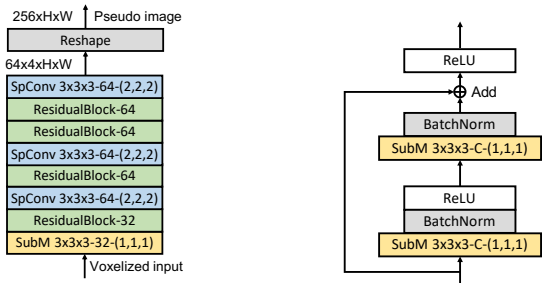
1st Place Solutions to the Real-time 3D Detection and the Most Efficient Model of the Waymo Open Dataset Challenges 2021

Runzhou Ge* Zhuangzhuang Ding* Yihan Hu*
Wenxin Shao Li Huang Kun Li Qiang Liu
Horizon Robotics
{runzhouge, dinghouzx, yihan.hu96}@gmail.com

Abstract

In this report, we introduce our winning solution to the Real-time 3D Detection and also the "Most Efficient Model" in the Waymo Open Dataset Challenges at CVPR 2021. Extended from our last year's award-winning model AFDet, we have made a handful of modifications to the base model, to improve the accuracy and at the same time to greatly reduce the latency. The modified model, named as AFDetV2, is featured with a lite 3D Feature Extractor, an improved RPN with extended receptive field and an added sub-head that produces an IoU-aware confidence score. These model enhancements, together with enriched data augmentation, stochastic weights averaging, and a GPU-based implementation of voxelization, lead to a winning accuracy of 73.12 mAPH/L2 for our AFDetV2 with a latency of 60.06 ms, and an accuracy of 72.57 mAPH/L2 for our AFDetV2-base, entitled as the "Most Efficient Model" by the challenge sponsor, with a winning latency of 55.86 ms.



(a) 3D Feature Extractor (b) Residual block

Figure 1: (a) The structure of the lite 3D Feature Extractor. 'SubM' stands for sub-manifold sparse convolutional layer [5] and 'SpConv' stands for 3D sparse convolutional layer [22]. The format of the layer setting follows 'kernel size-channels-(strides)', *i.e.* $k_W \times k_H \times k_D$-$C$-($s_H$,$s_W$,$s_D$). For residual blocks, only channels are shown. After the backbone, channel dimension and D dimension are reshaped together to form a pseudo image. (b) The structure of a residual block with C channels.

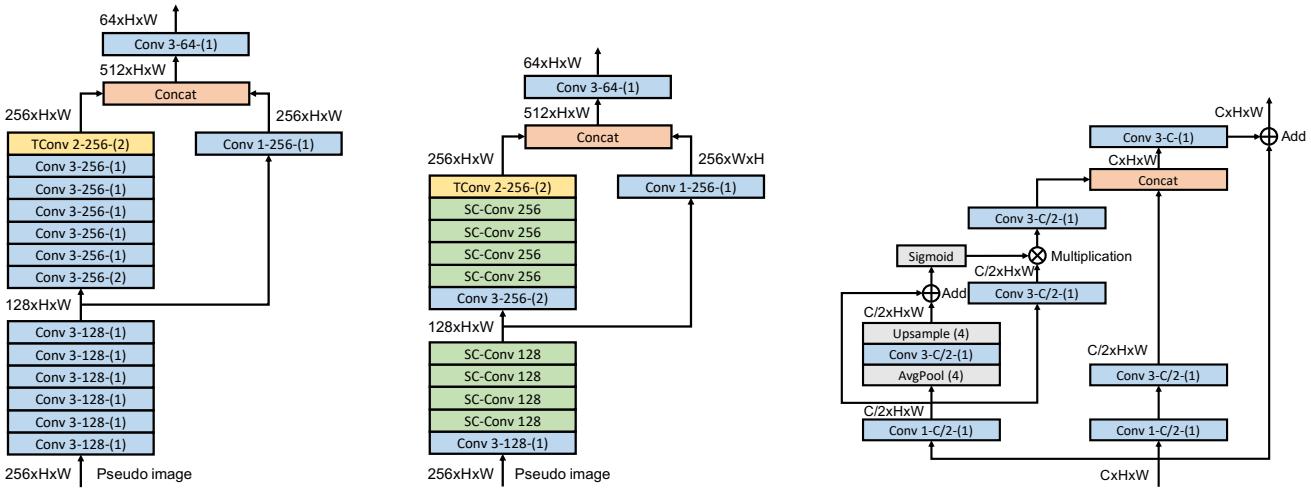
1. Introduction

The Waymo Open Dataset Challenges at CVPR 2021 are the most exciting competitions in the field of autonomous driving. The Waymo Open Dataset [18] and the Waymo Open Motion Dataset [3] are datasets with high-quality data collected from both LiDAR and camera sensors in real self-driving scenarios, which have enabled many new exciting research. The Real-time 3D Detection challenge requires an algorithm to detect the 3D objects of interest as a set of 3D bounding boxes within 70 ms per frame. The model with the highest performance (in mAPH/L2) while satisfying this real-time requirement wins this challenge. The model with the lowest latency and mAPH/L2 > 70 is given the title of "Most Efficient Model".

*These authors contributed equally to this work.

2. Methods

In this section, we present the details of our network used in the challenge. The overall network structure follows our previous work AFDet [4], which is a one-stage and anchor-free 3D point cloud detector. The whole network consists of four modules: point cloud voxelization, 3D Feature Extractor, Region Proposal Networks (RPN) and anchor-free detector heads. We have made significant improvements to the model structure to deal with the new requirements of this challenge, especially the low latency requirement. We explored each module and its structure thoroughly in order to get the best performance while keeping the model structure simple.



(a) RPN baseline (b) SC-Conv RPN (c) SC-Conv module

Figure 2: (a) and (b) denote the RPN baseline and the self-calibrated convolution (SC-Conv) RPN. 'Conv' stands for convolutional layer and 'TConv' stands for transposed convolutional layer. The format of the layer setting follows 'kernel size-channels-(strides)', *i.e.* k-C-(s). 'SC-Conv' stands for SC-Conv module and only channels are shown. (c) gives the detailed structure of the SC-Conv module.

2.1. Point Cloud Voxelization

Point cloud input is first voxelized [26, 27] into small voxels across x, y, z dimensions. For the voxelization, a fixed grid size and a fixed range are set so that all voxels positions are fixed. For a point in the point cloud, it is assigned to a certain voxel based on its coordinates. Then, in each voxel, we calculate the mean of all points assigned to it and use this mean value as the representative value for that voxel. Finally, a voxelized 3D input is generated and is sent to the 3D feature extractor.

2.2. 3D Feature Extractor

In this part, 3D convolutional layers are used to extract features from voxelized 3D inputs. We use 3D sparse convolutional layer [22] and submanifold sparse convolutional layer [5] to build our feature extractor.

We propose a lite version of the 3D Feature Extractor. It has fewer residual blocks at the early stage but slightly more channels for each residual block. It only down-samples z-axis dimension with a factor of 8. The channel dimension and the z-axis dimension are reshaped and combined together after the 3D feature extractor. In order to keep the same shape of the output feature map, the number of channels of the feature extraction layers at the final down-sampled scale is decreased accordingly, which also significantly accelerates the 3D Feature Extractor. The detailed structure of the 3D Feature Extractor is shown in Fig. 1. The resulting feature map is reshaped to form a Bird's Eye View (BEV) pseudo image.

2.3. Region Proposal Networks

The Region Proposal Network (RPN) takes a pseudo image as input and employs multiple down-sample and up-sample blocks to output stride 8 feature maps, so the final down-sample factor is 1 for the RPN. The RPN baseline structure is shown in Fig. 2a. We replace the basic 3×3 convolutional blocks with self-calibrated convolutions (SC-Conv) [13] as shown in Fig. 2b and 2c, which helps to enlarge the receptive field for spatial locations and introduces channel-wise and spatial-wise attention with cost-efficiency. The improved RPN structure boosts the detection accuracy with a similar number of parameters and computational costs as the baseline.

2.4. Anchor-free Head

In addition to the five sub-heads in AFDet [4], we devise 2 new sub-heads in our AFDetV2 to achieve better accuracy. The 5 sub-heads common to both AFDet and AFDetV2 are the heatmap prediction head, the local offset regression head, the z-axis location regression head, the 3D object size regression head and the orientation regression head. The 2 new sub-heads are IoU-aware confidence score prediction and keypoints auxiliary supervision, as shown in Fig. 3. First we will briefly introduce the differences of the shared 5 sub-heads between the AFDet [4] and AFDetV2. Then we will describe the newly devised sub-heads.

Differences of the 5 sub-heads. For the heatmap head, we enlarge the positive supervision for the heatmap prediction by setting a minimum allowed Gaussian radius [11] to

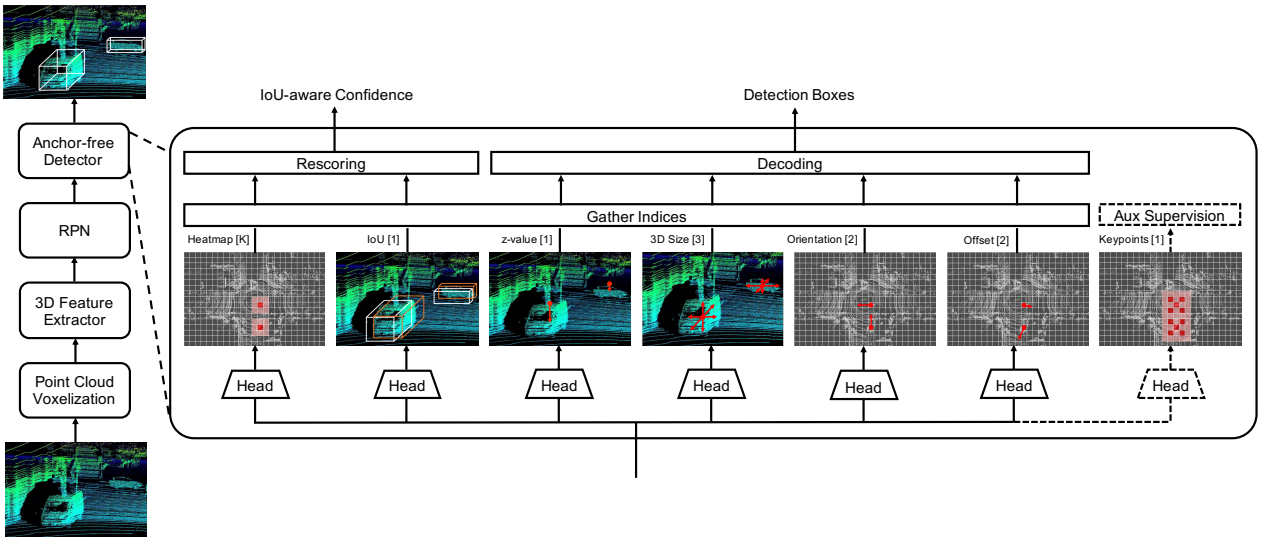


Figure 3: The framework of anchor-free one-stage 3D detection (AFDetV2) system. The whole pipeline consists of the Point Cloud Voxelization, 3D Feature Extractor, Region Proposal Networks (RPN) and the Anchor-free Detector. The number in the brackets indicates the output channels in the last convolution layer. K is the number of categories used in the detection. The auxiliary supervision which is turned off at inference is shown in dashed lines. Better viewed in color and zoom in for more details.

2 following [23]. For the orientation regression sub-head, we set the regression target to the sin and cos values of the yaw angle of the objects.

IoU-aware confidence score prediction. In the task of object detection, the classification score is commonly used as the final prediction score. However, it is not a good estimate of the localization accuracy. In this case, boxes with high localization accuracy but low classification scores may be deleted after Non-Maximum Suppression (NMS). Also, the misalignment harms the ranking-based metrics such as Average Precision (AP). To alleviate the misalignment, most of the existing methods adopt an IoU-aware prediction branch as the second stage network [10, 15, 23]. However, an additional stage will bring more computational cost and latency to the network. Also, special operator such as RoI Align [7, 23] or RoI pooling [16, 15, 17] is required in the second stage network.

In our solution, we adopt an IoU prediction head following [21, 25]. To incorporate IoU information into confidence score, we recalculate final confidence score by a post-processing function:

$$f = score^{1-\alpha} * iou^{\alpha} \tag{1}$$

where $score$ is the original classification score, iou is the predicted IoU and α is a parameter $\in [0, 1]$ that controls the contributions from the classification score and predicted IoU. In our solution, we use 0.68, 0.71, 0.65 for VEHICLE, PEDESTRIAN and CYCLIST respectively.

After rescoring, the ranking of the predictions takes both the classification confidence and localization accuracy into account. It will lower the confidence of the predictions with higher classification scores but worse localization accuracy, and vice versa. Our single-stage network runs much faster than most existing two-stage LiDAR detectors while surpassing their detection results, as shown in the leader board in Tab. 4.

Keypoints auxiliary supervision. Inspired by the corner classification in [19], we add a sub-head of keypoint prediction as an auxiliary supervision in the detection. Specifically, another heatmap that predicts 4 corners and the center of every object in BEV is added during training. We use the same method to draw the objects on the heatmap as the heatmap prediction sub-head but with a halved radius and minimum Gaussian radius [11] set to 1. It should be noted that this sub-head does not affect the inference speed as the sub-head is disabled at inference.

2.5. Loss

Following [4, 2, 20], we use the focal loss [12, 11, 23] for the heatmap prediction and keypoints auxiliary supervision. L_1 loss is employed in the local offset head, the z-axis location head, the 3D object size head, and orientation regression. For IoU prediction branch, we encode the target IoU as $2 * iou - 0.5 \in [-1, 1]$ following [25], where iou is the 3D axis-aligned IoU between the predicted box and associated ground truth box. Then smooth L_1 loss is em-

ployed to regress the encoded IoU.

The overall training objective is

$$\mathcal{L} = \mathcal{L}_{heat} + \lambda_{off}\mathcal{L}_{off} + \lambda_z\mathcal{L}_z + \lambda_{size}\mathcal{L}_{size} + \lambda_{ori}\mathcal{L}_{ori} + \lambda_{iou}\mathcal{L}_{iou} + \lambda_{kps}\mathcal{L}_{kps} \quad (2)$$

where λ represents the weight for each sub-task. For all regression sub-heads including local offset, z-axis location, 3D size, orientation, and IoU prediction, we only regress N foreground objects that are in the detection range.

3. Experiments

3.1. Point Clouds Densification and Data Inputs

Accumulating LiDAR sweeps is a simple but effective method to utilize temporal information and densify LiDAR point cloud [1, 2]. To distinguish points from different sweeps, the time lag Δt is attached to the point cloud as an additional attribute. In our solution, we use the past one frame combined with the current frame as our input point cloud.

Waymo Open Dataset [18] provides five types of LiDAR sweeps. To reduce the latency, we only utilize the first and second returns of the top LiDAR. Specifically, considering frame densification, our input format should be $(x, y, z, \mathit{reflectance}, \mathit{elongation}, \Delta t)$, while Δt being the time lag as mentioned above.

3.2. Data Augmentation

We use data augmentation strategy following [22, 27, 2]. First, we generate an annotation database containing labels and the associated point cloud data. During training, we randomly select 15, 10 and 10 ground truth samples for vehicle, pedestrian and cyclist respectively, and place them into the current frame. Second, we do random flipping along x, y-axis, global rotation with a uniform distribution of $\mathcal{U}\left(-\frac{\pi}{4}, \frac{\pi}{4}\right)$, global scaling of $\mathcal{U}\left(0.95, 1.05\right)$ and global translation along x, y, z-axis of $\mathcal{U}\left(-0.2m, 0.2m\right)$. Third, random rotation noise with a uniform distribution of $\mathcal{U}\left(-\frac{\pi}{20}, \frac{\pi}{20}\right)$ and random location noise with a Gaussian distribution $\mathcal{N}\left(0.0,\ 0.1\right)$ are applied to each instance.

3.3. Stochastic Weights Averaging

Stochastic Weights Averaging (SWA) is developed in [9] for improving generalization in deep networks. Inspired by VarifocalNet [24], we equipped SWA to our model and achieved ∼0.2 mAPH/L2 improvement as shown in Tab. 1. Specifically, we trained 5 extra epochs with 1/10 of the original learning rate. We adopted a cyclical learning rate scheduler with one-cycle policy [6]. In each epoch, after a warming up from 3×10^{-5} to 3×10^{-4}, the learning rate decreases at each iteration from 3×10^{-4} to 3×10^{-9}. Finally, we calculated the average of 5 checkpoints and trained an extra epoch to fix the Batch Normalization [8] parameters.

Checkpoints	Vehicle	Pedestrian	Cyclist	All
Baseline	70.26	71.17	72.20	71.21
Epoch 1	70.38	71.70	71.72	71.27
Epoch 2	70.47	71.22	72.23	71.31
Epoch 3	70.45	71.70	71.75	71.30
Epoch 4	70.41	71.74	71.82	71.32
Epoch 5	70.46	71.75	71.92	71.38
After SWA	70.54	71.72	72.00	71.42

Table 1: Improvement of model's performance after SWA. Performance of each SWA checkpoints are also listed. Values in this table are results of the mAPH/L2 evaluation metric. Models are tested on 1/5 validation set.

3.4. Improvements on Latency

To reduce the inference latency, we exploited GPU on the voxelization and the conversion of range data to point features.

Voxelization with GPU. Voxelization is composed of two processes: first, each point is quantized with the grid cell size (g_x, g_y, g_z) to the corresponding voxel (c_x, c_y, c_z), where $c_x = \lfloor x/g_x \rfloor, c_y = \lfloor y/g_y \rfloor, c_z = \lfloor z/g_z \rfloor$ are the voxel coordinates, and $\lfloor \cdot \rfloor$ denotes the $floor()$ operation; and second, an average of all the points in a voxel is calculated to form the feature vector for each voxel, *i.e.* $F_i(d) = \frac{1}{N} \sum_{k \in V_i} P_k(d), d \in D$, where P_k is a point, V_i is the corresponding voxel, and D is the dimension of the feature vector.

To accelerate voxelization, we implemented both processes in GPU, as follows:

(1) As an initialization step, we allocate two buffers in the GPU memory, one for the feature values in dense voxels in a grid of (v_x, v_y, v_z), denoted as F_g, and the other for the count of points in a voxel, also in the same grid, denoted as N_g.

(2) For each point P_i, we calculate the corresponding voxel coordinate (c_x, c_y, c_z) and do the atomic accumulation in $F_g(c_x, c_y, c_z)$ and $N_g(c_x, c_y, c_z)$. At the same time, we store the voxel index i_v for P_i to a list I_p, where $i_v = c_x + c_y \cdot v_x + c_z \cdot v_x \cdot y_y$. It should be noted that in this implementation, we do not limit the number of points per voxel.

(3) Transfer the voxel index list I_p from GPU memory to host memory. Note that the size of I_p is the number of points in the given point cloud.

(4) Using CPU, we remove the duplicates of voxel index in I_p and obtain a list of unique voxel index, denoted as I_u.

(5) Transfer I_u from CPU memory to GPU. Note that $[I_u] \leqslant [I_p]$, where $[\cdot]$ denotes the size of a set.

(6) Gather the cumulative features and the counts in F_g

and N_g respectively on GPU, and calculate F_i for each individual voxel indexed in I_u.

Range data conversion with GPU. A range data R_i is a vector containing $(range, intensity, elongation, x, y, z)$. We convert it to a point P_i containing $(x, y, z, intensity_{clamp}, elongation)$ before voxelization, where $intensity_{clamp}$ is $intensity$ clamped at 1.5. For the case of densification with previous frames, P_i contains $(x, y, z, intensity_{clamp}, elongation, \Delta t)$ where Δt is the time interval between the current frame and the previous frame.

To convert an R_i to P_i, we utilize both CPU and GPU in the following steps:

(1) Exclude the invalid R_i by checking the $range$ value. This is done on CPU.

(2) Transfer the valid range data to GPU memory. For densification with previous frames, also transfer the transform matrices (*i.e*. the extrinsics).

(3) On GPU, for each R_i, permute the entries as described above, and apply the extrinsics to the (x, y, z) values of the previous frames for densification.

3.5. Experiment Settings

Only the point clouds from the top LiDAR were used during training and inference for the purpose of acceleration. The maximum number of objects was set to 500. For all of our models, we set the max point per voxel to 15, max voxel num to $300,000$ during training. We did not set a limit for the max number of points at inference. We used AdamW [14] optimizer with one-cycle policy [6]. We set the learning rate max to 3×10^{-3}, division factor to 10, momentum ranges from 0.95 to 0.85, fixed weight decay to 0.01 to achieve convergence. we set all λ in Eqn. 2 to 2.0.

To quickly verify our idea, we sampled the validation set every 5 frames according to their timestamps. Unless explicitly indicated, all models used for the ablation studies, including Tab. 1 and Tab. 3, were tested on 1/5 validation data.

AFDetV2-Base and AFDetV2-Lite were trained and tested within range [(-75.2, 75.2), (-75.2, 75.2), (-2, 4)] in respect of x, y, z-axis as shown in Tab. 2. AFDetV2 was trained within range [(-75.2, 75.2), (-73.6, 73.6), (-2, 4)], while enlarging the detection range to [(-80, 80), (-76.16, 76.16), (-2, 4)] at inference. For AFDetV2-Base, we first trained 10 epochs on training data and finetuned 36 epochs on the whole trainval data with [0.1, 0.1, 0.15] grid size with respect to the x, y, z-axis. For AFDetV2, we adjusted the grid size to [0.1, 0.08, 0.15] and finetuned another 36 epochs on the whole trainval set based on AFDetV2-Base. Two frames of point clouds were utilized for densification in AFDetV2-Base and AFDetV2, while AFDetV2-Lite only exploited a single frame of the point cloud. AFDetV2-Lite was only finetuned 18 epochs on trainval set after being trained on training data for 10 epochs with grid size [0.1, 0.1, 0.15]. AFDetV2-Base and AFDetV2-Lite were trained with 10 samples per GPU while AFDetV2 was trained with 8 samples per GPU. For the offset regression head, we set the regression radius to 0. Besides, we replaced max pooling with class-specific NMS for better average precision. In our solution, we set IoU threshold to 0.8, 0.55, 0.55 for VEHICLE, PEDESTRIAN and CYCLIST respectively. Models were trained with Nvidia V100 GPUs.

To diminish latency, the following improvements are adopted at inference:

(1) Range images were converted to point clouds in Cartesian coordinate on GPU.

(2) Voxelization process was executed with GPU.

(3) All layers were cast to half-precision (FP16) except for the last layers in each sub-heads.

(4) Batch Normalization [8] parameters were merged into 3D Sparse Convolution and SubManifold 3D Sparse Convolution layers in 3D Feature Extractor.

(5) Keypoints auxiliary branch was disabled.

Models	Num. of Frames	Training Range		Inference Range		Grid Size	
AFDetV2	2	75.2	73.6	80.0	76.16	0.1	0.08
AFDetV2-Base	2	75.2	75.2	75.2	75.2	0.1	0.1
AFDetV2-Lite	1	75.2	75.2	75.2	75.2	0.1	0.1

Table 2: The configurations of our models. Values in "training range", "inference range" and "grid size" columns are in meters. The first value in a column refers to x-axis, while the second value refers to y-axis. Please refer to Sec. 3.5 for more details.

4. Results

4.1. Performance on the Test Set

The final results on the official real-time 3D detection leaderboard are shown in Tab. 4. As seen from Tab. 4, our AFDetV2 submission achieved the 1st place of the real-time 3D detection challenge and reached 73.12 mAPH/L2 with 60.06 ms latency. Models that are eligible for the title of "Most Efficient Model" are listed in Tab. 5, where the primary evaluation metric is the models' latency. Our AFDetV2-Base submission was given the title of "Most Efficient Model" by the challenge sponsor and reached 72.57 mAPH/L2 with 55.86 ms latency. Our AFDetV2-Lite submission achieved 69.95 mAPH/L2, only 0.05 below 70.00, but with 46.90 ms latency, which is 8.94 ms faster than our "Most Efficient Model".

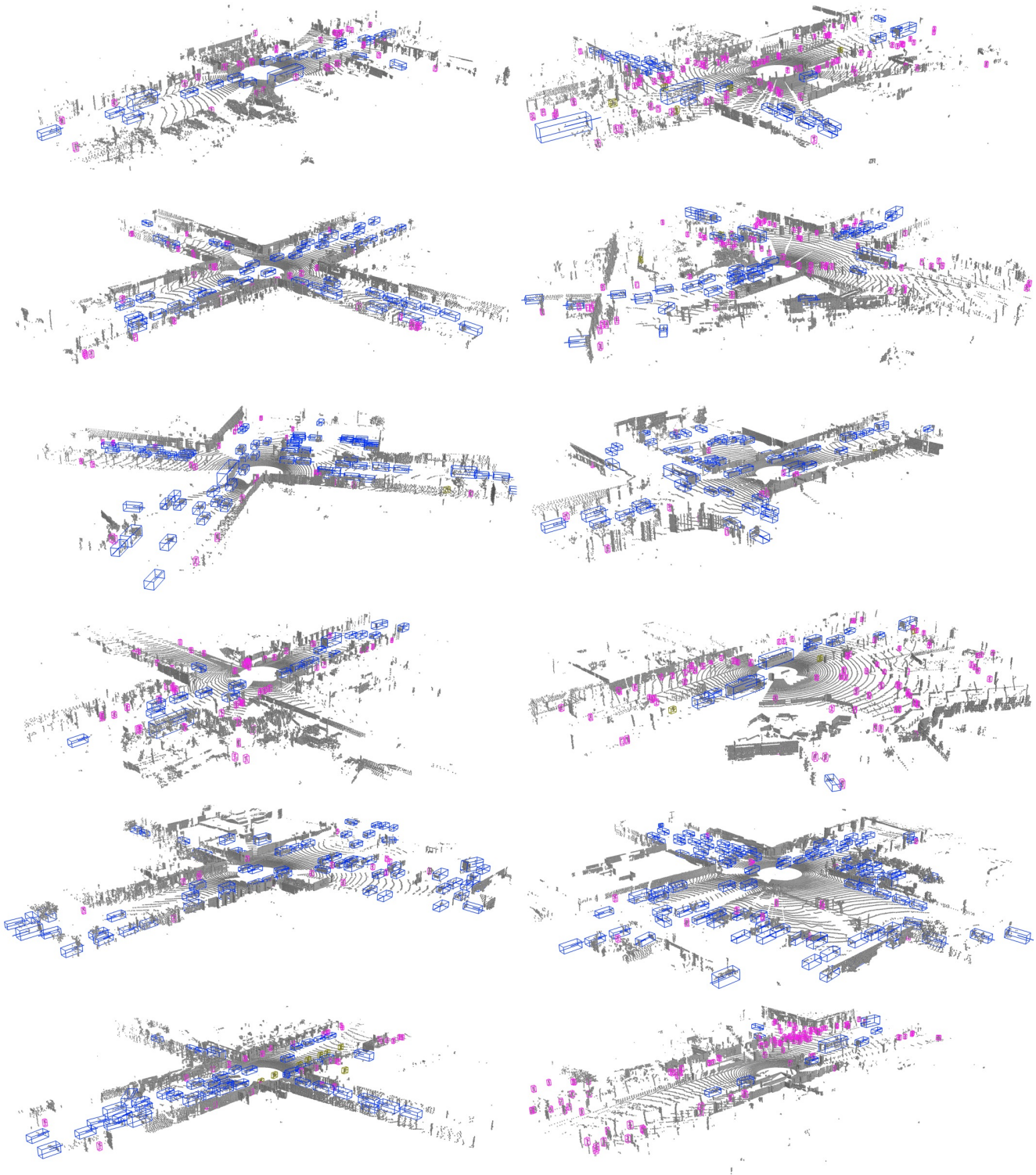


Figure 4: The examples of results on testing set of Real-time 3D detection track, only bounding boxes with score larger than 0.50 are visualized. Additional NMS is conducted for better visualization.

Model	Training Epoch	Keypoints Aux. Supervision	Refined Data Augmentation	SC-Conv RPN	ALL mAPH/L2
w/o IoU branch	3	✗	✗	✗	64.15
	3	✓	✗	✗	64.43
	3	✓	✓	✗	64.72
	3	✓	✓	✓	66.33
w/ IoU branch	3+2	✓	✗	✗	68.13
	3+2	✓	✓	✓	69.50

Table 3: The improvements of accuracy brought by different modules. All models were trained with densified point clouds. Results are shown in terms of mAPH/L2 for all classes, *i.e.* VEHICLE, PEDESTRIAN, CYCLIST. All models were tested on 1/5 validation set.

Models	VEHICLE		PEDESTRIAN		CYCLIST		ALL		Latency
	mAP/L2	mAPH/L2	mAP/L2	mAPH/L2	mAP/L2	mAPH/L2	mAP/L2	mAPH/L2	ms
AFDetV2 (Ours)	74.30	73.89	**75.47**	**72.41**	**74.05**	**73.04**	**74.60**	**73.12**	60.06
CenterPoint++	**75.47**	**75.05**	75.13	72.41	72.04	71.01	74.22	72.82	57.12
AFDetV2-Base (Ours)	73.89	73.46	75.34	72.29	72.96	71.97	74.06	72.57	55.86
X_Autonomous3D	74.04	73.60	72.29	68.27	70.55	69.50	72.29	70.46	68.42
AFDetV2-Lite (Ours)	72.98	72.55	73.71	68.61	69.84	68.67	72.18	69.95	**46.90**

Table 4: Top five submissions of the Real-time 3D Detection. Only models that run faster than 70 ms are qualified for awards. mAPH/L2 of all classes, *i.e.* VEHICLE, PEDESTRIAN, CYCLIST, is the primary metric for the award evaluation.

Models	ALL		Latency
	mAP/L2	mAPH/L2	ms
AFDetV2-Base (Ours)	74.06	72.57	**55.86**
CenterPoint++	74.22	72.82	57.12
AFDetV2 (Ours)	**74.60**	**73.12**	60.06
X_Autonomous3D	72.29	70.46	68.42

Table 5: Top four submissions of the Real-time 3D Detection "Most Efficient Model" title. Only models with mAPH/L2 greater than 70 are qualified for this title. Latency is the primary metric for the "Most Efficient Model" title evaluation.

4.2. Ablation Studies

To study the effect of each module deployed in our solution, we conducted ablation experiments on the 1/5 validation set as shown in Tab. 3. We divided the ablation study into two sections: one for the case without the IoU branch, and the other with the IoU branch. From the top part of Tab. 3, we can see that without the IoU branch, the SC-Conv [13] led to an increment of detection performance of 1.61 mAPH/L2. The keypoints auxiliary loss and the refined data augmentation each contributed about 0.3 mAPH/L2. From the bottom part of Tab. 3, we observe that the IoU branch can significantly increase the accuracy. With the IoU branch (and two additional training epochs), the accuracy increased by 3.17 mAPH/L2 compared to the case of without the IoU branch, when all the other modules were deployed.

5. Conclusion

We have shown a real-time, single-stage and anchor-free 3D object detection model named AFDetV2. This model is composed of a lite 3D Feature Extractor, a refined SC-Conv RPN and an anchor-free head with a novel IoU-aware branch and a keypoints auxiliary branch. To reduce the inference latency, we also devised a GPU-based voxelization method. Thanks to these innovations, AFDetV2 achieved the 1st Place award for Real-time 3D Detection and the title of "Most Efficient Model" of the Waymo Open Dataset Challenges 2021.

References

[1] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 4

[2] Zhuangzhuang Ding, Yihan Hu, Runzhou Ge, Li Huang, Sijia Chen, Yu Wang, and Jie Liao. 1st place solution

for waymo open dataset challenge–3d detection and domain adaptation. *arXiv preprint arXiv:2006.15505*, 2020. 3, 4

[3] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles Qi, Yin Zhou, et al. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. *arXiv preprint arXiv:2104.10133*, 2021. 1

[4] Runzhou Ge, Zhuangzhuang Ding, Yihan Hu, Yu Wang, Sijia Chen, Li Huang, and Yuan Li. Afdet: Anchor free one stage 3d object detection. In *CVPR Workshops*, 2020. 1, 2, 3

[5] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017. 1, 2

[6] Sylvain Gugger. The 1cycle policy. https://sgugger.github.io/the-1cycle-policy.html, 2018. 4, 5

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 3

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 4, 5

[9] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *UAI*, 2018. 4

[10] Borui Jiang, Ruixuan Luo, Jiayuan Mao, Tete Xiao, and Yuning Jiang. Acquisition of localization confidence for accurate object detection. In *ECCV*, 2018. 3

[11] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *ECCV*, 2018. 2, 3

[12] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *CVPR*, 2017. 3

[13] Jiang-Jiang Liu, Qibin Hou, Ming-Ming Cheng, Changhu Wang, and Jiashi Feng. Improving convolutional networks with self-calibrated convolutions. In *CVPR*, 2020. 2, 7

[14] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 5

[15] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *CVPR*, 2020. 3

[16] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR*, 2019. 3

[17] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *IEEE Transactions on PAMI*, 2020. 3

[18] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, 2020. 1, 4

[19] Guojun Wang, Bin Tian, Yunfeng Ai, Tong Xu, Long Chen, and Dongpu Cao. Centernet3d: An anchor free object detector for autonomous driving. *arXiv preprint arXiv:2007.07214*, 2020. 3

[20] Yu Wang, Sijia Chen, Li Huang, Runzhou Ge, Yihan Hu, Zhuangzhuang Ding, and Jie Liao. 1st place solutions for waymo open dataset challenges–2d and 3d tracking. *arXiv preprint arXiv:2006.15506*, 2020. 3

[21] Shengkai Wu, Xiaoping Li, and Xinggang Wang. Iou-aware single-stage object detector for accurate localization. *Image and Vision Computing*, 97:103911, 2020. 3

[22] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 1, 2, 4

[23] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. In *CVPR*, 2021. 3

[24] Haoyang Zhang, Ying Wang, Feras Dayoub, and Niko Sünderhauf. Varifocalnet: An iou-aware dense object detector. In *CVPR*, 2021. 4

[25] Wu Zheng, Weiliang Tang, Sijin Chen, Li Jiang, and Chi-Wing Fu. Cia-ssd: Confident iou-aware single-stage object detector from point cloud. In *AAAI*, 2021. 3

[26] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018. 2

[27] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019. 2, 4