SwiftPillars: High-efficiency Pillar Encoder for Lidar-based 3D Detection

Xin Jin[1,2*], Kai Liu[2*], Cong Ma[2*], Ruining Yang[1,2], Fei Hui[1†], Wei Wu[2,3†]

[1]Chang'an University, [2]SenseAuto Research, [3]Tsinghua University
{jinxin, yangruining, feihui}@chd.edu.cn, {liukai3.iag, macong, wuwei}@senseauto.com

Abstract

Lidar-based 3D Detection is one of the significant components of Autonomous Driving. However, current methods over-focus on improving the performance of 3D Lidar perception, which causes the architecture of networks becoming complicated and hard to deploy. Thus, the methods are difficult to apply in Autonomous Driving for real-time processing. In this paper, we propose a high-efficiency network, SwiftPillars, which includes Swift Pillar Encoder (SPE) and Multi-scale Aggregation Decoder (MAD). The SPE is constructed by a concise Dual-attention Module with lightweight operators. The Dual-attention Module utilizes feature pooling, matrix multiplication, etc. to speed up point-wise and channel-wise attention extraction and fusion. The MAD interconnects multiple scale features extracted by SPE with minimal computational cost to leverage performance. In our experiments, our proposal accomplishes 61.3% NDS and 53.2% mAP in nuScenes dataset. In addition, we evaluate inference time on several platforms (P4, T4, A2, MLU370, RTX3080), where SwiftPillars achieves up to 13.3ms (75FPS) on NVIDIA Tesla T4. Compared with PointPillars, SwiftPillars is on average 24.34% faster in inference speed with equivalent GPUs and a higher mAP of approximately 3.2% in the nuScenes dataset.

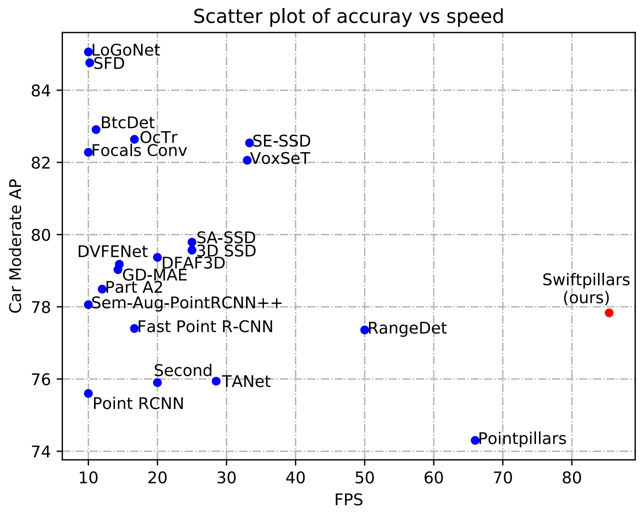

Figure 1: Performance of SwiftPillars on the KITTI test set with current mainstream methods in terms of inference speed and detection accuracy.

Introduction

Recently, autonomous driving technology has received considerable attention, which brings a huge impact on transportation. One of the most essential technologies for autonomous driving is computer vision, where efficient Lidar-based 3D object detection is the main area of research (Feng et al. 2023). This research plays a vital role in efficiently perceiving the surroundings of scenario and facilitating subsequent tasks such as driving decision and planning. But the development of Lidar-based 3D detectors for practical use in autonomous driving continues to face significant challenges, especially in real-time perception.

But in practical applications, real-time capability and deployability are crucial. Real-time perception allows timely responses to dynamic traffic changes, ensuring safe driving by detecting moving obstacles and signal light updates. Moreover, real-time perception improves driving efficiency by continuously capturing information about nearby vehicles' positions, enabling precise path planning and decision-making, and reducing unnecessary stops and waiting times.

However, most existing methods (Zhou et al. 2022; Zhao et al. 2021; Mao et al. 2021a; Li et al. 2021; Sun et al. 2022; Dong et al. 2022; Shi et al. 2020) prioritize improving detection accuracy, leading to network architectures complexity and overlooking practical deployment and inference speed. (Yang et al. 2020; Qi et al. 2019; Chen et al. 2022a; Shi, Wang, and Li 2019) primarily adopts PointNet (Qi et al. 2017) as the main feature extraction unit; (Yan, Mao, and Li 2018; Chen et al. 2023b,a; Deng et al. 2021; Hu, Kuai, and Waslander 2022; Xu, Zhong, and Neumann 2022; Yin, Zhou, and Krahenbuhl 2021) apply 3D convolutions after voxelizing the point cloud. Furthermore, some methods have utilized transformer architectures (Zhou et al. 2022; Sun et al. 2022; Feng et al. 2023) to provide a larger receptive field and extract global feature information. While each of these methods has contributed to the improvement of detection accuracy, their network structure become complex, which leads to deployment limited in practical application.

[*]These authors contributed equally.
[†]Corresponding Author.

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

For (Wu et al. 2021; Shi, Li, and Ma 2022; Li, Luo, and Yang 2023; Hu et al. 2022; Yang et al. 2020), although they achieve much faster inference than the previously proposed methods, these methods rely on sparse convolution, which is still hard to deploy in practice.

In order to solve the above issues, in this paper, we propose an efficient pillar-based detector, SwiftPillars. Firstly, we analyze the efficiency bottleneck of the most widely used detector, PointPillars, and find that its Point Feature Network (Qi et al. 2017) for encoding pillar features performs multiple matrix transpositions, which incurs significant I/O time cost, and its capability of feature extraction is also limited. As a result, we propose a novel architecture to improve both of efficiency and performance of Lidar Perception, which includes Swift Pillar Encoder (SPE) and Multi-scale Aggregation Decoder (MAD). The SPE fuses point-wise and channel-wise attention to encode the pillar information of the point cloud, achieving efficient pillar feature extraction. The MAD obtains pyramidal features by feature self-sampling, and multi-scale feature aggregation is achieved by interconnecting features at different scales. Then, for better real-time performance, we optimize the operators and interconnection of modules chiefly, analyze the impact of different operators on the inference speed, and propose an effective optimization scheme. Finally, we conduct sufficient experiments on three publicly available datasets (KITTI, nuScenes, and DAIR-V2X-I) and many different computing platforms. We compare SwiftPillars and PointPillars across multiple computing platforms (T4, P4, A2, MLU370, RTX3080). The results show that SwiftPillars inference speed is on average 24.34% faster than PointPillars on equivalent GPUs. On the nuScenes dataset, SwiftPillars achieves the state-of-the-art inference speed, up to 13.3ms (75FPS) on an NVIDIA Tesla T4, and achieves 53.2% mAP, 3.2% higher than PointPillars(Lang et al. 2019). The results show that our SwiftPillars achieve the fastest inference speeds currently while guaranteeing satisfactory detection accuracy. The main contributions of this paper are as follows:

- We propose a real-time LiDAR perception architecture, SwiftPillars, which achieves high-speed and accurate 3D object detection and enables easy deployment;
- We design two novel models, Swift Pillar Encoder (SPE) and Multi-scale Aggregation Decoder (MAD). The SPE achieves efficient pillar feature extraction by Dual-attention Module. The MAD enables multi-scale feature aggregation;
- We analyze the operators currently widely used in 3D detectors, identify their performance bottlenecks, and propose an effective optimization scheme that greatly improves the inference speed, which can provide an effective reference for future network development;
- We conduct evaluations across multiple platforms and achieve the fastest inference speed among all mainstream methods. Our SwiftPilars is on average 24.34% faster than PointPillars and even 100.2% faster on NVIDIA Tesla A2. Additionally, our method outperforms PointPillars in mAP by 3.2%.

Related Work

High-precision 3D detectors. Currently, Existing high-precision methods typically use Voxel-based approaches and Transformer-based approaches. Voxel-based methods (Hu et al. 2022; Yu et al. 2022a; Kuang et al. 2020; Mahmoud, Hu, and Waslander 2023; Wang et al. 2022; Chen et al. 2022b) grid the point cloud and apply 3D CNN. For example, Second (Yan, Mao, and Li 2018) considers the sparsity of point clouds and introduces 3D sparse convolution to improve efficiency. PV-RCNN series work (Shi et al. 2020, 2023) compensates for the loss of precise positional information during voxelization by introducing additional raw point cloud information. Largekernel3D (Chen et al. 2023a) proposes a large-kernel 3D CNN network to increase the network's receptive field. Transformer-based methods (Wang et al. 2023; Sun et al. 2022; Yang et al. 2023; Fan et al. 2022a) improve model performance by incorporating transformer architecture into different parts of the network. Point transformer (Zhao et al. 2021) first applies transformer to point cloud processing. Voxel transformer (Mao et al. 2021b) combines transformer with voxels. Centerformer (Zhou et al. 2022) uses multi-frame point clouds and center point features as inputs to the transformer. Although the methods achieve outstanding detection accuracy, techniques like expanding convolutional kernels or introducing transformer architecture unavoidably increase the complexity of the network, resulting in exceptionally slow model detection speed. In contrast to the methods, our method stands out by offering both high speed and high precision that are practically applicable.

High-speed 3D Detectors. To meet the demands of practical applications, some studies (Zheng et al. 2021a; He et al. 2020; Yang et al. 2020; Fan et al. 2022b; Wu et al. 2021) have made contributions to improve network speed. One of the most well-known examples is PointPillars (Lang et al. 2019), which compresses the height dimension information during voxelization and directly employs 2D convolutions for feature extraction. This design makes it easy to deploy and significantly faster, making it widely used in the industry. However, this compression of information leads to significant information loss. To achieve higher detection speeds, 3DSSD (Yang et al. 2020) removes the upsampling layer, which is initially essential in point-based methods, and proposes a new point cloud downsampling method to preserve more foreground points to maintain network performance. CIA-SSD (Zheng et al. 2021a) introduces a lightweight spatial semantic feature aggregation module that adaptively fuses high and low-level spatial features. PillarNet (Shi, Li, and Ma 2022) aims to retain the extracted information from pillars and inserts 2D sparse convolutions for further feature extraction, along with a novel neck module to aggregate multi-scale features. While these methods strike a balance between accuracy and speed, they still fall short of real-time perception and rely heavily on operations such as sparse convolutions, which are challenging to deploy in practice. Therefore, we propose SwiftPillars, which demonstrates significantly improved inference speed over existing methods on multiple computing platforms.

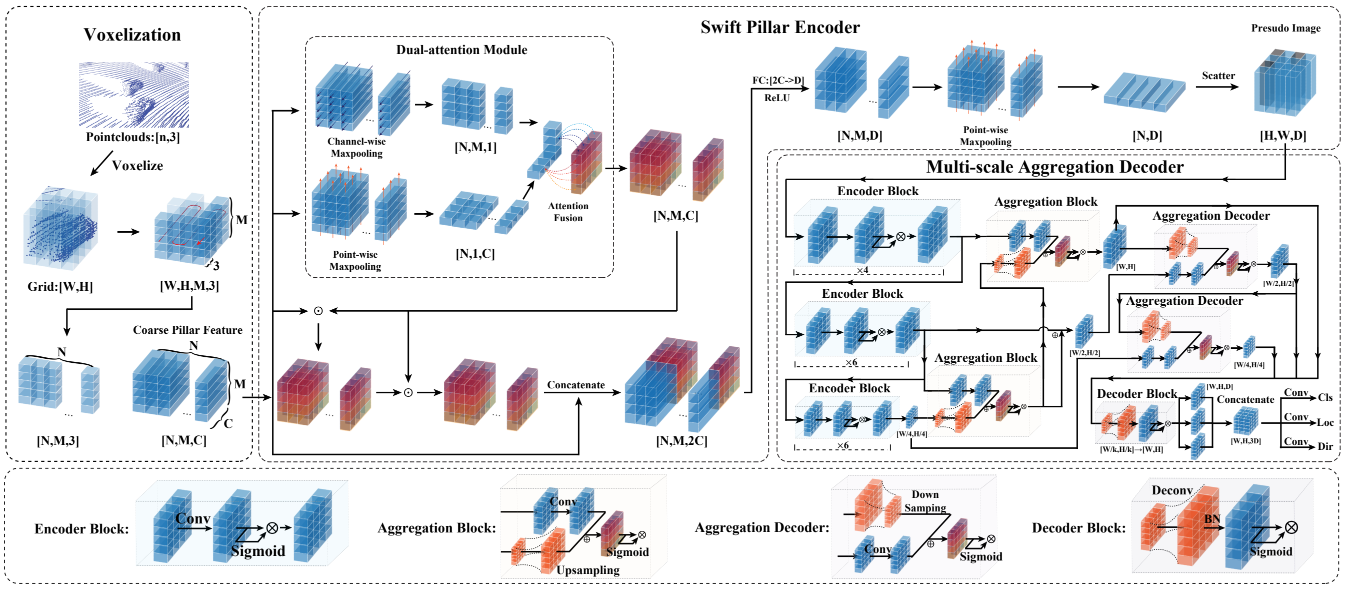

Figure 2: Illustration of the proposed SwiftPillars. It is primarily composed of three components: 1) Voxelization: Voxelizing the input point cloud into pillars, consistent with the approach of PointPillars (Lang et al. 2019); 2) Swift Pillar Encdoer: Performing feature extraction on pillars to generate pseudo-images; and, 3) Multi-scale Aggregation Decoder: Aggregating multi-scale pseudo-image features and generating 3D detection results.

Method

In this section, we first present the overview of the SwiftPillars architecture. Then, we describe the detailed explanation of two critical modules, the Swift Pillar Encoder (SPE) and the Multi-scale Aggregation Decoder (MAD).

Overview

The overall architecture of SwiftPillars, as shown in Fig. 2. Firstly, the point cloud is voxelized using the pillar-based approach to obtain coarse pillar features. Then, the coarse pillar features are fed into the Swift Pillar Encoder, which utilizes the Dual-attention module to extract features and generate pseudo-images. Subsequently, the Multi-scale Aggregation Decoder aggregates features from pseudo-images across various scales. Lastly, these features are used by specific detection heads to produce the final results.

Swift Pillar Encoder

Given input point clouds $P \in \mathbb{R}^{N\times 3}$, where N represents the number of points and each point denotes 3-dimensional space coordinates $[x, y, z]$. The 3D scene of P is partitioned as several voxels according to the given grid size, and the shape of the grid along the X and Y directions indicates W and H respectively. The initial number of pillars is divided into $W \times H$. However, due to the sparsity of the point cloud, many pillars do not contain any points. We denote the number of non-empty pillars as N. Within each pillar, the maximum number of points is set as M. Each point is enlarged from 3 to C, which is defined as $V : [x, y, z] \rightarrow [x, y, z, m_x, m_y, c_x, c_y, c_z]$, where c_x, c_y, c_z represent the arithmetic average distance of all points inside the pillar and m_x, m_y indicate the horizontal distance from the center of the pillar. Each voxel indicates $V : \{v_1, v_2, ..., v_N\}$, v_N refers to the N-th non-empty pillar in the 3D scenes. The pillars are composed as Coarse Pillar Feature $\in \mathbb{R}^{N\times M\times C}$ as the SwiftPillars Network input.

Dual-attention Module We utilize both channel-wise attention and point-wise attention to extract more advanced features within each pillar.

Channel-wise Attention. For each pillar in V, we perform max pooling along the point dimension, resulting in channel-wise features with an output of $C^{in} \in \mathbb{R}^{N\times 1\times C}$. Subsequently, the feature is fed into two fully connected layers, with an activation function added after the first fully connected layer. The output of the channel-wise attention, denoted as $C^{out} \in \mathbb{R}^{N\times 1\times C}$, is calculated as follows:

$$C^{out} = fc_{c2}(\sigma(fc_{c1}(C^{in}))) \tag{1}$$

where fc_{c1} represents first fully connected layer $fc\ (C \rightarrow C)$, fc_{c2} represents the second fully connected layer $fc\ (C \rightarrow C)$ and σ represents the ReLU activate function.

Point-wise Attention. Similar to channel-wise attention, for each pillar in V, we perform max pooling along the channel dimension, resulting in point-wise features with an output of $P^{in} \in \mathbb{R}^{N\times M\times 1}$. The output of the channel-wise attention, denoted as $P^{out} \in \mathbb{R}^{N\times M\times 1}$, is calculated as follows:

$$P^{out} = fc_{p2}(\sigma(fc_{p1}(P^{out}))) \tag{2}$$

where fc_{p1} represents the first fully connected layer $fc\ (M \rightarrow M)$, and fc_{p2} represents second fully connected layer $fc\ (M \rightarrow M)$ and σ represents the ReLU activate function.

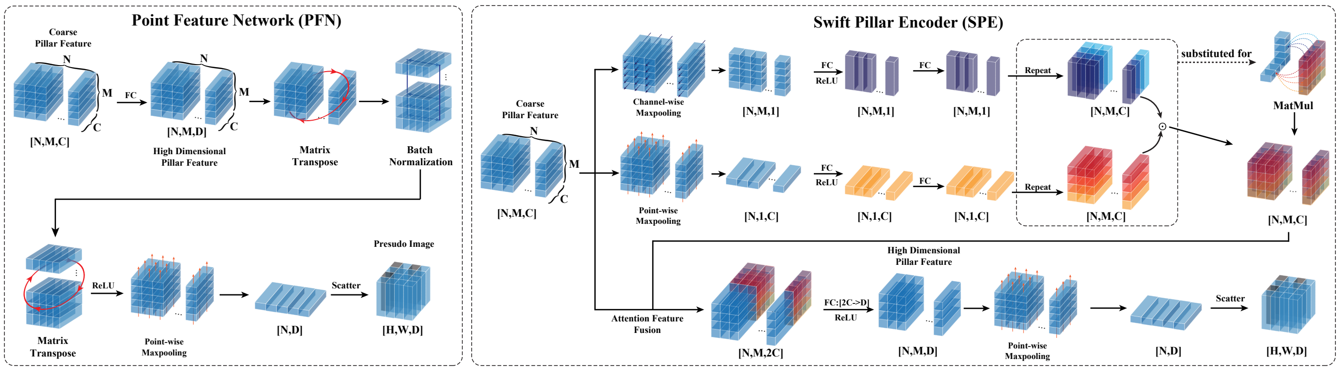

Figure 3: The structure of the Swift Pillar Encoder (SPE).

And then, both attention features C^{out}, P^{out} are fused by Attention Fusion, which is described as follows:

$$PC^{out} = (P^{out} \otimes \varphi_{1\times C}) \odot (\varphi_{M\times 1} \otimes C^{out}) \qquad (3)$$

where φ represents the All-Ones Matrix to align the dimension of different attention features C^{out}, P^{out}. And $\odot$ indicates Hadamard Product, which aims to fuse the outputs of the repeated features. However, considering that the $matmul$ operator can achieve speedups on parallel platforms, Eq.3 is replaced by $matmul$ operator, which is shown in Fig. 3. The formula is substituted for:

$$PC^{out} = matmul\{P^{out}, C^{out}\} \qquad (4)$$

Pseudo-image Generation Define the coarse pillar features obtained after voxelization as $P_{coarse} \in \mathbb{R}^{N\times M\times C}$, which we feature-weight with the output of Dual Attention Module PC^{out} to better capture those features with contributing information. Then, the $F \in \mathbb{R}^{N\times M\times C}$ is obtained by fusing it with the PC^{out}. Subsequently, concatenating F with the coarse pillar features generates a pillar feature of size $(N, M, 2C)$, and feeding it into a regular fully-connected layer $FC(2C \to D)$ enhances the feature dimensionality to output a feature of size (N, M, D). Finally, maximum pooling is performed in point-wise dimension to obtain a feature of (N, D), which is scattered to generate a pseudo-image of size (H, W, D), where H and W represent the height and width of the feature body.

SPE vs PFN As shown in Fig. 3, existing pillar-based approaches usually adopt the Point Feature Network (PFN) proposed by PointPillars (Lang et al. 2019) as a component of their pillar encoding network. (Liu et al. 2020; Le et al. 2022), feed the output of attention into the PFN to make up their complete pillar encoder, thus are actually two separate modules. We design a novel architecture that fuses PFN and attention into a single module and made a lot of effort on the operators to improve the inference speed. For example, in the feature fusion part, We found that in order to perform a Batch Normalization along point feature dimension, PFN has to use Matrix Transpose twice to accomplish the above operation (corresponding to permute of pytorch), which in turn increased the overall inference time consuming of PFN. And the reasoning elapsed time increases with the increase of input dimensions. So, we removed the operator. In the end, SPE achieves a significant improvement in inference speed while maintaining the original encoding capability.

Multi-scale Aggregation Decoder

The main idea of Multi-scale Aggregation Decoder is that after extracting pyramid features, the features of each scale are connected with other scales like BiFPN (Tan, Pang, and Le 2020), and the features containing multi-scale information are aggregated to get the final output. Specifically, we define the pseudo-image feature output from SPE as p_0 and input it into three cascaded Encoder Blocks. p_1, p_2 and p_3 are three intermediate outputs, representing the features of three different scales. The calculation formula is:

$$p_i = conv(conv(p_{i-1})) \otimes \sigma(conv(conv(p_{i-1}))) \qquad (5)$$

where σ represents $sigmoid$ activate function.

Then, the features of each scale are connected with each other. The Aggregation Block is used to connect low-scale features to high-scale features, and the Aggregation Decoder is used to connect high-scale features to low-scale features. The corresponding output at each scale can be formulated as follows:

$$\begin{aligned} p_1^{out} &= AB_1(p_1, AB_2(p_2, p_3)), \\ p_2^{out} &= AD_1(p_1^{out}, p_2 + AB_2(p_2, p_3)), \\ p_3^{out} &= AD_2(p_2^{out}, p_3), \end{aligned} \qquad (6)$$

where AB_i, AD_i denote the i-th Aggregation Block, Aggregation Decoder, p_i^{out} represents the output corresponding to the fused multi-scale features at scale p_i. The Decoder Block is used to align feature size at each scale. And the final feature output p^{out} is the concatenation of features from all scales. It can be expressed by the following formulas:

$$P^{out} = concate[DB(p_1), DB(p_2), DB(p_3)] \qquad (7)$$

where DB denotes the Decoder Block. Finally, P_{out} is fed into the detection head of SwiftPillars to output the final 3D Bounding Boxes.

Head and Loss Function

Following previous methods, We use two detection heads for specific datasets, anchor-based and anchor-free.

Method		mAP	NDS	Car	Truck	Bus	Trailer	CV	Ped	Motor	BC	TC	Barrier
3DSSD (2020)	P	42. 6	56. 4	81. 2	47. 2	61. 4	30. 5	12. 6	70. 2	36. 0	8.6	31. 1	47. 9
SASA (2022a)		45. 0	61. 0	76. 8	45. 0	66. 2	36. 5	16. 1	69. 1	39. 6	16. 9	29. 9	53, 6
CenterPoint (2021)	V	58. 0	65. 5	84. 6	51. 0	60. 2	53. 2	17. 5	83. 4	53. 7	28. 7	76. 7	70. 9
Focals Conv (2023c)		63. 8	70. 0	86. 7	56. 3	67. 7	59. 5	23. 8	87. 5	64. 5	36. 3	81. 4	74. 1
LargeKernel3D (2023a)		65. 4	70. 6	85. 5	53. 8	64. 4	59. 5	29. 7	85. 9	72. 7	46. 8	79. 9	75. 5
TransFusion-L (2022)	T	65. 5	70. 2	86. 2	56. 7	66. 3	58. 8	28. 2	86. 1	68. 3	44. 2	82. 0	**78. 2**
SphereFormer (2023)		65. 5	70. 7	84. 9	55. 1	66. 4	59. 3	29. 9	86. 0	71. 4	47. 1	79. 7	75. 2
FocalFormer (2023c)		**68. 7**	**72. 6**	**87. 2**	**57. 1**	**69. 6**	**64. 9**	**34. 4**	**88. 2**	**76. 2**	**49. 6**	**82. 3**	77. 8
PointPillars (2019)	p	30. 5	45. 3	68. 4	23. 0	28. 2	23. 4	4.1	59. 7	27. 4	1.1	30. 8	38. 9
SA-Det3D (2021)		47. 0	59. 2	81. 2	43. 8	57. 2	47. 8	11. 3	74. 5	38. 0	13. 9	63. 9	54. 2
PillarAcc (2023)		47. 8	59. 0	-	-	-	-	-	-	-	-	-	-
SwiftPillars (ours)		**53. 2**	**61. 3**	**83. 2**	**45. 1**	**52. 9**	**52. 1**	**16. 0**	**79. 0**	**47. 7**	**20. 6**	**68. 8**	**65. 7**

Table 1: Comparison of LiDAR-only based methods on the nuScenes test set. Results are reported by the official nuScenes evaluation server. The meaning of abbreviation: "P", "V", "T" and "p" mean "Point-based", "Voxel-based", "Transformer-based" and "Pillar-based"; "CV", "Ped", "BC" and "TC" mean "Construction_Vehicle", "Pedestrian", "Bicycle" and "Traffic_Cone". We submit our test results without using Test-Time Augmentation and Double Flip.

The anchor-based head are the same as PointPillars (Lang et al. 2019), using Smooth-L1 to compute the regression loss $\mathcal{L}_{reg}$, focal loss to compute the classification loss $\mathcal{L}_{cls}$, and cross-entropy loss computes the direction loss $\mathcal{L}_{dir}$, and the total loss $\mathcal{L}_{anchor-based}$ is defined as:

$$\mathcal{L}_{anchor-based} = \lambda_{reg}\mathcal{L}_{reg} + \lambda_{cls}\mathcal{L}_{cls} + \lambda_{dir}\mathcal{L}_{dir} \quad (8)$$

where $\lambda_{reg}, \lambda_{cls}$ and λ_{dir} is used to control the weights of different losses.

The anchor-free head adds an iou branch to (Yin, Zhou, and Krahenbuhl 2021), following it, the cross-entropy loss and Smooth-L1 loss are used for heat-map classification loss $\mathcal{L}^{*}_{cls}$ and 3D box regression loss $\mathcal{L}^{*}_{reg}$. In addition, we use iou loss to compute $\mathcal{L}iou$, and the total loss $\mathcal{L}_{anchor-free}$ is defined as follows:

$$\mathcal{L}_{anchor-free} = \lambda^{*}_{reg}\mathcal{L}^{*}_{reg} + \lambda^{*}_{cls}\mathcal{L}^{*}_{cls} + \lambda_{iou}\mathcal{L}_{iou} \quad (9)$$

where we set $\lambda^{*}_{reg}, \lambda^{*}_{cls}$, and λ_{iou} denote the weight factors of the corresponding loss, respectively.

Experiments

Datasets and Metrics

In this section, we perform extensive experiments of the proposed SwiftPillars on three major benchmarks of large-scale nuScenes dataset (Caesar et al. 2020), KITTI (Geiger, Lenz, and Urtasun 2012) and DAIR-V2X (Yu et al. 2022b).

nuScenes Dataset. nuScenes is a challenging large-scale dataset that provides 1000 scenes, of which 700 are used for training, 150 for validation, and 150 for testing. The official evaluation server uses NDS to measure the performance of the model, which jointly considers the mean detection accuracy (mAP) and several different aspects of the error (e.g., Orientation, Velocity).

KITTI Dataset. KITTI is a prevalent dataset for 3D object detection tasks, including 7481 training samples and 7518 test samples. It contains three different categories of annotated information, which are categorized into three difficulty levels: easy, medium, and hard according to the degree of occlusion. We follow the latest official test metric and use 40 recall positions to compute AP, to measure the performance of the model on the val set.

DAIR-V2X Dataset. DAIR-V2X is a novel publicly available dataset targeting detection tasks in autonomous driving. We chose its roadside data DAIR-V2X-I to conduct our experiment. It contains 10084 frames of point cloud labeled with 10 classes, and 7058 samples have been released for training and validation. The evaluation metric is consistent with KITTI.

Implementation details

We conduct our experiments in the OpenPCDet framework (Team 2020). The models are all trained using the Adam optimizer with one-cycle learning rate strategy with an initial learning rate of 0.001 on 8 NVIDIA V100 GPUs. In speed evaluation, all models are converted to ONNX on different GPUs (P4, T4, A2, MLU370, RTX3080) with TensorRT accelerating. And their hyperparameters are set to be consistent with those of (Shi, Li, and Ma 2022), with a weight decay of 0.01 and a momentum of 0.9. Additionally, we employ some normal data augmentation, including gt-sampling, random flip, random rotation, random scaling.

nuScenes Dataset. We voxelize the 3D space with a detection range of $[-51.2, -51.2, -5.0, 51.2, 51.2, 3.0]$ using a voxel size of $[0.2, 0.2, 8]$, where the maximum number of points within each voxel is set to 32. The batch size is set to 32 to train 20 epochs, which takes about 15 hours. The detection head follows (Yin, Zhou, and Krahenbuhl 2021), and an additional iou branch is added, the weight of IOU loss is set to 1.0, and the rest of the parameter configurations are consistent with (Yin, Zhou, and Krahenbuhl 2021).

KITTI and DAIR-V2X-I Dataset. Following the official repository, we convert the DAIR-V2X-I to KITTI format. The same training configuration is used for both of them. we voxelize the 3D space with a detection range of $[0, -39.68, -3, 69.12, 39.68, 1]$ using a voxel size of $[0.16, 0.16, 4]$, where the maximum number of points within each voxel is set to 32. Batch size is set to 32 to train 80 epochs. Following (Lang et al. 2019), we use the anchor-based detection head and the same parameters.

Dataset	Method	Car			Ped.			Cyc.			mAP
		Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	
KITTI	PointPillars	87. 04	78. 02	75. 38	52. 94	48. 51	44. 39	82. 43	**63. 19**	**59. 09**	63. 43
	SwiftPillars (ours)	**87. 55**	**78. 58**	**75. 72**	**56. 32**	**50. 57**	**45. 61**	**82. 83**	62. 37	57. 86	**63. 84**
DAIR-V2X-I	PointPillars	69. 39	56. 94	56. 97	69. 35	66. 67	66. 85	59. 38	30. 33	30. 75	51. 31
	SwiftPillars (ours)	**69. 52**	**57. 06**	**57. 08**	**71. 15**	**68. 41**	**68. 55**	**60. 63**	**30. 60**	**31. 52**	**52. 02**

Table 2: Comparison with PointPillars on KITTI and DAIR-V2X-I validation sets. The experimental results are performed using the official evaluator for the calculations. Abbreviation Explanations: ”Ped.” and ”Cyc.” stand for ”Pedestrian” and ”Cyclist”; ”Mod.” signifies ”Moderate”.

Comparison Evaluation

We evaluate the detection performance of SwiftPillars, which is the basis on which our SwiftPillars can be practically applied to autonomous driving perception tasks. Specifically, We compare our method with the current mainstream methods on nuScenes to analyze its detection accuracy. And we directly compare it with PointPillars on DAIR-V2X-I and KITTI to show the performance of SwiftPillars in practical application.

Results of nuScenes. The evaluation results on the nuScenes test set are shown in Table 1. We classify the existing methods into four categories according to the different ways of data encoding. It can be seen that Voxel-based methods and Transformer-based methods have been leading the road in terms of accuracy in recent years, with NDS reaching over 70% and mAP over 60%. Although SwiftPillars still has a gap in detection accuracy compared to these two types of methods, it is more accurate than existing point-based methods, with 8.2% higher mAP and 0.3% higher NDS compared to SASA. And we also have some advantages in the same pillar-based methods. The main reason for our low accuracy is that pillar encoding of point clouds inevitably loses more information compared to voxel encoding and transformer encoding, and the transformer architecture is able to obtain a larger receptive field, thus improving the ability to extract important information. However, SwiftPillars has a much faster detection speed, which is a huge improvement in the speed of acquisition at a small accuracy loss, making it better suited to meet the real-time requirements of autonomous driving perception tasks.

Results of KITTI and DAIR-V2X-I. The evaluation results on the KITTI and DAIR-V2X-I validation sets are shown in Table 2. Under medium difficulty, compared to PointPillars, the mAP of SwiftPillars is 0.41% and 0.71% higher on the KITTI dataset and DAIR-V2X-I dataset, respectively. Among them, in the category of pedestrians, SwiftPillars is 2.06% and 1.74% higher under medium difficulty, respectively.

Although our SwiftPillars is slightly lower in accuracy compared to the methods proposed in recent years, SwiftPillars can be directly applied to real-world application scenarios. Moreover, PointPillars is still widely used in industry, while our method shows better performance than it, with higher detection accuracy on all three datasets, and at the same time, we achieve faster inference speed (Analyzed in the following subsections). Thus, our method achieves a better balance of performance between speed and accuracy.

Method	NDS (%)	mAP (%)	FPS
PFN + RPN (CP)	57. 93	48. 72	70
SPE + RPN	59. 05	49. 75	81
PFN + MAD	59.43	51.16	65
SPE + MAD (SwiftPillars)	**60. 82**	**52. 00**	75

Table 3: Independent analysis of the components of SwiftPillars on nuScenes val set. We did not use Test-Time Augmentation or Double Flip. ”CP” represents our baseline Centerpoint-pillar (Yin, Zhou, and Krahenbuhl 2021), ”SPE” represents ”Swift Pillar Encoder”, and ”MAD” represents ”Multi-scale Aggregation Decoder”.

Method	FPS	mAP_{3D}@car on val			
		Easy	Mod.	Hard	mAP
Point-RCNN (2019)	10	86.65	75.68	68.92	77.08
FocalConv (2022b)	10	89.52	84.93	79.18	84.54
OcTr (2023)	17	89.80	**86.97**	79.28	85.35
3D SSD (2020)	25	89.71	79.45	78.67	82.61
TANet (2020)	27	88.21	77.85	75.62	80.56
SE-SSD (2021b)	33	**93.19**	86.12	**83.31**	**87.54**
PointPillars (2019)	66	86.62	76.06	68.91	77.20
SwiftPillars (Ours)	**85**	87.55	78.58	75.72	80.61

Table 4: Comparison of the speed of inference and the detection accuracy with previous methods. The inference time is obtained from officially published sources. ”Mod.” means ”Moderate”.

Ablation Study

To validate the effectiveness of our proposed Swift Pillar Encoder and Multi-scale Aggregation Decoder, we evaluate their impact on network performance and efficiency based on nuScenes validation set.

Effects of Swift Pillar Encoder As shown in Table 3, after replacing Point Feature Network of Centerpoint-pillar with Swift Pillar Encoder, the mAP is improved by 1.03%, the NDS is improved by 1.12%, and the detection speed is increased from 70 FPS to 81 FPS. The results show that compared with the Point Feature Network, Swift Pillar Encoder can extract prominent pillar features more efficiently.

Effects of Multi-scale Aggregation Decoder As shown in Table 3, after replacing the original RPN with MAD, the mAP improves by 2.44%, and the NDS improves by 1.5%, without increasing inference time. This indicates that the Multi-scale Aggregation Decoder is a powerful feature de-

Dataset	Platform	PointPillars	SwiftPillars	Improvment
DAIR-V2X-I	MLU370	12.24ms	10.02ms	20.0%
	RTX3080	3.45ms	3.39ms	1.8%
	P4	11.7ms	10.0ms	17%
	T4	6.70ms	5.18ms	29.3%
	A2	6.70ms	4.49ms	33.2%
KITTI	MLU370	29.24ms	29.18ms	0.2%
	RTX3080	6.73ms	6.56ms	2.5%
	P4	27.75ms	24.06ms	15.3%
	T4	15.01ms	11.71ms	28.2%
	A2	21.53ms	10.75ms	**100.2%**

Table 5: Comparison of inference time between PointPillars and SwiftPillars on different GPU platforms. ”P4”, ”A2”, ”T4” and ”RTX3080” mean ”NVIDIA Tesla P4”, ”NVIDIA A2”, ”NVIDIA T4” and ”NVIDIA GeForce RTX3080”. All results are based on the same input data.

coder, and it can integrate more information from different scales to improve detection performance.

Runtime Overhead Analysis

To evaluate the inference speed of SwiftPillars, we perform three experiments to make a comprehensive comparison.

Comparison of Runtimes cross Different Methods This experiment is used to compare the performance of SwiftPillars with previous methods in terms of inference speed. The results are shown in Table 4, where our SwiftPillars implementation reaches the current state-of-the-art level of 85FPS while achieving 78.58% mAP. Compared to the state-of-the-art OcTr model with the highest accuracy, SwiftPillars achieves a speed increase of 307% while exhibiting a 10.6% lower accuracy on moderate difficulty. Compared to some fast methods, our mAP achieves outperformance by 2.52% for PointPillars and 0.73% for TANet on moderate difficulty, while being faster than them. As a result, SwiftPillars achieves a better balance between accuracy and speed.

SwiftPillars vs PointPillars cross Platforms This experiment is conducted to evaluate the performance of the end-to-end inference time of SwiftPillars and PointPillars across different platforms and also verifies that SwiftPillars can achieve multi-platform compatibility, which has high practical application value. From Table 5, it is evident that SwiftPillars achieves faster inference speeds than PointPillars across all platforms. Notably, on the KITTI dataset using NVIDIA Tesla A2, a remarkable 100.2% speed improvement is achieved. Across all platforms, the average speed improvement reached 24.34%. We noticed a considerable variation in the speed improvement across different GPUs. This difference is due to varying memory-bound characteristics arising from distinct GPU memory and core architectures. And, the primary reason SwiftPillars achieves faster speeds is by removing memory-bound features related to matrix transpose.

SwiftPillars vs PointPillars cross Inputs Utilizing TensorRT exec on the NVIDIA Tesla T4 platform with CUDA 11.4 and TensorRT 8.2.0, we conduct a comparative analysis of the inference speeds of SPE and PFN across different parameters. As illustrated in Fig. 4, the inference speeds of

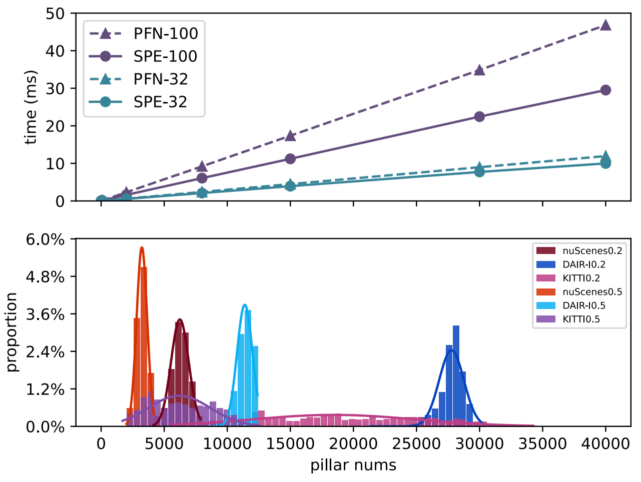

Figure 4: Performance of inference speed for different numbers of pillars and points. Where the X-label of pillar nums refers to the number of non-empty pillars in the input network; 0.2 and 0.5 correspond to grid sizes of 0.2 and 0.5, respectively; and, the numbers following SPE and PFN (e.g., SPE-32), respectively, indicate the maximum number of points within an individual pillar.

both SPE and PFN rise with increased pillar numbers and point numbers. Moreover, the growth rate of PFN is bigger than SPE. The main reason for this difference is that as the input dimensions increase, the feature extraction network needs to deal with larger dimensional tensors, i. e., the number of elements, besides, PFN has a higher memory-bound demand. We sample and compute the number of non-empty pillars at voxel sizes of 0.2 and 0.5 for the DAIR-V2X, KITTI, and nuScenes datasets. This approach provides insight into the real-world inference scenarios for both PFN and SPE based on dataset quantity distribution. For instance, considering the DAIR-V2X dataset with the voxel size of 0.2, the pillar count falls within the range of 25000-30000. In this scenario, the inference speed of SPE is 16%/55% faster than PFN for point numbers 32/100 within each pillar, demonstrating the effectiveness of our SPE.

Conclusion

This paper proposes a novel and efficient Lidar-based 3D object detector, including two essential modules: Swift Pillar Encoder (SPE) and Multi-scale Aggregation Decoder (MAD). SPE encodes pillars efficiently using the Dual-attention Module and lightweight operators. MAD achieves multi-scale feature aggregation. Through extensive experiments, we demonstrate the effectiveness of SwiftPillars in terms of both speed and accuracy. Among all known methods, SwiftPillars achieves the fastest inference speed while maintaining accuracy. Specifically, our SwiftPilars is on average 24.34% faster than PointPillars and even 100.2% faster on NVIDIA Tesla A2. Additionally, our method outperforms PointPillars in mAP by 3.2%. We aspire for SwiftPillars to serve as a robust baseline for future real-time perception applications.

Acknowledgments

This work was supported in part by the following grants: 1) National Natural Science Foundation of China (No. 52172380); 2) National Key Research and Development Program of China (No. 2021YFB2501200).

References

Bai, X.; Hu, Z.; Zhu, X.; Huang, Q.; Chen, Y.; Fu, H.; and Tai, C.-L. 2022. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 1090–1099.

Bhattacharyya, P.; Huang, C.; and Czarnecki, K. 2021. Sa-det3d: Self-attention based context-aware 3d object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, 3022–3031.

Caesar, H.; Bankiti, V.; Lang, A. H.; Vora, S.; Liong, V. E.; Xu, Q.; Krishnan, A.; Pan, Y.; Baldan, G.; and Beijbom, O. 2020. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 11621–11631.

Chen, C.; Chen, Z.; Zhang, J.; and Tao, D. 2022a. Sasa: Semantics-augmented set abstraction for point-based 3d object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 221–229.

Chen, Y.; Li, Y.; Zhang, X.; Sun, J.; and Jia, J. 2022b. Focal sparse convolutional networks for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5428–5437.

Chen, Y.; Liu, J.; Zhang, X.; Qi, X.; and Jia, J. 2023a. LargeKernel3D: Scaling Up Kernels in 3D Sparse CNNs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13488–13498.

Chen, Y.; Liu, J.; Zhang, X.; Qi, X.; and Jia, J. 2023b. Voxelnext: Fully sparse voxelnet for 3d object detection and tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 21674–21683.

Chen, Y.; Yu, Z.; Chen, Y.; Lan, S.; Anandkumar, A.; Jia, J.; and Alvarez, J. 2023c. FocalFormer3D: Focusing on Hard Instance for 3D Object Detection. *arXiv preprint arXiv:2308.04556*.

Deng, J.; Shi, S.; Li, P.; Zhou, W.; Zhang, Y.; and Li, H. 2021. Voxel r-cnn: Towards high performance voxel-based 3d object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 1201–1209.

Dong, S.; Ding, L.; Wang, H.; Xu, T.; Xu, X.; Wang, J.; Bian, Z.; Wang, Y.; and Li, J. 2022. MsSVT: Mixed-scale sparse voxel transformer for 3d object detection on point clouds. *Advances in Neural Information Processing Systems*, 35: 11615–11628.

Fan, L.; Pang, Z.; Zhang, T.; Wang, Y.-X.; Zhao, H.; Wang, F.; Wang, N.; and Zhang, Z. 2022a. Embracing single stride 3d object detector with sparse transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 8458–8468.

Fan, L.; Wang, F.; Wang, N.; and ZHANG, Z.-X. 2022b. Fully sparse 3d object detection. *Advances in Neural Information Processing Systems*, 35: 351–363.

Feng, X.; Du, H.; Fan, H.; Duan, Y.; and Liu, Y. 2023. Seformer: Structure embedding transformer for 3d object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 632–640.

Geiger, A.; Lenz, P.; and Urtasun, R. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, 3354–3361. IEEE.

He, C.; Zeng, H.; Huang, J.; Hua, X.-S.; and Zhang, L. 2020. Structure aware single-stage 3d object detection from point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 11873–11882.

Hu, J. S.; Kuai, T.; and Waslander, S. L. 2022. Point density-aware voxels for lidar 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8469–8478.

Hu, Y.; Ding, Z.; Ge, R.; Shao, W.; Huang, L.; Li, K.; and Liu, Q. 2022. Afdetv2: Rethinking the necessity of the second stage for object detection from point clouds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 969–979.

Kuang, H.; Wang, B.; An, J.; Zhang, M.; and Zhang, Z. 2020. Voxel-FPN: Multi-scale voxel feature aggregation for 3D object detection from LIDAR point clouds. *Sensors*, 20(3): 704.

Lai, X.; Chen, Y.; Lu, F.; Liu, J.; and Jia, J. 2023. Spherical Transformer for LiDAR-based 3D Recognition. In *CVPR*.

Lang, A. H.; Vora, S.; Caesar, H.; Zhou, L.; Yang, J.; and Beijbom, O. 2019. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 12697–12705.

Le, D. T.; Shi, H.; Rezatofighi, H.; and Cai, J. 2022. Accurate and real-time 3D pedestrian detection using an efficient attentive pillar network. *IEEE Robotics and Automation Letters*, 8(2): 1159–1166.

Lee, M.; Kim, H.; Park, S.; Yoon, M.; Lee, J.; Choi, J.; Kang, M.; and Choi, J. 2023. PillarAcc: Sparse PointPillars Accelerator for Real-Time Point Cloud 3D Object Detection on Edge Devices. *arXiv preprint arXiv:2305.07522*.

Li, J.; Dai, H.; Shao, L.; and Ding, Y. 2021. From voxel to point: Iou-guided 3d object detection for point cloud with voxel-to-point decoder. In *Proceedings of the 29th ACM International Conference on Multimedia*, 4622–4631.

Li, J.; Luo, C.; and Yang, X. 2023. PillarNeXt: Rethinking network designs for 3D object detection in LiDAR point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 17567–17576.

Liu, Z.; Zhao, X.; Huang, T.; Hu, R.; Zhou, Y.; and Bai, X. 2020. Tanet: Robust 3d object detection from point clouds with triple attention. In *Proceedings of the AAAI conference on artificial intelligence*, 07, 11677–11684.

Mahmoud, A.; Hu, J. S.; and Waslander, S. L. 2023. Dense voxel fusion for 3D object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 663–672.

Mao, J.; Niu, M.; Bai, H.; Liang, X.; Xu, H.; and Xu, C. 2021a. Pyramid r-cnn: Towards better performance and adaptability for 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2723–2732.

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

Qi, C. R.; Litany, O.; He, K.; and Guibas, L. J. 2019. Deep Hough Voting for 3D Object Detection in Point Clouds. In *Proceedings of the IEEE International Conference on Computer Vision*.

Qi, C. R.; Su, H.; Mo, K.; and Guibas, L. J. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 652–660.

Shi, G.; Li, R.; and Ma, C. 2022. Pillarnet: Real-time and high-performance pillar-based 3d object detection. In *European Conference on Computer Vision*, 35–52. Springer.

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. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 10529–10538.

Shi, S.; Jiang, L.; Deng, J.; Wang, Z.; Guo, C.; Shi, J.; Wang, X.; and Li, H. 2023. PV-RCNN++: Point-voxel feature set abstraction with local vector representation for 3D object detection. *International Journal of Computer Vision*, 131(2): 531–551.

Shi, S.; Wang, X.; and Li, H. 2019. PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud. arXiv:1812.04244.

Sun, P.; Tan, M.; Wang, W.; Liu, C.; Xia, F.; Leng, Z.; and Anguelov, D. 2022. Swformer: Sparse window transformer for 3d object detection in point clouds. In *European Conference on Computer Vision*, 426–442. Springer.

Tan, M.; Pang, R.; and Le, Q. V. 2020. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 10781–10790.

Team, O. D. 2020. OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds. https://github.com/open-mmlab/OpenPCDet.

Wang, H.; Chen, Z.; Cai, Y.; Chen, L.; Li, Y.; Sotelo, M. A.; and Li, Z. 2022. Voxel-RCNN-complex: An effective 3-D point cloud object detector for complex traffic conditions. *IEEE Transactions on Instrumentation and Measurement*, 71: 1–12.

Wang, H.; Shi, C.; Shi, S.; Lei, M.; Wang, S.; He, D.; Schiele, B.; and Wang, L. 2023. Dsvt: Dynamic sparse voxel transformer with rotated sets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13520–13529.

Wu, Y.; Zhang, S.; Ogai, H.; Inujima, H.; and Tateno, S. 2021. Realtime single-shot refinement neural network with adaptive receptive field for 3D object detection from LiDAR point cloud. *IEEE Sensors Journal*, 21(21): 24505–24519.

Xu, Q.; Zhong, Y.; and Neumann, U. 2022. Behind the curtain: Learning occluded shapes for 3d object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 2893–2901.

Yan, Y.; Mao, Y.; and Li, B. 2018. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10): 3337.

Yang, H.; Wang, W.; Chen, M.; Lin, B.; He, T.; Chen, H.; He, X.; and Ouyang, W. 2023. PVT-SSD: Single-Stage 3D Object Detector with Point-Voxel Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13476–13487.

Yang, Z.; Sun, Y.; Liu, S.; and Jia, J. 2020. 3dssd: Point-based 3d single stage object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 11040–11048.

Yin, T.; Zhou, X.; and Krahenbuhl, P. 2021. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 11784–11793.

Yu, C.; Lei, J.; Peng, B.; Shen, H.; and Huang, Q. 2022a. SIEV-Net: A structure-information enhanced voxel network for 3D object detection from LiDAR point clouds. *IEEE Transactions on Geoscience and Remote Sensing*, 60: 1–11.

Yu, H.; Luo, Y.; Shu, M.; Huo, Y.; Yang, Z.; Shi, Y.; Guo, Z.; Li, H.; Hu, X.; Yuan, J.; et al. 2022b. Dair-v2x: A large-scale dataset for vehicle-infrastructure cooperative 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 21361–21370.

Zhao, H.; Jiang, L.; Jia, J.; Torr, P. H.; and Koltun, V. 2021. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, 16259–16268.

Zheng, W.; Tang, W.; Chen, S.; Jiang, L.; and Fu, C.-W. 2021a. Cia-ssd: Confident iou-aware single-stage object detector from point cloud. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 3555–3562.

Zheng, W.; Tang, W.; Jiang, L.; and Fu, C.-W. 2021b. SE-SSD: Self-ensembling single-stage object detector from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14494–14503.

Zhou, C.; Zhang, Y.; Chen, J.; and Huang, D. 2023. OcTr: Octree-based Transformer for 3D Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5166–5175.

Zhou, Z.; Zhao, X.; Wang, Y.; Wang, P.; and Foroosh, H. 2022. Centerformer: Center-based transformer for 3d object detection. In *European Conference on Computer Vision*, 496–513. Springer.