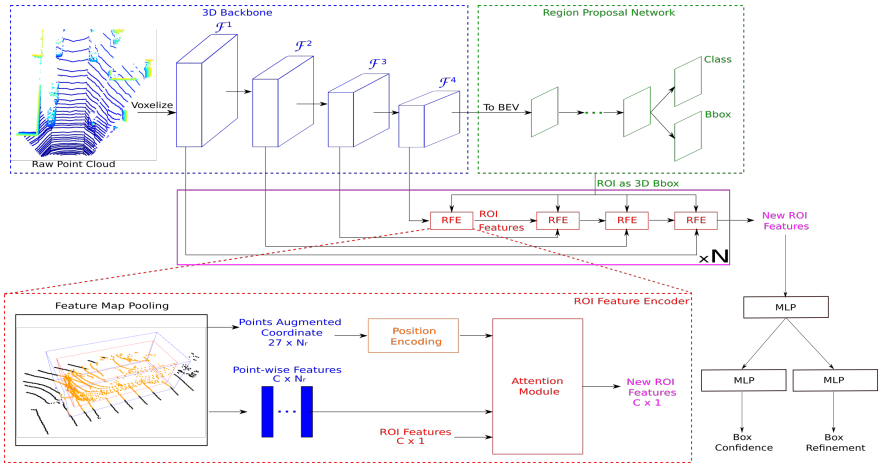

Fig. 1. The overall architecture of APRO3D-Net. The voxelized point cloud is fed to a 3D backbone for feature extraction. The backbone's output is then converted to a BEV representation on which an RPN is applied to generate ROI. Several ROI Feature Encoders (RFE) transform feature maps produced by backbone into ROI features by: first pooling from inputted feature maps, then encoding pooled points position, finally refining previous ROI feature using pooled features and their position encoding via Attention Module. Refined ROI feature is mapped to confidence and refinement vector by two MLP-based detection heads. Here, blue cuboids and green parallelogram respectively denote feature maps computed by 3D and 2D convolution. Notice that the channel dimension is omitted for clarity.

into point-wise features which can be pooled to compute ROI features.

To be specific, let the LiDAR frame $\mathtt{L}$ relative to which point cloud is expressed be defined as follows: origin is at LiDAR's location, X-axis coincides with ego vehicle's heading direction, Z-axis is the reversed gravity direction, Y-axis is the cross product of Z and X-axis. An occupied grid location (d, h, w) is interpreted into a 3D location (x, y, z) by

$$^{\mathtt{L}}\left[x, y, z\right] = \left(\left[w, h, d\right] + 0.5\right) \cdot V^i + {}^{\mathtt{L}}\left[x, y, z\right]_{\min} \qquad (2)$$

Here, V^i is the voxel size of $\mathcal{F}^i$. It worth to notice that d, h, w are respectively the voxel's *grid location* along the Z-axis, Y-axis, and X-axis. ${}^{\mathtt{L}}[x, y, z]_{\min}$ is the minimum coordinate in LiDAR frame $\mathtt{L}$.

Applying Eq.(2) to every occupied voxel of $\mathcal{F}^i$ results in a set of point-wise features $\mathcal{P}^i = \left\{\left({}^{\mathtt{L}}\mathbf{p}^i_j = \left[x^i_j, y^i_j, z^i_j\right], \mathbf{f}^i_j\right)\right\}_{j=1}^{N^i}$. Here, $\mathbf{f}^i_j$ is the feature at the grid location gives rise to $\mathbf{p}^i_j$, while N^i is the number of occupied voxels in $\mathcal{F}^i$.

The pooling scheme, illustrated in the bottom-right corner of Figure.1, is performed based on the location of point-wise features $\mathcal{P}^i$ with respect to the box defined by the enlarged version of ROI r. Let $\mathfrak{R}_r$ denote the 3D volume occupied by ROI r after being enlarged by $[\Delta_x, \Delta_y, \Delta_z]$. A point feature $\left(\mathbf{p}^i_j, \mathbf{f}^i_j\right)$ is pooled into ROI r if $\mathbf{p}^i_j \in \mathfrak{R}_r$. The reason for enlarging ROIs is to incorporate missing foreground points due to the miss alignment with ground truth.

Inspired by [6], [10], we transform pooled point-wise features to ROI's canonical coordinate system to reduce the variance during training, thus improving model's generality. This coordinate system, shown in Figure.2, is defined as follows: origin is at ROI's center, the X-axis has the same direction as ROI's heading direction, the Z-axis is vertical

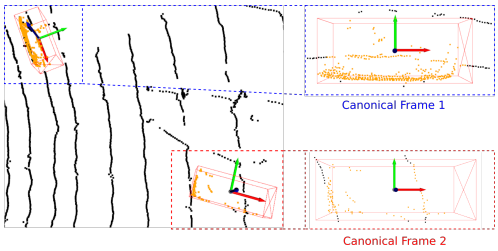

Fig. 2. ROI and pooled points in LiDAR frame (Right) compared to them in canonical frame (Left). Here, ROI are denoted by red cuboids while pooled points are colored orange. Red, green and blue arrows respectively represents canonical frame's X, Y, Z axis.

and points upward. From Eq.(1), a ROI is characterized by a seven-vector $[x_r, y_r, z_r, dx_r, dy_r, dz_r, \theta_r]$. A point $\mathbf{p}_j$ is transformed to a ROI's canonical frame by

$${}^{r}\mathbf{p}_j = \begin{bmatrix} \cos\theta_r & \sin\theta_r & 0 \\ -\sin\theta_r & \cos\theta_r & 0 \\ 0 & 0 & 1 \end{bmatrix} \left({}^{\mathtt{L}}\mathbf{p}^T_j - \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} \right) \qquad (3)$$

2) Attention Module:

a) Discussion: Once pooled, point-wise features can be used to compute a single feature vector that represents the entire ROI. A straightforward way is to transform each point feature individually (e.g. via a MLP) and synthesize their information with a permutation invariant operation such as sum, mean or max pooling. However, such a computation disregards valuable information about ROI geometry such as how points are distributed in ROI or ROI's size. To remedy this, we propose to use the attention mechanism to compute ROI feature given pooled point-wise features. The advantage of using attention mechanism is two folds

- Model can dynamically define how much each point feature contributes to ROI feature. This naturally re-

duces the impact of background points while not suppress them entirely. Such a balance can be useful since background points, especially those on ground, can provide context for estimating height.
- Geometry information (e.g. points location, ROI size) can be explicitly injected into the computation via position encoding.

While the original multi-head attention [14] used by ViT [24] and its variants has achieved remarkable successes in the realm of computer vision, it has a drawback of treating every channel equally. In other words, a single set of scalar weights is applied to C-dimension point-wise features in the weighted sum for ROI feature. Since we pool from feature maps generated by backbone made of convolution layers, each channel of any feature maps is a detector for a certain feature [25]. Therefore, using a single set of scalar weights can risk less important features overshadowing important ones. In addition, using multi-head attention can introduce inconsistency since ViT and CNNs learn significantly different features [26]. For the reasons above, we opt for vector attention [15] which has been shown to be effective in 3D classification and segmentation tasks [27].

b) Vector Attention: Essentially, the computation of ROI feature is cross-attention where ROI feature is used to query the set of pooled point-wise features. Let $\mathbf{r}$ be the initial value of ROI feature, and $\mathcal{P}_r = \{({}^r\mathbf{p}_j, \mathbf{f}_j)\}_{j=1}^{N_r}$ be the set of pooled point-wise features. The new ROI feature $\hat{\mathbf{r}}$ is computed as follows

$$\hat{\mathbf{r}} = \sum_{\mathcal{P}_r} \rho\left(\gamma\left(\varphi\left(\mathbf{r}\right) - \psi\left(\mathbf{f}_j\right) + \boldsymbol{\zeta}\right)\right) \odot \left(\alpha\left(\mathbf{f}_j\right) + \boldsymbol{\zeta}\right) \tag{4}$$

Here, φ, ψ, α are linear projection, γ is an MLP, and ρ is the `softmax` operation. $\odot$ denotes the Hadamard product (i.e. element wise multiplication). $\boldsymbol{\zeta}$ represents the position encoding whose detail is presented in the following. In Eq.(4), $\varphi\left(\mathbf{r}\right), \psi\left(\mathbf{f}_j\right), \alpha\left(\mathbf{f}_j\right)$ respectively take the role of query, key, and value.

Using different set of weights for different channels requires storing $N_r \times C$ parameters for computing $\hat{\mathbf{r}}$. As a result, the space complexity of the vector attention is $O\left(MN_rC\right)$ with M is the number of ROI. This effectively makes vector attention more expensive than multi-head attention. However, given that the number of ROIs in the refining stage is relatively small thanks to NMS (100 during testing), vector attention is still affordable on mid-end hardware.

The complete architecture of the Attention Module made of Vector Attention combined with residual connection, normalization layers (BatchNorm by default), and MLP is shown in Figure.3

3) Position Encoding: As mentioned earlier, we exploit position encoding to inject geometry information including points location and ROI size into the attention mechanism. A point j 's location is readily available in its coordinate ${}^r\mathbf{p}_j$ in the ROI canonical frame. To incorporate ROI 's size, we use the approach propose by [13] in which a point 's coordinate is concatenated with its displacement w.r.t ROI's eight vertices.

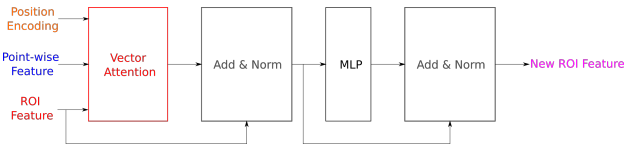

Fig. 3. Architecture of the Attention Module.

$$ {}^r\tilde{\mathbf{p}}_j = \begin{bmatrix} {}^r\mathbf{p}_j & {}^r\mathbf{p}_{j,1} & \dots & {}^r\mathbf{p}_{j,8} \end{bmatrix} \in \mathbb{R}^{1\times 27} \tag{5}$$

In Eq.(5), ${}^r\mathbf{p}_{j,k}(k = \{1, \dots, 8\})$ denotes the vector going from vertex k to ${}^r\mathbf{p}_j$.

Position encoding $\boldsymbol{\zeta}$ used in Eq.(4) is computed by

$$\boldsymbol{\zeta} = \text{MLP}\left({}^r\tilde{\mathbf{c}} - {}^r\tilde{\mathbf{p}}_j\right) \tag{6}$$

where ${}^r\tilde{\mathbf{c}}$ is the result of applying Eq.(5) to ROI's center.

4) Handling Multi-scale Feature Maps: While prior works pool from one fixed source such as raw point cloud [13], a set of sampled points [11], or certain feature maps [10], [12], we propose to pool from every feature map. Our motivation is that each feature map has a different scale thus being helpful for detecting objects of different sizes. For example, low-resolution feature maps can help detect large objects such as cars thanks to their large receptive field. On the other hand, their low resolution results in their high sparsity. Therefore, pooling with small ROI (e.g. ROI of class pedestrians or cyclists) returns significantly low number of points or event empty, making extracting meaningful ROI feature difficult.

Our pooling scheme, illustrated in Alg.1, is inspired by [16] where one feature map is fed to the Transformer decoder at a time. In short, we sequentially pool feature maps from the lowest to the highest resolution. Once a feature map got pooled, ROI features are computed from their associated point-wise features using Eq.(4). This process is repeated N times, each time a different set of RFEs is used, to increase model's depth.

C. Detection Heads and Learning Targets

After being encoded by a series of RFEs, ROI features are mapped to a higher dimension space by a two-hidden layer MLP. The resulted feature vectors are decoded by two separated detection heads having the same architecture to obtain ROIs' confidence and refinement vector.

Following [10], ROIs' confidence is set to the normalized Intersection over Union (IoU) between a ROI and its associated ground truth box, thus making the refinement stage class-agnostic. Let `IoU` denote ROI r's regular IoU, the normalized IoU is defined as

$$c_r^* = \begin{cases} 1 & \text{if } \texttt{IoU} > \chi_H \\ 0 & \text{if } \texttt{IoU} < \chi_L \\ \frac{\texttt{IoU}-\chi_L}{\chi_H-\chi_L} & \text{otherwise} \end{cases} \tag{7}$$

where, χ_H and χ_L are foreground and background threshold.

Algorithm 1: Computing ROI features by pooling from multiple feature maps

Input:
$\mathcal{F}^i$ $(i = \{1, \ldots, 4\})$
: feature maps generated by backbone
$\mathcal{R} = \{([x_r, y_r, z_r, dx_r, dy_r, dz_r, \theta_r], \text{cls}_r)\}_{r=1}^{M}$
: set of ROI generated by RPN
Output: $\mathcal{RF} = \{\mathbf{r}_r\}$: set of ROI features
// Initialize ROI features with model's learnable parameter Θ
$\mathcal{RF} \leftarrow \emptyset$;
for r *in* $\{1, ..., |\mathcal{R}|\}$ **do**
  $\mathbf{r}_r \leftarrow \Theta$;
  $\mathcal{RF} \leftarrow \mathcal{RF} \cup \{\mathbf{r}_r\}$;
end
for N *times* **do**
  for i *in* $\{4, ..., 1\}$ **do**
    for r *in* $\{1, ..., |\mathcal{R}|\}$ **do**
      $\mathcal{P}_r = \{({}^r\mathbf{p}_j, \mathbf{f}_j)\} \leftarrow \texttt{Pool}\left(\mathcal{F}^i\right)$
      $\mathbf{r}_r \leftarrow \texttt{Attention}\left(\mathbf{r}_r, \mathcal{P}_r\right)$; // Eq.(4)
    end
  end
end

The target of the refinement head is the normalized residue of a ROI with respect to its associated ground truth box. Given the parameters of ROI r defined in Eq.(1) and its associated ground truth, its normalized residue $\boldsymbol{\delta}_{\boldsymbol{r}}^* = [x^*, y^*, z^*, dx^*, dy^*, dz^*, \theta^*]$ is

$$\begin{aligned} x^* &= \tfrac{x_r^g - x_r}{d}, \quad & y^* &= \tfrac{y_r^g - y_r}{d}, \quad & z^* &= \tfrac{z_r^g - z_r}{dz_r} \\ dx^* &= \log \tfrac{dx_r^g}{dx_r}, \quad & dy^* &= \log \tfrac{dy_r^g}{dy_r}, \quad & dz^* &= \log \tfrac{dz_r^g}{dz_r} \\ \theta^* &= \theta_r^g - \theta_r \end{aligned} \tag{8}$$

In Eq.(8), the superscript g denotes the ground truth box's parameters, while the subscript r represents ROI index. $d = \sqrt{x_r^2 + y_r^2}$ is the diagonal of the base of the ROI.

D. Loss Function

Our module can be trained an end-to-end fashion with the RPN. The total loss function is the summation of RPN loss, the refinement stage's loss and an auxiliary loss.

$$\mathcal{L} = \mathcal{L}_{\text{RPN}} + \mathcal{L}_{\text{refine}} + \mathcal{L}_{\text{aux}} \tag{9}$$

a) RPN Loss: Since we adopt SECOND's backbone and RPN to generate ROIs, the RPN loss is defined as in [4] which is a summation of classification and regression loss

$$\mathcal{L}_{\text{RPN}} = \frac{1}{A_+} \sum_a \left[\mathcal{L}_{\text{cls}}\left(c_a, c_a^*\right) + \mathbb{1}\left(c_a^* \neq 0\right) \mathcal{L}_{\text{reg}}\left(\delta_a, \delta_a^*\right)\right] \tag{10}$$

where, c_a and δ_a are output of RPN's Class branch and Bbox branch, A_+ is the number of positive anchors. The classification target of the RPN c_a^* is the class of ground truth box of anchor a. The regression target δ_a^* of anchor a is calculated according to Eq.(8). $\mathbb{1}\left(c_a^* \neq 0\right)$ is the indicator function which takes value of 1 for positive anchors whose $c_a^* \neq 0$ and 0 otherwise. The classification loss $\mathcal{L}_{\text{cls}}$ is the Focal Loss [28], while the regression loss $\mathcal{L}_{\text{reg}}$ is the Huber Loss (i.e. smooth-L1).

b) Refining Loss: Similar to RPN loss, this loss is also made up by a classification and a regression loss.

$$\mathcal{L}_{\text{refine}} = \frac{1}{M} \sum_r \left[\mathcal{L}_{\text{cls}}\left(c_r, c_r^*\right) + \mathbb{1}\left(c_r^* \geq \chi_{reg}\right) \mathcal{L}_{\text{reg}}\left(\delta_r, \delta_r^*\right)\right] \tag{11}$$

In the refining loss, the classification target is the normalized IoU (Eq.(7)) and the total loss is normalized over the total number of ROI M. The regression threshold χ_{reg} used in Eq.(11) is different than the foreground threshold χ_H of Eq.(7).

c) Auxiliary Loss: Inspired by [29], [10], an auxiliary supervision is applied to two backbone-generated feature maps $\mathcal{F}^3$ and $\mathcal{F}^4$ to guide the feature extraction process conducted by the 3D backbone. To be specific, $\mathcal{F}^i (i = 3, 4)$ is interpreted to point-wise features. Each point feature k is then fed to an MLP to predict its foreground probability f_k, offset toward associated ground truth box's center o_k, and part probability p_k [10]. A point is labeled as foreground if it is contained in a ground truth bounding box. The label of part probability of foreground points is essentially their coordinate in the canonical frame (Figure.2) of associated ground truth box normalized by the box's sizes.

$$\begin{aligned} \mathcal{L}_{\text{aux}} = \frac{1}{P_+} \left[\sum_{k=1}^{P} \mathcal{L}_{\text{cls}}\left(f_k, f_k^*\right)\right] + \\ \frac{1}{P_+} \left[\sum_{k=1}^{P_+} \mathcal{L}_{\text{reg}}\left(o_k, o_k^*\right) + \mathcal{L}_{\text{bce}}\left(p_k, p_k^*\right)\right] \end{aligned} \tag{12}$$

In Eq.(12), $\mathcal{L}_{\text{bce}}$ is the Binary Cross Entropy (BCE) loss. The $*$ in the superscript denotes the label, while subscript k is the point index. P_+ is the number of foreground points. The regression loss and BCE loss are only calculated for foreground points.

IV. Experiments

To demonstrate the effectiveness of our method, we evaluate it on KITTI [17] and NuScenes [18] dataset. Furthermore, we carry out comprehensive ablation studies to understand the influence of each module on the overall performance.

A. Datasets

The KITTI Dataset contains 7481 samples for training and 7581 samples for testing. Each sample is made of sensory measurements collected by a LiDAR and several cameras. A common practice when working with KITTI is to split the original training data into 3712 training samples and 3769 validation samples for experimental studies. On the other hand, we adjust the training to validation ratio to 4:1 when preparing the submission to the official test sever for benchmarking.

Compared to KITTI, NuScenes is more challenging due to its larger size and requirement of detecting more classes. Specifically, NuScenes offers 28130 training and 6019 validation samples. Each sample, or keyframe, comprises of data collected by one LiDAR, 6 cameras, and 5 radars when they are in sync. Annotation is provided for 23 object classes among which 10 are targeted by the detection task.

B. Implementation Details

We build our method to work on top of SECOND (or other 3D proposals methods). For efficiency, we use the implementation of SECOND as well as other RPNs provided by the OpenPCDet [30] toolbox. To demonstrate the robustness of our choice of model's hyperparameters, we keep them constant for experiments on both KITTI and NuScenes. The three exceptions are the point cloud range, the initial voxel size and the number of channels of the last feature map produced by the 3D backbone $\mathcal{F}^4$.

1) RPN: Since we don't introduce any modification to SECOND, the following presents the parameters that are directly related to our method. Omitted parameters can found in OpenPCDet [30]. In KITTI, the point cloud is clipped by the range of $[0\text{m}, 70.4\text{m}]$ in the X-axis, $[-40\text{m}, 40\text{m}]$ in the Y-axis, and $[-3\text{m}, 1\text{m}]$ in the Z-axis and voxelized with grid size of $[0.05\text{m}, 0.05\text{m}, 0.1\text{m}]$. Along with a set of proposals, SECOND outputs 4 feature maps having 16, 32, 64, 64 channels respectively. This set of proposals is post-processed by NMS with the overlapped threshold of 0.8 or 0.7 to obtain 512 or 100 ROI during training or testing.

In NuScenes, the point cloud range is set to $[-51.2\text{m}, 51.2\text{m}]$ along X and Y-axis, and $[-5\text{m}, 3\text{m}$ along Z-axis. The voxel size for discretizing the input point cloud is $[0.1\text{m}, 0.1\text{m}, 0.2\text{m}]$. The point cloud of a NuScenes keyframe (i.e. sample) contains about 40k points which is just one third the size of a KITTI point cloud, thus making it highly difficult for any methods. We follow the common practice which maps point cloud in 10 previous non-keyframe to the timestamp of the keyframe using ego vehicle's odometry to increase the number of points by 10 times. Regarding the 3D backbone, the number of channels in the last feature map $\mathcal{F}^4$ is 128, while the rest are similar to KITTI configuration.

2) ROI Feature Encoder: In KITTI, 128 ROIs are sampled from RPN output for the refinement stage during training. Each ROI is then enlarged by $0.5m$ along three dimensions for pooling. We empirically find that the pooling from second feature map $\mathcal{F}^2$ does not bring significant improvement to the final performance. Therefore, we opt for pooling from $\left\{\mathcal{F}^4, \mathcal{F}^3, \mathcal{F}^1\right\}$ with the number of pooled points per ROI being set to $64, 128, 256$, respectively.

Throughout the Attention Module, the feature dimension is kept constant at d_a which is set to 128. Features pooled from $\left\{\mathcal{F}^4, \mathcal{F}^3, \mathcal{F}^1\right\}$ are linearly mapped to d_a prior to being passed to the Vector Attention. The MLP of the Attention Module has a hidden layer made of 256 neurons. The nonlinearity of this MLP is the ReLU function. Position Encoding uses an MLP of the same architecture. The process of sequentially pooling from $\mathcal{F}^4$ to $\mathcal{F}^1$ for computing ROI features is repeated 3 times. Each time, a different set of RFEs is used.

3) Training: The entire architecture presented in Figure.1 is optimized end-to-end by the Adam optimizer. For KITTI, we train the model for 100 epochs with the total batch size of 24. The learning rate is set according to the one-cycle policy [31] with 0.01 maximum learning rate. For NuScenes, the model is trained for 20 epochs with the same batch size. The learning rate is also modulated by the one-cycle policy with 0.03 maximum learning rate. In the detection head, the foreground χ_H, background χ_L and regression χ_{reg} IoU threshold are $0.75, 0.25$ and 0.55 for both datasets. We use the same data augmentation strategy of [4], [11], [10].

C. Results On KITTI Dataset

The detection task of KITTI dataset concerns 3 classes: Car, Pedestrian and Cyclist. The evaluation is based the Average Precision (AP) metric computed at 40 recall position[2] with IoU threshold 0.7 for cars and 0.5 for others. The comparison of our method against the state-of-the-art on KITTI *test* set is presented in Table.I. Among methods that built on top SECOND namely [29], [11], [12], [13], we achieve the best performance in class Cyclist and competitive performance in class Car, passing the 80 AP threshold, while having the least number of parameters. Note that we train a single model for two classes instead of separate models for each class as previously done by [4], [5], [6].

Our performance on KITTI *val* set with AP calculated at 40 recall position is also reported in Table.II which shows that the gap between our method and top performers in class Car is significantly narrowed down with the largest difference being just 0.45 AP. Compared to CT3D which also computes ROI feature using the attention mechanism, we surpass their AP for class Pedestrian and Cyclist by 1.42 and 1.47.

D. Results On NuScenes Dataset

The main metric used by NuScenes in the Average Precision (AP) score which is computed as the normalized area under precision-recall curve with the minimum recall is set at 0.1. Instead of using IoU as matching criteria like KITTI, NuScenes use the euclidean distance between the center of a predicted boxes and ground truth. The performance on NuScenes *validation* set is shown in Table.III. In this table, SECOND and PointPillars are single-stage methods while the others including ours are two-stage. In this more challenging method, the benefit of integrating our RFE to SECOND is more prominent, indicated by almost 20 mAP improvement. In addition, ours outperforms 3DSSD [8] and InfoFocus [32], two recent two-stage methods, by a large margin, except for class Car and Truck. This competitive performance on NuScenes dataset shows our method's ability of handling different object classes with large variance on scale.

[2]Since 08.10.2019, the number of recall position for computing AP has been increased from 11 to 40.

TABLE I

PERFORMANCE COMPARISON ON KITTI *test* SET WITH AP CALCULATED WITH 40 RECALL POSITIONS

Method	Num Parameters (M)	Car - 3D Detection			Cyclist - 3D Detection		
		Easy	Mod.	Hard	Easy	Mod.	Hard
SECOND [4]	20	83.34	72.55	65.82	71.33	52.08	45.83
PointPillar [5]	18	82.58	74.31	68.99	77.10	58.65	51.92
PointRCNN [6]	16	86.96	75.64	70.70	74.96	58.82	52.53
SA-SSD [29]	226	88.75	79.79	74.16	-	-	-
Part A^2 [10]	40.8	87.81	78.49	73.51	-	-	-
PV-RCNN [11]	50	90.25	81.43	76.82	**78.60**	63.71	57.65
Voxel R-CNN [12]	28	**90.90**	81.62	77.06	-	-	-
CT3D [13]	30	87.83	**81.77**	**77.16**	-	-	-
APRO3D-Net (ours)	22.4	87.09	80.30	76.10	78.54	**64.55**	**57.78**

TABLE II

PERFORMANCE COMPARISON ON KITTI *val* SET WITH AP CALCULATED AT 40 RECALL POSITIONS

Method	AP_{3D} - Moderate		
	Car	Pedestrian	Cyclist
PV-RCNN [23]	84.83	56.67	71.95
Voxel R-CNN [12]	**85.29**	-	-
Voxel R-CNN [3]	84.95	**58.24**	71.43
CT3D [13]	84.97	55.58	71.88
APRO3D-Net (ours)	84.84	57.00	**73.35**

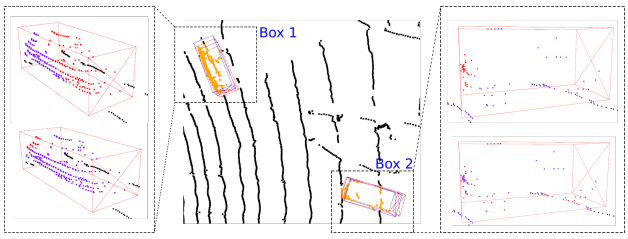

Fig. 4. Visualization of attention weights. Predictions and their associated ground truth boxes are respectively marked by red and blue. Orange denotes pooled points. In two zoomed window, points are color coded according to their attention weights. The hotter the color, the higher attention weight.

E. Qualitative Performance

To shows that different channels within the same point feature contribute differently to ROI feature, we visualize pooled points' attention weight (the term on the left of $\odot$ in Eq.(4)) in Figure.4. Evidently, the region of ROI where attention is concentrated varies across channels. For example, Box 1 respectively pays most attention to its front and rear to compute two different channels. In addition, visual evaluation of our method's performance on the *test* split of KITTI and NuScenes dataset made by projecting predictions onto images as in Figure.5 shows good result.

F. Ablation studies

To validate our design choices as well as to understand the impact of each module on the overall performance, we perform extensive ablation studies. All models used in this section are trained on KITTI *train* set and evaluated on KITTI *val* set. Unless stating otherwise, evaluations are based on AP calculated at 40 recall precision.

[3]Performance of model trained for 3 classes, reproduced from official code release

First, we verify our motivation for choosing vector attention over multi-head attention by changing the attention formula in Eq.(4) while keeping the rest of the architecture unchanged. The result shown in Table.IV confirms the superiority of vector attention with significant AP difference for class Car and Cyclist. The reason for this is vector attention enables model to choose where to look (which points) and what to look for (which channels) when computing ROI features. On the other hand, multi-head attention can only choose where to look due to its scalar weight.

The second experiment is to analyze the impact of position encoding on the overall performance. The first row of Table.V shows the result of the model which does not use position encoding that is to set ζ in Eq.(4) to 0. The second row is the performance when building position encoding from the point displacement relative to ROI center only. In other words, ${}^r\mathbf{p}_{j,1}, \ldots, {}^r\mathbf{p}_{j,8}$ are removed from Eq.(5). As can be seen from Table.V, compared to not using position encoding (first row), performance can increase up to 8.19, 6.24, 4.03 AP for class Car, Cyclist and Pedestrian. Moreover, position encoding contains the most information about ROI geometry (third row) performs the best in overall, especially in the most important class Car.

Next, three pooling strategies are compared. The performance shown in the first row of Table.VI is obtained by equally pooling M points from each feature map $\mathcal{F}^i$ then concatenating their features before passing to RFE for ROI feature computation. In the other rows, we sequentially pool from the feature map having the lowest resolution $\mathcal{F}^4$ to the highest one $\mathcal{F}^1$. The difference between second and third row is the sequential pooling process is not repeated in the second row while it is repeated three times in the third. In other words, N of Alg.1 is set to 1 and 3 in row second and third, respectively. Even though pooling all at once (first row) does not show any significant performance drop in class Car while achieves the best performance in class Pedestrian, this pooling strategy is the most memory intensive. The reason is the number of points to be processed is scaled by the number of pooled feature maps. This compounds with number of ROI can quickly overflow GPUs' memory. Another aspect to be noticed is repeating the pooling process (with a different set of RFE) only bring marginal performance gain.

Finally, the versatility of our method is demonstrated by its integreation with different RPNs: SECOND, PartA^2,

TABLE III

AP ON NUSCENES DATASET

Method	Car	Ped	Bus	Barrier	Traf. Cone	Truck	Trailer	Motor	Cons. Veh.	Bicycle	mAP
SECOND [4]	75.53	59.86	29.04	32.21	22.49	21.88	12.96	16.89	0.36	0	27.12
PointPillars [5]	70.5	59.9	34.4	33.2	29.6	25.0	20.0	16.7	4.5	1.6	29.5
3DSSD [8]	**81.20**	70.17	61.41	47.94	31.06	**47.15**	30.45	35.96	12.64	8.63	42.66
InfoFocus [32]	77.6	61.7	50.5	43.4	33.4	35.4	25.6	25.2	8.3	2.5	36.4
APRO3D-Net (ours)	77.75	**74.02**	**64.86**	**52.61**	**46.34**	43.99	**34.9**	**39.36**	**13.44**	**23.00**	**47.03**

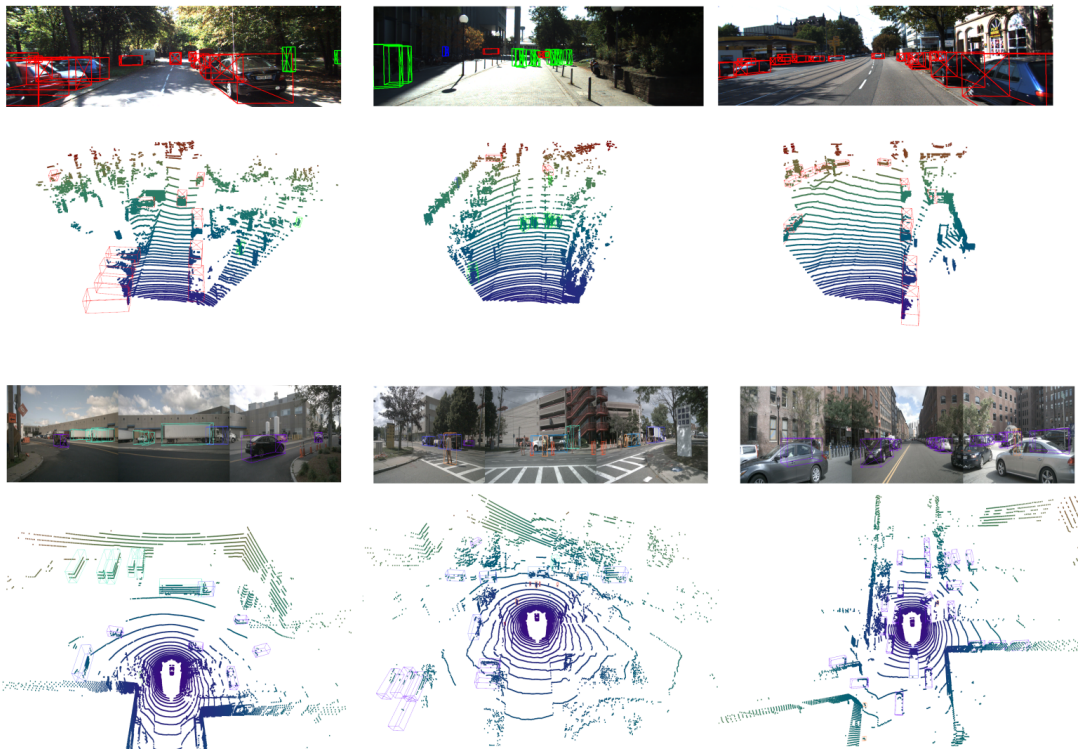

Fig. 5. Visualization of prediction made by APRO3D-Net on the *test* split of KITTI (upper row) and NuScenes (lower row) dataset.

TABLE IV

PERFORMANCE OF MULTI-HEAD ATTENTION COMPARED TO VECTOR ATTENTION

Method	3D Detection - Moderate		
	Car	Cyclist	Pedestrian
Multi-head Attention	82.50	70.35	57.58
Vector Attention	84.85	73.35	57.41
Improvement	**2.35**	**3.00**	-0.17

TABLE V

PERFORMANCE OF DIFFERENT POSITION ENCODING METHODS

Method	AP_{3D} - Moderate		
	Car	Cyclist	Pedestrian
None	76.66	69.65	53.38
Center	82.72	**75.89**	55.74
Center and Vertices	**84.85**	73.35	**57.41**

PointRCNN. Since PartA^2 has a UNet-like backbone, we pool from feature maps produced by its up-sampling branch while keeping the rest of the architecture unchanged. In the case of PointRCNN, we pool from the final output of its backbone which is a set of point-wise features and repeat for 4 times (each time with a different RFE). Note that when not using RFE, PointRCNN and PartA^2 use their own refinement stage. Table.VII shows the performance gain in different level of difficulty of class Car confirming the effectiveness of our method. The limited gain when integrating with PointRCNN can be explained by its lower recall rate compared to SECOND. This confirm our design choice regarding RPN method.

TABLE VI

PERFORMANCE OF DIFFERENT POOLING METHODS

Method	3D Detection - Moderate		
	Car	Cyclist	Pedestrian
All at once	84.15	71.04	**57.55**
Sequential without repetition	84.66	**75.34**	56.15
Sequential with repetition	**84.85**	73.35	57.41

V. CONCLUSION

In conclusion, this paper develops a proposals refinement stage for 3D object detection. The core of this stage is the RFE module which transforms pooled features to ROI feature using Vector Attention. In addition, we propose a

TABLE VII

PERFORMANCE GAIN BROUGHT BY RFE TO DIFFERENT RPNS

Method	Car - 3D Detection		
	Easy	Moderate	Hard
SECOND	88.61	78.62	77.22
SECOND + RFE	**+0.75**	**+4.89**	**+1.56**
PartA^2	89.47	79.47	78.54
PartA^2 + RFE	-0.08	**+3.16**	**+0.36**
PointRCNN	88.88	78.63	77.38
PointRCNN + RFE	**+0.06**	**+0.38**	**+1.02**

pooling strategy that effectively fuses multi-scale features extracted by the 3D backbone, thus increasing model's ability to handle objects of different sizes. Experiments on KITTI and NuScenes dataset validate the effectiveness of our method. As for future work, we would like to extend the RFE module to enable fusion of LiDAR with other sensing modalities such as cameras or radars. Another possibility is to explore how ROI features extracted by RFEs can be used for tracking.

ACKNOWLEDGMENT

This work was granted access to the HPC resources of IDRIS under the allocation 2021-AD011012128R1 made by GENCI. This work has been supported in part by the ANR AIby4 under the number ANR-20-THIA-0011.

REFERENCES

[1] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[2] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 7652–7660.

[3] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499.

[4] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[5] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[6] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*,, 2019, pp. 770–779.

[7] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "Std: Sparse-to-dense 3d object detector for point cloud," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1951–1960.

[8] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3dssd: Point-based 3d single stage object detector," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 040–11 048.

[9] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[10] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, "From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network," *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[11] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 529–10 538.

[12] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li, "Voxel r-cnn: Towards high performance voxel-based 3d object detection," *arXiv preprint arXiv:2012.15712*, 2020.

[13] H. Sheng, S. Cai, Y. Liu, B. Deng, J. Huang, X.-S. Hua, and M.-J. Zhao, "Improving 3d object detection with channel-wise transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2743–2752.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[15] H. Zhao, J. Jia, and V. Koltun, "Exploring self-attention for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 076–10 085.

[16] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, "Masked-attention mask transformer for universal image segmentation," *arXiv preprint arXiv:2112.01527*, 2021.

[17] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[18] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[19] J. Mao, Y. Xue, M. Niu, H. Bai, J. Feng, X. Liang, H. Xu, and C. Xu, "Voxel transformer for 3d object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 3164–3173.

[20] X. Pan, Z. Xia, S. Song, L. E. Li, and G. Huang, "3d object detection with pointformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7463–7472.

[21] I. Misra, R. Girdhar, and A. Joulin, "An end-to-end transformer model for 3d object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2906–2917.

[22] L. Fan, Z. Pang, T. Zhang, Y.-X. Wang, H. Zhao, F. Wang, N. Wang, and Z. Zhang, "Embracing single stride 3d object detector with sparse transformer," *arXiv preprint arXiv:2112.06375*, 2021.

[23] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," in *Advances in Neural Information Processing Systems*, 2019.

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

[25] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[26] M. Raghu, T. Unterthiner, S. Kornblith, C. Zhang, and A. Dosovitskiy, "Do vision transformers see like convolutional neural networks?" *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[27] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16 259–16 268.

[28] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[29] C. He, H. Zeng, J. Huang, X.-S. Hua, and L. Zhang, "Structure aware single-stage 3d object detection from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 873–11 882.

[30] O. D. Team, "Openpcdet: An open-source toolbox for 3d object detection from point clouds," https://github.com/open-mmlab/OpenPCDet, 2020.

[31] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006. International Society for Optics and Photonics, 2019, p. 1100612.

[32] J. Wang, S. Lan, M. Gao, and L. S. Davis, "Infofocus: 3d object detection for autonomous driving with dynamic information modeling," in *European Conference on Computer Vision*. Springer, 2020, pp. 405–420.