DSVT: Dynamic Sparse Voxel Transformer with Rotated Sets

Haiyang Wang[1*] Chen Shi[5*] Shaoshuai Shi[2†] Meng Lei[1,4]
Sen Wang[3] Di He[5] Bernt Schiele[2] Liwei Wang[1,5†]

[1]Center for Data Science, Peking University

[2]Max Planck Institute for Informatics, Saarland Informatics Campus [3]Huawei [4]Zhejiang Lab

[5]National Key Laboratory of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University

{wanghaiyang@stu, shichen@stu, leimeng@stu, dihe@, wanglw@cis}.pku.edu.cn

{sshi, schiele}@mpi-inf.mpg.de wangsen31@huawei.com

Abstract

Designing an efficient yet deployment-friendly 3D backbone to handle sparse point clouds is a fundamental problem in 3D perception. Compared with the customized sparse convolution, the attention mechanism in Transformers is more appropriate for flexibly modeling long-range relationships and is easier to be deployed in real-world applications. However, due to the sparse characteristics of point clouds, it is non-trivial to apply a standard transformer on sparse points. In this paper, we present Dynamic Sparse Voxel Transformer (DSVT), a single-stride window-based voxel Transformer backbone for outdoor 3D perception. In order to efficiently process sparse points in parallel, we propose Dynamic Sparse Window Attention, which partitions a series of local regions in each window according to its sparsity and then computes the features of all regions in a fully parallel manner. To allow the cross-set connection, we design a rotated set partitioning strategy that alternates between two partitioning configurations in consecutive self-attention layers. To support effective downsampling and better encode geometric information, we also propose an attention-style 3D pooling module on sparse points, which is powerful and deployment-friendly without utilizing any customized CUDA operations. Our model achieves state-of-the-art performance with a broad range of 3D perception tasks. More importantly, DSVT can be easily deployed by TensorRT with real-time inference speed (27Hz). Code will be available at https://github.com/Haiyang-W/DSVT.

1. Introduction

3D perception is a crucial challenge in computer vision, garnering increased attention thanks to its potential applications in various fields such as autonomous driving systems [2, 45] and modern robotics [48, 58]. In this paper, we propose DSVT, a general-purpose and deployment-friendly Transformer-only 3D backbone that can be easily applied to various 3D perception tasks for point clouds processing.

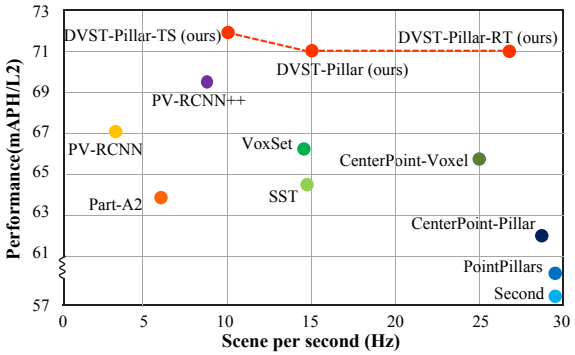


Figure 1. Detection performance (mAPH/L2) vs speed (Hz) of different methods on Waymo [40] validation set. All the speeds are evaluated on an NVIDIA A100 GPU with AMD EPYC 7513 CPU.

Unlike the well-studied 2D community where the input image is in a densely packed array, 3D point clouds are sparsely and irregularly distributed in continuous space due to the nature of 3D sensors, which makes it challenging to directly apply techniques used for traditional regular data. To support sparse feature learning from raw point clouds, previous methods mainly apply customized sparse operations, such as PointNet++ [31, 32] and sparse convolution [14, 15]. PointNet based methods [31, 32, 43] use point-wise MLPs with the ball-query and max-pooling operators to extract features. Sparse convolution based methods [7, 14, 15] first convert point clouds to regular grids and handle the sparse volumes efficiently. Though impressive, they suffer from either the intensive computation of sampling and grouping [32] or the limited representation capacity due to submanifold dilation [15]. More importantly, these specifically-designed operations generally can not be implemented with well-optimized deep learning tools (*e.g.*, PyTorch and TensorFlow) and require writing customized CUDA codes, which

*Equal contribution.
†Corresponding author.

greatly limits their deployment in real-world applications.

Witnessing the success of transformer [44] in the 2D domain, numerous attention-based 3D vision methods have been investigated in point cloud processing. However, because of the sparse characteristic of 3D points, the number of non-empty voxels in each local region can vary significantly, which makes directly applying a standard Transformer nontrivial. To efficiently process the attention on sparse data, many approaches rebalance the token number by random sampling [29, 54] or group local regions with similar number of tokens together [12, 41]. These methods are either inseparable from superfluous computations (*e.g.*, dummy token padding and non-parallel attention) or noncompetitive performance (*e.g.*, token random dropping). Alternatively, some approaches [18, 28] try to solve these problems by writing customized CUDA operations, which require heavy optimization to be deployed on modern devices. Hence, building an efficient and deployment-friendly 3D transformer backbone is the main challenge we aim to address in this paper.

In this paper, we seek to expand the applicability of Transformer such that it can serve as a powerful backbone for outdoor 3D perception, as it does in 2D vision. Our backbone is efficient yet deployment-friendly without any customized CUDA operations. To achieve this goal, we present two major modules, one is the dynamic sparse window attention to support efficient parallel computation of local windows with diverse sparsity, and the other one is a novel learnable 3D pooling operation to downsample the feature map and better encode geometric information.

Specifically, as illustrated in Figure 2, given the sparse voxelized representations and window partition, we first divide each window's sparse voxels into some non-overlapped subsets, where each subset is guaranteed to have the same number of voxels for parallel computation. The partitioning configuration of these subsets will be changed in consecutive self-attention layers based on the rotating partition axis between the X-Axis and Y-Axis. It bridges the subsets of preceding layers for intra-window fusion, providing connections among them that significantly enhance modeling power (see Table 6). To better process the inter-window fusion and encode multi-scale information, we propose the hybrid window partition, which alternates its window shape in successive transformer blocks. It leads to a drop in computation cost with even better performance (see Table 7). With the above designs, we process all regions in a fully parallel manner by calculating them in the same batch. This strategy is efficient in regards to real-world latency: i) all the windows are processed in parallel, which is nearly independent of the voxel distribution, ii) using self-attention without key duplication, which facilitates memory access in hardware. Our experiments show that the dynamic sparse window attention approach has much lower latency than previous bucketing-based strategies [12, 41] or vanilla window attention [24],

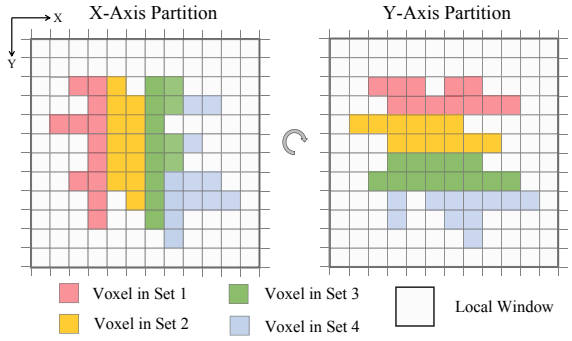


Figure 2. A demonstration of dynamic sparse window attention in our DSVT block. In the X-Axis DSVT layer, the sparse voxels will be split into a series of window-bounded and size-equivalent subsets in X-Axis main order, and self-attention is computed within each set. In the next layer, the set partition is switched to Y-Axis, providing connections among the previous sets.

yet is similar in modeling power (see Table 5).

Secondly, we present a powerful yet deployed-friendly 3D sparse pooling operation to efficiently process downsampling and encode better geometric representation. To tackle the sparse characteristic of 3D point clouds, previous methods adopt some custom operations, *e.g.*, customized scatter function [13] or strided sparse convolution to generate downsampled feature volumes [51, 55]. The requirement of heavy optimization for efficient deployment limits their real-world applications. More importantly, we empirically find that inserting some linear or max-pooling layers between our transformer blocks also harms the network convergence and the encoding of geometric information. To address the above limitations, we first convert the sparse downsampling region into dense and process an attention-style 3D pooling operation to automatically aggregate the local spatial features. Our 3D pooling module is powerful and deployed-friendly without any self-designed CUDA operations, and the performance gains (see Table 8) demonstrate its effectiveness.

In a nutshell, our contributions are four-fold: 1) We propose Dynamic Sparse Window Attention, a novel window-based attention strategy for efficiently handling sparse 3D voxels in parallel. 2) We present a learnable 3D pooling operation, which can effectively downsample sparse voxels and encode geometric information better. 3) Based on the above key designs, we introduce an efficient yet deployment-friendly transformer 3D backbone without any customized CUDA operations. It can be easily accelerated by NVIDIA TensorRT to achieve real-time inference speed (*27Hz*), as shown in Figure 1. 4) Our approach outperforms previous state-of-the-art methods on the large-scale Waymo [39] and nuScenes [4] datasets across various 3D perception tasks.

2. Related Work

3D Perception on Point Clouds. Previous 3D perception methods can be coarsely classified into point-based meth-

ods and voxel-based methods in terms of point representations. Thanks to the powerful PointNet series [31, 32], point-based methods [6, 26, 30, 37, 47, 52] have become extensively employed to extract geometric features directly from raw point clouds. However, these methods suffer from the time-consuming process of point sampling and neighbor searching, and computation-intensive point-wise feature duplication. Voxel-based methods [9, 10, 16, 35, 36, 38, 46, 51, 53] are mainly applied in outdoor autonomous driving scenarios. The input point clouds are first converted into regular 3D voxels and then processed by 3D sparse convolution. Though impressive, to avoid computation expansion, these methods usually adopt submanifold sparse convolution, which greatly decreases the receptive field and limits its representation capacity. More importantly, both the PointNet++ and sparse convolution are implemented with customized CUDA codes, which require heavy optimization for being deployed in real-world applications. To tackle this problem, we propose an efficient yet deployment-friendly point cloud processor for boosting the development of 3D perception.

Transformer for 3D Perception. Motivated by the significant success of Transformer [44] in computer vision community, many researchers have tried to introduce this architecture into point cloud processing. Due to the sparse nature of point clouds, how to efficiently apply a standard transformer is non-trivial. VoTr [28] first proposes local attention and dilated attention with a self-designed voxel query to enable attention mechanism on sparse voxels. SST [12] and SWFormer [41] batch regions with similar number of tokens together and pads them separately to implement parallel computation. Although these methods have explored various sparse attention strategies, they are inevitable to redundant computations (*e.g.* non-parallel attention, token duplication) or uncompetitive performance (*e.g.* random token dropping and sampling). Besides that, most of these methods are also highly dependent on some self-designed CUDA operations (*e.g.*, scatter function [18] and query function [28]), which greatly limits their real-world application. To address these problems, we present dynamic sparse window attention, an efficient attention mechanism that computes all the sparse tokens in the same batch in parallel only based on the well-optimized deep learning framework (*e.g.*, PyTorch).

3. Methodology

3.1. Overview

An overview of the DSVT architecture is presented in Figure 3, which illustrates the pillar version (DSVT-P). It first converts the input point clouds into sparse voxels by a voxel feature encoding (VFE) module, like previous voxel-based methods [41, 50, 53]. Each voxel will be treated as a "token". Considering the sufficient receptive field of transformer and the tiny scale of outdoor objects, instead of using hierarchical representation, we follow [12] to adopt a single-stride network design, which doesn't reduce the scale of the feature map in X/Y-Axis and is demonstrated to perform better in outdoor 3D Object Detection. With this design, several voxel transformer blocks with dynamic sparse window attention (*DSVT Block*) are applied on these voxels. To provide connections among the sparse voxels, we design two partitioning approaches: rotated set and hybrid window (described in §3.2), which introduce intra- and inter-window feature propagation while maintaining efficient computation. To support a 3D variant and better encode accurate 3D geometric information without any extra CUDA operations, a learnable 3D pooling module is designed for effective downsampling (See §3.3). Voxel features extracted by our proposed DSVT are then projected to a bird's eye view (BEV) feature map. In the end, various perception heads can be attached for diverse 3D perception tasks (See §3.4). Our proposed architecture can seamlessly substitute the 3D backbone in existing methods, enhancing outdoor 3D perception performance.

3.2. Dynamic Sparse Window Attention

Window-based Transformer architectures [11, 24] have made great success in 2D object detection. However, since point clouds are sparsely distributed in 3D space, many voxels are empty with no valid points. Therefore, the number of non-empty voxels in each window may vary significantly, which makes directly applying a standard Transformer non-trivial. The computation cost of simply padding the sparse volume to dense in each window is expensive (See Table 5). Previous methods [12, 18, 41] try to rebalance the voxel number by sampling [29, 54] or grouping regions with a similar number of tokens together [12, 41]. Though impressive, they suffer from redundant computations or unstable performance. Alternatively, some approaches write customized CUDA operations to alleviate these problems, which generally have inferior running speed than well-optimized native operations in deep learning frameworks (*e.g.*, PyTorch), limiting their deployment in real-world applications.

To address the above limitations, we propose Dynamic Sparse Window Attention, a window-based attention strategy for efficiently handling sparse 3D voxels in parallel. Notably, it is all implemented by well-optimized native operations in deep learning tools without any self-designed CUDA operations, thus is friendly to be deployed on modern GPUs.

Dynamic set partition. To efficient perform standard attention among the given sparse voxels inside each window, we reformulate it as parallel computing self-attention within a series of window-bounded and size-equivalent subsets.

Specifically, after converting points to 3D voxels, they are further partitioned into a list of non-overlapping 3D windows with fixed size $L \times W \times H$, like previous window-based approaches [12, 24, 41]. As for a specific window, it has N non-empty voxels, *i.e.*,

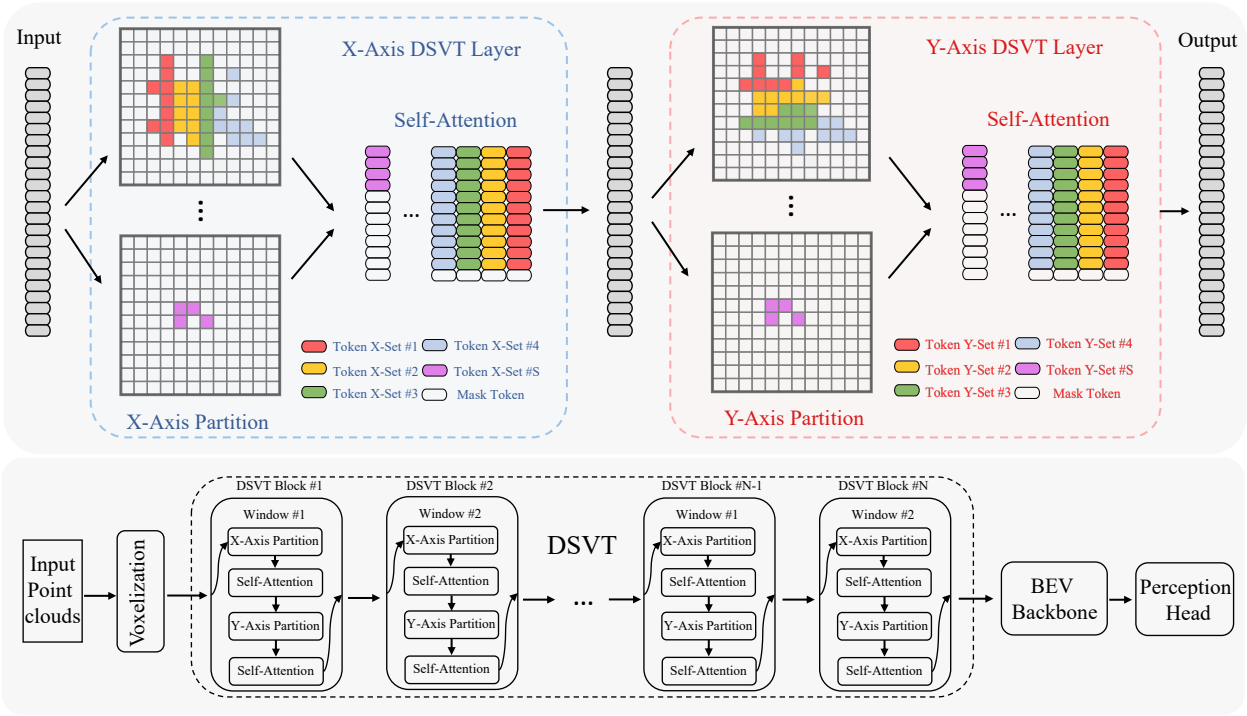


Figure 3. **Top**: An illustration of the Dynamic Sparse Voxel Transformer block, including one X-Axis DSVT Layer and one Y-Axis DSVT Layer with different set partitions. **Bottom**: The overall architecture of our proposed DSVT for 3D perception from the point cloud.

$$\mathcal{V} = \{v_i | v_i = [(x_i, y_i, z_i); f_i; d_i]\}_{i=1}^{N}, \tag{1}$$

where $(x_i, y_i, z_i) \in \mathbb{R}^3$ and $f_i \in \mathbb{R}^C$ denote the coordinates and features of sparse voxels, respectively. $d_i \in [1, N]$ is the corresponding inner-window voxel ID, which can be generated by specifying a sorting strategy of these voxels. To generate non-overlapped and size-equivalent local sets, we first compute the required number of sub-sets in this window, as follows:

$$S = \lfloor \frac{N}{\tau} \rfloor + \mathbb{I}[(N \,\%\, \tau) > 0], \tag{2}$$

where $\lfloor \cdot \rfloor$ is the floor function and $\mathbb{I}[\cdot]$ is the indicator function. τ is a hyperparameter that indicates the maximum number of non-empty voxels allocated to each set. In this way, we can cover all the voxels in this window with a minimum number of subsets. Notably, S dynamically varies with the sparsity of the window. The more non-empty voxels, the more sets and computation resources will be assigned to process this window, which is the key design of our dynamic sparse window attention.

With the number of assigned sets S, we evenly distribute N non-empty voxels into S sets. Specifically, for the voxel indices that belong to j-th set (denoted as $\mathcal{Q}_j = \{q_k^j\}_{k=0}^{\tau-1}$), we compute its k-th index as

$$q_k^j = \left\lfloor \frac{(j \times \tau + k)}{S \times \tau} \times N \right\rfloor, \quad \text{for } k = 0, ..., \tau - 1, \tag{3}$$

where $\lfloor \cdot \rfloor$ is the floor function. Note that this operation can generate a specific number (*i.e.*, τ) of voxels for each set, regardless of N, which enables the attention layer can be performed on all sets in a fully parallel manner. Although Eq. (9) may duplicate some voxels within each set to achieve the number of τ voxels, these redundant voxels will be masked when conducting attention.

After obtaining the partition $\mathcal{Q}_j$ of j-th set, we then collect the corresponding voxel features and coordinates based on the voxel inner-window id $\mathcal{D} = \{d_i\}_{i=1}^{N}$, as follows,

$$\mathcal{F}_j, \mathcal{O}_j = \text{INDEX}(\mathcal{V}, \mathcal{Q}_j, \mathcal{D}), \tag{4}$$

where $\text{INDEX}(\cdot_{\text{voxels}}, \cdot_{\text{partition}}, \cdot_{\text{ID}})$ is the index operation, $\mathcal{F}_j \in \mathbb{R}^{\tau \times C}$ and $\mathcal{O}_j \in \mathbb{R}^{\tau \times 3}$ are the corresponding voxel features and spatial coordinates (x, y, z) of this set.

In this way, we obtain some non-overlapped and window-bounded subsets with the same number of sparse voxels. Notably, our dynamic set partition is highly dependent on the inner-window voxel ID (see Eq. (4)), so we can easily control the covered local region of each set by voxel ID reordering with different sorting strategies.

Rotated set attention for intra-window feature propagation. The above dynamic set partition can group sparse voxels into non-overlapped and window-bounded subsets, where each subset has the same number of voxels that enables to conduct self-attention layers within each subset in a fully parallel manner. However, computing self-attention inside the invariant partition lacks connections across the subsets, limiting its modeling power. To bridge the voxels among the non-overlapping sets while maintaining efficient

computation, we propose the rotated-set attention approach that alternates between two partitioning configurations in consecutive attention layers.

As illustrated in Figure 3, our DSVT block contains two self-attention layers. And the first layer uses the X-Axis Partition, in which the voxel ID is sorted according to their coordinates in X-Axis main order. The next layer adopts a rotated partition configuration that is sorted in Y-Axis main order. With the above design, DSVT block is computed as

$$\begin{aligned}
&\mathcal{F}^l, \mathcal{O}^l = \text{INDEX}(\mathcal{V}^{l-1}, \{\mathcal{Q}_j\}_{j=0}^{S-1}, \mathcal{D}_x), \\
&\mathcal{V}^l = \text{MHSA}(\mathcal{F}^l, \text{PE}(\mathcal{O}^l)), \\
&\mathcal{F}^{l+1}, \mathcal{O}^{l+1} = \text{INDEX}(\mathcal{V}^l, \{\mathcal{Q}_j\}_{j=0}^{S-1}, \mathcal{D}_y), \\
&\mathcal{V}^{l+1} = \text{MHSA}(\mathcal{F}^{l+1}, \text{PE}(\mathcal{O}^{l+1})),
\end{aligned} \tag{5}$$

where $\mathcal{D}_x$ and $\mathcal{D}_y$ are the inner-window voxel index sorted in X-Axis and Y-Axis respectively. MHSA$(\cdot)$ denotes the conventional Multi-head Self-Attention layer with FFN and Layer Normalization. PE$(\cdot)$ stands for the positional encoding function used in [24]. $\mathcal{F} \in \mathbb{R}^{S\times\tau\times C}$ and $\mathcal{O} \in \mathbb{R}^{S\times\tau\times 3}$ are the corresponding indexed voxel features and coordinates of all sets. To this end, the original sparse window attention will be approximatively reformulated as several set attention, which can be processed in the same batch in parallel. In this way, it is easy to implement an efficient DSVT module in current popular deep learning frameworks without extra engineering efforts of deployment.

The rotated set partition introduces connections between non-overlapping sets in the previous layer and is found to be effective in outdoor 3D perception (see Table 6).

Hybrid window partition for inter-window feature propagation. To connect the voxels across windows and reduce the computation cost, we present the hybrid window partition approaches at the block level. We observe that our dynamic window attention layer has linear computational complexity with respect to the total number of subsets. However, increasing the window sizes will reduce the number of sets and lead to the drop of self-attention's computation cost, but also destroy the detection performance for small objects. This motivates us to adopt the hybrid window splitting to deliver good performance-efficiency trade-off.

Specifically, we follow the swin-transformer [24] using a window shifting technique between two consecutive DSVT blocks to re-partition the sparse window, but their window size is different. With this design, our block can effectively save computation costs without sacrificing performance.

3.3. Attention-style 3D Pooling

Compared with the pillar-style 2D encoder, the voxel-based 3D backbone can capture more precise position information, which is beneficial for 3D perception. However, we observed that simply padding the sparse regions and applying an MLP network for downsampling as [24] will drop the performance (See Table 8). It is challenging for one-layer MLP to learn the sparse geometric information with too many zero-padded holes. Moreover, we observe that inserting an MLP between our transformer blocks also harms network optimization. To support effective 3D downsampling and better encode spatial information, we present an attention-style 3D pooling operation, which is compatible with our full attention setting and implemented without any self-design CUDA operations.

Given a pooling local region $l \times w \times h$, (*e.g.*, $2 \times 2 \times 2$), with P non-empty voxels $\{p_i\}_{i=1}^{P}$, we first pad this sparse region into dense, $\{\widetilde{p}_i\}_{i=1}^{l\times w\times h}$, and perform standard max-pooling along the voxel dimension as

$$\mathcal{P} = \text{MaxPool}(\{\widetilde{p}_i\}_{i=1}^{P}). \tag{6}$$

Instead of directly feeding the pooled feature to the succeeding module, we use pooled feature $\mathcal{P}$ to construct the query vector, while the original unpooled $\{\widetilde{p}_i\}_{i=1}^{P}$ serves that role of key and value vectors, *i.e.*,

$$\widetilde{\mathcal{P}} = \text{Attn}(\text{Q} = \mathcal{P}, \text{KV} = \{\widetilde{p}_i\}_{i=1}^{l\times w\times h}) \tag{7}$$

With this attention-style 3D pooling operation, our 3D backbone holds the characteristic of fully attention and achieves better performance than our pillar variant.

3.4. Perception Model

Backbone Variants We build our base backbone model, called DSVT-P, a single-stride pillar-based sparse voxel backbone that has a similar model size to the widely-used sparse convolution backbone [51, 55]. We also introduce DSVT-V, a single-stride voxel-based 3D backbone that only gradually reduces the size of the feature map in the Z-Axis by our attention-style 3D pooling module. Notably, the output voxel number and corresponding coordinates of these two variants are equal, which can be easily used to ablate the effectiveness of our pooling module.

Detection head and Training Objectives. Our DSVT is flexible and can be applied in most voxel-based detection frameworks [22, 53]. In the Waymo variant, we follow the framework of CenterPoint-Pillar [53] and append our DSVT before BEV backbone. For nuScenes, we simply replace the 3D encoder of Transfusion-L [1] with DSVT. Besides that, we also follow [20, 23, 34] that uses IoU-rectification scheme to incorporate the IoU information into confidence scores.

Two-stage model. Although our main contributions focus on the design of the sparse voxel transformer architecture in the backbone network, there is still a considerable gain between one-stage and two-stage detectors. To match the performance gap, we provide a two-stage version with CT3D [33], which is a widely used two-stage framework that performs proposal-aware aggregation from point clouds.

Segmentation head. We follow the lidar branch of BEVFusion [25] in map segmentation task and only switch the 3D backbone while keeping other components unchanged.

Methods	Present at	Stages	mAP/mAPH L1	mAP/mAPH L2	*Vehicle* 3D AP/APH L1	*Vehicle* 3D AP/APH L2	*Pedestrian* 3D AP/APH L1	*Pedestrian* 3D AP/APH L2	*Cyclist* 3D AP/APH L1	*Cyclist* 3D AP/APH L2
SECOND [51]	Sensors'18	One	67.2/63.1	61.0/57.2	72.3/71.7	63.9/63.3	68.7/58.2	60.7/51.3	60.6/59.3	58.3/57.0
PointPillars‡ [22]	CVPR'19	One	69.0/63.5	62.8/57.8	72.1/71.5	63.6/63.1	70.6/56.7	62.8/50.3	64.4/62.3	61.9/59.9
CenterPoint-Voxel† [53]	CVPR'21	One	74.4/71.7	68.2/65.8	74.2/73.6	66.2/65.7	76.6/70.5	68.8/63.2	72.3/71.1	69.7/68.5
SST‡ [12]	CVPR'22	One	74.5/71.0	67.8/64.6	74.2/73.8	65.5/65.1	78.7/69.6	70.0/61.7	70.7/69.6	68.0/66.9
VoxSet [18]	CVPR'22	One	75.4/72.2	69.1/66.2	74.5/74.0	66.0/65.6	80.0/72.4	72.5/65.4	71.6/70.3	69.0/67.7
AFDetV2 [19]	AAAI'22	One	77.2/74.8	71.0/68.8	77.6/77.1	69.7/69.2	80.2/74.6	72.2/67.0	73.7/72.7	71.0/70.1
SWFormer [41]	ECCV'22	One	-/-	-/-	77.8/77.3	69.2/68.8	80.9/72.7	72.5/64.9	-/-	-/-
PillarNet-34 [34]	ECCV'22	One	77.3/74.6	71.0/68.5	79.1/78.6	70.9/70.5	80.6/74.0	72.3/66.2	72.3/71.2	69.7/68.7
CenterFormer [56]	ECCV'22	One	75.3/72.9	71.1/68.9	75.0/74.4	69.9/69.4	78.6/73.0	73.6/68.3	72.3/71.3	69.8/68.8
Ours (Pillar)	-	One	**79.5/77.1**	**73.2/71.0**	**79.3/78.8**	**70.9/70.5**	**82.8/77.0**	**75.2/69.8**	**76.4/75.4**	**73.6/72.7**
Ours (Voxel)	-	One	**80.3/78.2**	**74.0/72.1**	**79.7/79.3**	**71.4/71.0**	**83.7/78.9**	**76.1/71.5**	**77.5/76.5**	**74.6/73.7**
PV-RCNN† [35]	CVPR'20	Two	76.2/73.6	69.6/67.2	78.0/77.5	69.4/69.0	79.2/73.0	70.4/64.7	71.5/70.3	69.0/67.8
Part-A2-Net [38]	TPAMI'20	Two	73.6/70.3	66.9/63.8	77.1/76.5	68.5/68.0	75.2/66.9	66.2/58.6	68.6/67.4	66.1/64.9
CenterPoint-Voxel [53]	CVPR'21	Two	-/-	-/-	76.7/76.2	68.8/68.3	79.0/72.9	71.0/65.3	-/-	-/-
PV-RCNN++(center) [36]	IJCV'22	Two	78.1/75.9	71.7/69.5	79.3/78.8	70.6/70.2	81.3/76.3	73.2/68.0	73.7/72.7	71.2/70.2
FSD [13]	NeurIPS'22	Two	79.6/77.4	72.9/70.8	79.2/78.8	70.5/70.1	82.6/77.3	73.9/69.1	77.1/76.0	74.4/73.3
Ours (Pillar-TS)	-	Two	**80.6/78.2**	**74.3/72.1**	**80.2/79.7**	**72.0/71.6**	**83.7/78.0**	**76.1/70.7**	**77.8/76.8**	**74.9/73.9**
Ours (Voxel-TS)	-	Two	**81.1/78.9**	**74.8/72.8**	**80.4/79.9**	**72.2/71.8**	**84.2/79.3**	**76.5/71.8**	**78.6/77.6**	**75.7/74.7**
CenterFormer(2f) [56]	ECCV'22	One	78.3/76.7	74.3/72.8	77.0/76.5	72.1/71.6	81.4/78.0	76.7/73.4	76.6/75.7	74.2/73.3
Ours (Pillar-2f)	-	One	**81.4/79.8**	**75.4/73.9**	**80.8/80.3**	**72.7/72.3**	**84.5/81.3**	**77.2/74.1**	**78.8/77.9**	**76.3/75.4**
Ours (Voxel-2f)	-	One	**81.9/80.4**	**76.0/74.6**	**81.1/80.6**	**73.0/72.6**	**84.9/81.7**	**77.8/74.8**	**79.8/78.9**	**77.3/76.4**
SST(3f) [12]	CVPR'22	One	-/-	-/-	77.0/76.6	68.5/68.1	82.4/78.0	75.1/70.9	-/-	-/-
Ours (Pillar-3f)	-	One	**81.9/80.5**	**76.2/74.8**	**81.2/80.8**	**73.3/72.9**	**85.0/82.0**	**78.0/75.0**	**79.6/78.8**	**77.2/76.4**
Ours (Voxel-3f)	-	One	**82.1/80.8**	**76.3/75.0**	**81.5/81.1**	**73.6/73.2**	**85.3/82.4**	**78.2/75.4**	**79.6/78.8**	**77.2/76.4**
CenterFormer(4f) [56]	ECCV'22	One	78.5/77.0	74.7/73.2	78.1/77.6	73.4/72.9	81.7/78.6	77.2/74.2	75.6/74.8	73.4/72.6
Ours (Pillar-4f)	-	One	**82.5/81.0**	**76.7/75.3**	**81.7/81.2**	**73.8/73.4**	**85.4/82.3**	**78.5/75.5**	**80.3/79.4**	**77.9/77.1**
Ours (Voxel-4f)	-	One	**82.6/81.3**	**76.9/75.6**	**81.8/81.4**	**74.1/73.6**	**85.6/82.8**	**78.6/75.9**	**80.4/79.6**	**78.1/77.3**
PV-RCNN++(2f)† [36]	IJCV'22	Two	79.4/78.0	73.3/71.9	80.2/79.7	72.1/71.7	83.5/80.4	75.5/72.6	74.6/73.8	72.4/71.5
Ours (Pillar-TS-2f)	-	Two	**81.9/80.4**	**76.0/74.5**	**81.3/80.9**	**73.4/73.0**	**85.2/81.9**	**77.9/74.7**	**79.2/78.3**	**76.7/75.9**
Ours (Voxel-TS-2f)	-	Two	**82.3/80.8**	**76.6/75.1**	**81.4/81.0**	**73.5/73.1**	**85.4/82.2**	**78.4/75.3**	**80.2/79.3**	**77.8/76.9**
SST_TS(3f) [12]	CVPR'22	Two	-/-	-/-	78.7/78.2	70.0/69.6	83.8/80.1	75.9/72.4	-/-	-/-
Ours (Pillar-TS-3f)	-	Two	**82.5/81.0**	**76.7/75.4**	**81.8/81.3**	**74.0/73.6**	**85.6/82.6**	**78.5/75.6**	**80.1/79.2**	**77.7/76.9**
Ours (Voxel-TS-3f)	-	Two	**82.6/81.2**	**76.8/75.5**	**81.8/81.4**	**74.0/73.6**	**85.8/82.9**	**78.8/75.9**	**80.1/79.2**	**77.7/76.9**
MPPNet (4f) [5]	ECCV'22	Two	81.1/79.9	75.4/74.2	81.5/81.1	74.1/73.6	84.6/82.0	77.2/74.7	77.2/76.5	75.0/74.4
Ours (Pillar-TS-4f)	-	Two	**82.9/81.5**	**77.3/75.9**	**82.1/81.6**	**74.4/74.0**	**85.8/82.8**	**79.0/76.1**	**80.9/80.0**	**78.6/77.7**
Ours (Voxel-TS-4f)	-	Two	**83.1/81.7**	**77.5/76.2**	**82.1/81.6**	**74.5/74.1**	**86.0/83.2**	**79.1/76.4**	**81.1/80.3**	**78.8/78.0**

Table 1. The results of 3D object detection on Waymo Open validation set (100% training data). "2f", "3f" and "4f" stand for 2-, 3- and 4-frame models. †: re-implemented by OpenPCDet. ‡: reported by [13]. We highlight the top-2 entries with **bold** font in each column.

Methods	Present at	*val* NDS	*val* mAP	*test* NDS	*test* mAP
PointPillars [22]	CVPR'19	-	-	45.3	30.5
CBGS [57]	ArXiv'19	62.3	50.6	63.3	52.8
CenterPoint-Voxel [53]	CVPR'21	66.8	59.6	67.3	60.3
Transfusion-L [1]	CVPR'22	69.3	64.7	70.2	65.5
PillarNet-34 [34]	ECCV'22	-	-	71.4	66.0
Ours (Pillar)	-	**71.1**	**66.4**	**72.7**	**68.4**

Table 2. The results of 3D object detection on nuScenes Dataset.

Encoder	DA	PC	WW	SL	CP	DI	mIoU
2D Conv [25]	72.0	43.1	53.1	29.7	27.7	37.5	43.8
3D SpConv [25]	75.6	48.4	57.5	36.5	31.7	41.9	48.6
Ours (Pillar)	**79.7**	**51.8**	**61.1**	**38.2**	**33.8**	**45.3**	**51.6**
Ours (Pillar)†	**87.6**	**67.2**	**72.7**	**59.7**	**62.7**	**58.2**	**68.0**

Table 3. The results of BEV map segmentation on nuScenes(val). All the baselines are reported by [25]. 1^{st}-3^{rd} rows adopt the same segmentation framework except for 3D encoder. † means a deeper BEV backbone. Notion: Drivable(DA), Ped.Cross.(PC), Walkway(WW), StopLine(SL), Carpark(CP), Divider(DI).

4. Experiments

Waymo Open [39] is a widely used outdoor 3D perception benchmark consisting of 1150 point cloud sequences in total (more than 200K frames). Each frame covers a large perception range (150m $\times$ 150m). All results are evaluated by the standard protocol using 3D mean Average Precision (mAP) and its weighted variant by heading accuracy (mAPH).

NuScenes [4] is a challenging outdoor dataset providing diverse annotations for various tasks, (*e.g.*, 3D object detection and BEV map segmentation). It contains 40,157 annotated samples, each with six cameras and a 32-beam LiDAR scan. For 3D object detection, we report the nuScenes detection score (NDS) and mean average precision (mAP), while the mean IoU is provided for the map segmentation task.

4.1. Implementation Details.

Network Architecture. For the DSVT-P variant, we build our backbone with 4 blocks and each block consists of two

Backbone	#param.	Latency	L2 mAP	L2 mAPH
ResBackbone1x	88M	56ms	69.61	66.81
DSVT(Pillar)	71M	60ms	**71.14**	**68.59**

Table 4. Comparison with sparse convolution. Only switch the 3D backbone while other components remain unchanged.

Method	Latency	PadRatio	Memory	L_2 mAP	L_2 mAPH
Fully-Padding	77ms	-	7974MB	64.17	60.94
Bucketing [12]	52ms	35.1%	3566MB	64.07	60.86
Ours (12 × 12)	30ms	38.9%	3688MB	**64.41**	**61.22**
Ours (12 × 24)	29ms	28.3%	3640MB	**64.59**	**61.40**

Table 5. Comparison with standard self-attention. We report the latency of transformer backbone based on the code base of SST [12].

DSVT attention layers. In case of Waymo 3D object detection, we use the grid size of (0.32m, 0.32m, 6m) following CenterPoint-Pillar [53]. The hybrid window sizes are set to (12, 12, 1) and (24, 24, 1), and the maximum number of voxels assigned to each set is 36. For the DSVT-V, this variant is similar to the pillar-based framework except for splitting the Z-Axis into more voxels. Specifically, its backbone has four stages with block numbers $\{1, 1, 1, 1\}$. The voxel size is held as (0.32m, 0.32m, 0.1875m) and we downsample the voxel feature map using our attention-style 3D pooling module only in the Z-Axis with the stride $\{4, 4, 2\}$. The window sizes along the Z-Axis are $\{32, 8, 2, 1\}$, and the maximum number of subsets is set to 48. All the attention modules are 192 input channels. For the nuScenes variant, the architecture differs slightly. More details of architecture and hyper-parameter analysis are in Appendix.

Training and Inference. All the variants are trained by AdamW optimizer [27] on 8 NVIDIA A100 GPUs. We adopt the same learning rate scheme as [53]. All inference times are profiled on the same workstation (single NVIDIA A100 GPU). More implementation details are in Appendix.

4.2. 3D Object Detection

For performance benchmarking, we compare our model with previous methods on Waymo and NuScenes datasets.

Waymo. All the results are summarized in Table 1. As for the one-stage framework, our single frame DSVT-V model achieves 72.1 mAPH on L2 difficulties, which is +3.2 better than the previous best one-stage method [56] and even +1.3 higher than the best two-stage approaches [13]. Compared with our pillar variant, DSVT-V obtains significant improvements in small objects, (*i.e.*, pedestrian and cyclist), which demonstrates the effectiveness of our 3D pooling module for capturing detailed geometric information. Moreover, our model reaches 74.6, 75.6 in terms of L2 mAPH on 2, 4 frame settings, which outperforms the previous best one-stage multi-frame methods [56] by +1.8 and +2.4, respectively. These gains are more remarkable on L1 mAPH, *i.e.*, +3.7 and +4.3. Note that our model is not specifically designed for multi-frame detection (*e.g.*, multi-frame fusion), and only takes concatenated point clouds as input.

Random	Sparse	Regional	Rotate	L2 mAP	L2 mAPH
✓				63.66	60.20
	✓			64.15	60.89
		✓		64.27	61.03
			✓	**64.59**	**61.40**

Table 6. Effect of set partition method. All the experiments are based on the code base of SST [12].

We also evaluate our model against several competing two-stage approaches in Table 1. Our DSVT-TS voxel variant outperforms all the previous methods on all frame settings with a large margin, which gains +2.0, +3.2 and +2.0 of 3D mAPH(LEVEL_2) on 1-, 2- and 4-frame settings respectively. Note that we simply use raw-point based CT3D [33] as our second stage without some other elaborate designs, (*i.e.*, features interpolated from feature maps), leaving huge space for exploring better two-stage performance.

NuScenes. As shown in Table 2, our model outperforms all methods with remarkable gains. It achieves state-of-the-art performance, 72.7 and 68.4 in terms of *test* NDS and mAP, surpassing PillarNet [34] by +1.3 and +2.4 separately.

4.3. BEV map segmentation

To further demonstrate the generalizability of our approach, we evaluate our model on BEV Map Segmentation task of nuScenes dataset with only lidar input. The comparison results with state-of-the-art methods are reported in Table 3. Note that we follow the lidar branch of BEVFusion [25] and only switch the 3D backbone. All the baselines are reported by [25]. Thanks to the large receptive field of Transformer, our DSVT can further boost the performance with a remarkable gain (+3.0). With a deeper bev backbone, our model can be further improved by a huge gain (+19.4).

4.4. Ablation Studies

In this section, a set of ablative studies are conducted on Waymo *val* to investigate the key designs of our model. We follow OpenPCDet [42] to train all models on 20% training data with 30 epochs. Notably, all the variants are dim-128.

Comparison with sparse convolution backbone. Following the VoxelBackBone8x used in CenterPoint-Voxel [53], we extend it to a single-stride pillar-based residual backbone with a similar model size of our DSVT-P, named ResBackBone1x. For a fair comparison, we only switch the sparse 3D backbone while all other settings remain unchanged. Note that this variant is a very strong baseline, which is +4.9 higher than the original version [53] on L2 mAPH. As evidenced in 1^{st} and 2^{nd} rows of Table 10, thanks to the large receptive field of Transformer, even on such a strong baseline, our DSVT-P still brings +1.78 L2 mAPH gains over sparse convolution with a slightly larger latency. Due to the characteristic of friendly deployment, our model can achieve 2 × faster by TensorRT acceleration. More details of the architecture and analysis are in Appendix.

Window1	Window2	Latency	L2 mAP	L2 mAPH
(12, 12)	(12, 12)	6.04ms	70.93	68.35
(12, 12)	(24, 24)	5.79ms	**71.14**	**68.59**
(12, 12)	(36, 36)	**5.77ms**	70.80	68.18

Table 7. Effect of hybrid window partition. The reported latencies are the consumption of one DVST block.

Pooling Module	L_2 mAP	L_2 mAPH
Pillar	71.14	68.59
Linear	70.91	68.60
Max Pooling	71.21	68.91
Attention+Mask	71.24	68.94
Attention	**71.65**	**69.31**

Table 8. Effect of 3D sparse pooling module. The set size of pillar- and voxel-based variants are unified to 36 for a fair comparison.

Effect of dynamic sparse window attention. Table 5 ablates the modeling power and efficiency of our dynamic sparse window attention. All the experiments are conducted on the code base of SST [12], which applies a bucketing-based method [12, 41] to group the windows into several batches for efficient computation. Fully-padding means that simply converts all the non-empty sparse windows into dense ones. Notably, we eliminate the influence of window shift and only switch the attention strategy used in sparse backbone. Compared with the previous methods, the transformer backbone with our sparse attention strategy achieves 29 ms inference latency, which is nearly 2 $\times$ faster than the previous bucketing-based approach [12] while bringing +0.54 performance gain on L2 mAPH (*i.e.*, 60.86 $\rightarrow$ 61.40). That verifies our sparse attention strategy is much more efficient than standard attention in sparse data processing.

We further ablate the effectiveness of rotated set partitioning in Table 6. Several baselines are designed for comparison. Random: randomly sample non-empty voxels inside each window. Sparse: voxels of each subset are evenly distributed across the window region. Regional: a non-rotating variant of our partition approach. These methods can be implemented by controlling inner-window voxel ID (See §3.2). As shown in 1^{st} and 3^{rd} rows, the results of our regional partition are +0.83 L_2 mAPH higher than random sampling due to the better encoding of part-aware geometric information. Additional rotated configuration provides connections across subsets, further boosting the performance of 3D perception.

Table 5 and 7 show the results of different hybrid window shapes. Our method can greatly reduce the padding cost and deliver good performance-efficiency trade-off. Notably, we observe that attention is efficient actually, thus slight padding doesn't significantly slow down the inference. However, frequent function calls to attention will increase the latency due to additional overheads, (*e.g.*, memory access).

Effect of 3D pooling module. To demonstrate the effectiveness of our 3D pooling module, we refer several widely-used pooling strategies as baselines and only switch the pooling strategies while all other modules remain unchanged. Note that we unify the set size of both two variants (*i.e.*, DSVT-P and DSVT-V) into 36 for a fair comparison, which is different from the best setting reported in Table 1. See appendix for more baseline details and hyper-parameter analyses. As shown in Table 8, our pooling operation outperforms all other baselines. Interestingly, we find that adding key masks on empty voxels will harm the performance of our attention-style pooling operation, which suggests that the empty voxels also encode the object geometry information.

Models	Latency	L2 mAP	L2 mAPH
Centerpoint-Pillar	35ms	66.0	62.2
Centerpoint-Voxel	40ms	68.2	65.8
PV-RCNN++(center)	113ms	71.7	69.5
Ours(Pillar)	67ms	73.2	71.0
Ours(Voxel)	97ms	74.0	72.1
Ours(Pillar+TensorRT)	37ms	73.2	71.0

Table 9. The latency and performance on Waymo validation set.

4.5. Inference Speed

We present a comparison with other state-of-the-art methods on both inference speed and performance accuracy in Table 9. Our pillar variant significantly outperforms PV-RCNN++ [36] while achieving a much lower latency, *i.e.*, 67 ms vs 113 ms. After being deployed by NVIDIA TensorRT, our model can achieve a real-time running speed (27 *Hz*), which is almost as fast as the widely-used Centerpoint-Pillar [53] with much higher performance (+8.8 on L2 mAPH). Such running time is enough since a lidar typically operates at 10 *Hz* to 20 *Hz*. All the experiments are evaluated on the same workstation and environment.

5. Conclusion

In this paper, we propose DSVT, a deployment-friendly yet powerful transformer-only backbone for 3D perception. To efficiently handle sparse point clouds, we introduce dynamic sparse window attention, a novel attention strategy that partitions all the sparse voxels into a series of size-equivalent and window-bounded subsets, which can be processed in parallel without any customized CUDA operations. Thus, our proposed DSVT can be accelerated by well-optimized NVIDIA TensorRT, which achieves state-of-the-art performance on various 3D perception benchmarks with real-time running speed. We hope that our DVST can not only be a reliable point cloud processor for 3D perception in real-world applications but also provide a potential solution for efficiently handling sparse data in other tasks.

6. Acknowledgement

This work is supported by National Key R&D Program of China (2022ZD0114900) and National Science Foundation of China (NSFC62276005).

References

[1] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers. In *CVPR*, 2022. 5, 6, 13

[2] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. 2018. 1

[3] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *ICML*, 2021. 13

[4] 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. 2, 6

[5] Xuesong Chen, Shaoshuai Shi, Benjin Zhu, Ka Chun Cheung, Hang Xu, and Hongsheng Li. Mppnet: Multi-frame feature intertwining with proxy points for 3d temporal object detection. In *ECCV*, 2022. 6

[6] Bowen Cheng, Lu Sheng, Shaoshuai Shi, Ming Yang, and Dong Xu. Back-tracing representative points for voting-based 3d object detection in point clouds. In *CVPR*, 2021. 3

[7] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019. 1

[8] Spconv Contributors. Spconv: Spatially sparse convolution library. https://github.com/traveller59/spconv, 2022. 12

[9] Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wen gang Zhou, Yanyong Zhang, and Houqiang Li. Voxel r-cnn: Towards high performance voxel-based 3d object detection. In *AAAI*, 2021. 3

[10] Shaocong Dong, Lihe Ding, Haiyang Wang, Tingfa Xu, Xinli Xu, Jie Wang, Ziyang Bian, Ying Wang, and Jianan Li. MsSVT: Mixed-scale sparse voxel transformer for 3d object detection on point clouds. In *NeurIPS2022*, 2022. 3

[11] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *CVPR*, 2022. 3

[12] Lue Fan, Ziqi Pang, Tianyuan Zhang, Yu-Xiong Wang, Hang Zhao, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. Embracing single stride 3d object detector with sparse transformer. In *CVPR*, 2022. 2, 3, 6, 7, 8

[13] Lue Fan, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. Fully sparse 3d object detection. In *NeurIPS*, 2022. 2, 6, 7, 12

[14] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *CVPR*, 2018. 1

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

[16] Tianrui Guan, Jun Wang, Shiyi Lan, Rohan Chandra, Zuxuan Wu, Larry Davis, and Dinesh Manocha. M3detr: Multi-representation, multi-scale, mutual-relation 3d object detection with transformers. In *WACV*, 2022. 3

[17] Sylvain Gugger. The 1cycle policy. https://sgugger.github.io/the-1cycle-policy.html, 2018. 12

[18] Chenhang He, Ruihuang Li, Shuai Li, and Lei Zhang. Voxel set transformer: A set-to-set approach to 3d object detection from point clouds. In *CVPR*, 2022. 2, 3, 6

[19] Yihan Hu, Zhuangzhuang Ding, Runzhou Ge, Wenxin Shao, Li Huang, Kun Li, and Qiang Liu. Afdetv2: Rethinking the necessity of the second stage for object detection from point clouds. In *AAAI*, 2022. 6

[20] Yihan Hu, Zhuangzhuang Ding, Runzhou Ge, Wenxin Shao, Li Huang, Kun Li, and Qiang Liu. Afdetv2: Rethinking the necessity of the second stage for object detection from point clouds. *NCAI*, 2022. 5, 12

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 12

[22] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 5, 6

[23] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *CVPR*, 2019. 5

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. 2, 3, 5

[25] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han. Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation. In *ICRA*, 2023. 5, 6, 7, 13

[26] Ze Liu, Zheng Zhang, Yue Cao, Han Hu, and Xin Tong. Group-free 3d object detection via transformers. In *ICCV*, 2021. 3

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 7

[28] Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu. Voxel transformer for 3d object detection. In *ICCV*, 2021. 2, 3

[29] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3d object detection with pointformer. In *CVPR*, 2021. 2, 3

[30] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *ICCV*, 2019. 3

[31] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 1, 3

[32] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 1, 3

[33] Hualian Sheng, Sijia Cai, Yuan Liu, Bing Deng, Jianqiang Huang, Xian-Sheng Hua, and Min-Jian Zhao. Improving 3d object detection with channel-wise transformer. In *ICCV*, 2021. 5, 7, 13

[34] Guangsheng Shi, Ruifeng Li, and Chao Ma. Pillarnet: Real-time and high-performance pillar-based 3d object detection. In *ECCV*, 2022. 5, 6, 7, 12

[35] 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, 6

[36] Shaoshuai Shi, Li Jiang, Jiajun Deng, Zhe Wang, Chaoxu Guo, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn++: Point-voxel feature set abstraction with local vector representation for 3d object detection. *IJCV*, 2022. 3, 6, 8

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

[38] 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. *TPAMI*, 2020. 3, 6

[39] 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. 2, 6

[40] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay K. Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. *CVPR*, 2020. 1

[41] Pei Sun, Mingxing Tan, Weiyue Wang, Chenxi Liu, Fei Xia, Zhaoqi Leng, and Dragomir Anguelov. Swformer: Sparse window transformer for 3d object detection in point clouds. In *ECCV*, 2022. 2, 3, 6, 8

[42] OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet, 2020. 7, 12

[43] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*, 2019. 1

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 2, 3

[45] Dequan Wang, Coline Devin, Qi-Zhi Cai, Philipp Krähenbühl, and Trevor Darrell. Monocular plan view networks for autonomous driving. In *IROS*, 2019. 1

[46] Haiyang Wang, Lihe Ding, Shaocong Dong, Shaoshuai Shi, Aoxue Li, Jianan Li, Zhenguo Li, and Liwei Wang. CA-Group3d: Class-aware grouping for 3d object detection on point clouds. In *NeurIPS*, 2022. 3

[47] Haiyang Wang, Shaoshuai Shi, Ze Yang, Rongyao Fang, Qi Qian, Hongsheng Li, Bernt Schiele, and Liwei Wang. Rbgnet: Ray-based grouping for 3d object detection. In *CVPR*, 2022. 3

[48] Haiyang Wang, Wenguan Wang, Xizhou Zhu, Jifeng Dai, and Liwei Wang. Collaborative visual navigation. *arXiv preprint arXiv:2107.01151*, 2021. 1

[49] Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *ECCV*, 2020. 13

[50] Yue Wang, Alireza Fathi, Abhijit Kundu, David A Ross, Caroline Pantofaru, Tom Funkhouser, and Justin Solomon. Pillar-based object detection for autonomous driving. In *ECCV*, 2020. 3

[51] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 2018. 2, 3, 5, 6

[52] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3dssd: Point-based 3d single stage object detector. In *CVPR*, 2020. 3

[53] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *CVPR*, 2021. 3, 5, 6, 7, 8, 12

[54] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *ICCV*, 2021. 2, 3

[55] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018. 2, 5

[56] Zixiang Zhou, Xiangchen Zhao, Yu Wang, Panqu Wang, and Hassan Foroosh. Centerformer: Center-based transformer for 3d object detection. In *ECCV*, 2022. 6, 7

[57] 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. 6

[58] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 1

In the supplementary material, we first elaborate on the proof of non-overlapped set partition in §A, and then provide more implementation details of the network architecture, training schemes, and ablation baselines in §B. Finally, we present more ablative studies for hyper-parameter analyses in §C and visualization of quantitative results in §D. We also discuss the difference of Axis-attention in §E and limitation of DSVT in §F.

A. Proof of Non-overlap Set Partition

Definition A.1 (***Dynamic set partition***) N is the number of non-empty voxels for a specific window, and τ is the maximum number of sparse voxels allocated to each local set. The required number of sub-sets in this window is computed as follows,

$$S = \lfloor \frac{N}{\tau} \rfloor + \mathbb{I}[(N \,\%\, \tau) > 0]. \tag{8}$$

For the j-th set (denoted as $\mathcal{Q}_j = \{q_k^j\}_{k=0}^{\tau-1}$), the index of voxel can be computed:

$$q_k^j = \left\lfloor \widetilde{q}_k^j \right\rfloor = \left\lfloor \frac{(j \times \tau + k)}{S \times \tau} \times N \right\rfloor, \text{ for } k = 0, ..., \tau - 1. \tag{9}$$

The following theorems show that our algorithm formulation in the main paper satisfies all the necessary concepts of the proposed dynamic set partition. The case of $N = S \times \tau$ is trivial. Now suppose the case of $(S-1) \times \tau < N < S \times \tau$.

Theorem A.2. (***Non-overlap***): *For any two local sets,* $0 \leq i, j \leq S-1$, *then* $\mathcal{Q}_i \cap \mathcal{Q}_j = \emptyset$.

Proof. Obviously, this theorem can be converted to verify the non-overlap of two neighboring sets. Specifically, for any $1 \leq j \leq S-1$, $\mathcal{Q}_{j-1} \cap \mathcal{Q}_j = \emptyset$. We formulate q_0^j (the first voxel index in Q_j) as follows,

$$q_0^j = \left\lfloor \frac{j \times N}{S} \right\rfloor = \left\lfloor N_0^j + \frac{k_0^j}{S} \right\rfloor = N_0^j, \;\; 0 \leq k_0^j \leq S-1. \tag{10}$$

Then we can compute the last voxel index in Q_{j-1}:

$$\begin{aligned} q_{\tau-1}^{j-1} &= \left\lfloor \frac{(j-1) \times \tau + \tau - 1}{S \times \tau} \times N \right\rfloor \\ &= \left\lfloor \frac{j \times N}{S} - \frac{N}{S \times \tau} \right\rfloor = \left\lfloor N_0^j + \frac{k_0^j}{S} - \frac{N}{S \times \tau} \right\rfloor. \end{aligned} \tag{11}$$

Note that $(S-1) \times \tau < N < S \times \tau$, thus $\frac{S-1}{S} < \frac{N}{S \times \tau} < 1$ and $0 \leq \frac{k_0^j}{S} \leq \frac{S-1}{S}$. So we have

$$\left\lfloor N_0^j - 1 \right\rfloor < \left\lfloor N_0^j + \frac{k_0^j}{S} - \frac{N}{S \times \tau} \right\rfloor < \left\lfloor N_0^j \right\rfloor. \tag{12}$$

To this end, we can prove that $q_{\tau-1}^{j-1} = N_0^j - 1$. For any $1 \leq j \leq S-1$, $q_{\tau-1}^{j-1} \neq q_0^j$, thus $\mathcal{Q}_{j-1} \cap \mathcal{Q}_j = \emptyset$.

Theorem A.3. (***Completeness***): $\mathcal{Q}_0 \cup \mathcal{Q}_1 \cup ... \cup \mathcal{Q}_{S-1} = U$, *where* $U = \{0, 1, ..., N-1\}$.

Proof. If $u \in \{0, 1, ..., N-1\}$ but $u \notin U$, there must exist two continuous indexes q_{k-1}^j and q_k^j satisfying $q_{k-1}^j < u$ and $q_k^j \geq u + 1$. Thus we have:

$$\frac{(j \times \tau + k)}{S \times \tau} \times N - \frac{(j \times \tau + k - 1)}{S \times \tau} \times N > 1. \tag{13}$$

This yields a contradiction, because $N < S \times \tau$, and concludes the proof.

Theorem A.4. (***Equivalent***): *For any j-th subset,* $0 \leq j \leq S-1$, *we have* $\lfloor \frac{N}{S} \rfloor \leq |\mathcal{Q}_j| \leq \lfloor \frac{N}{S} \rfloor + 1$. $|\mathcal{Q}_j|$ *denotes the number of valid and unique voxels belonging to j-th set, which needs to be distinguished from* τ.

Proof. We reformulate N as:

$$N = \left\lfloor \frac{N}{S} \right\rfloor \times S + res = l \times S + res, \tag{14}$$

where $l = \lfloor \frac{N}{S} \rfloor$ and $0 \leq res \leq S-1$. In the case of $res = 0$, we have:

$$q_0^0 = 0, \; q_0^1 = l, \; ..., \; q_0^{S-1} = (S-1) \times l. \tag{15}$$

Following the proof of Theorem A.2, we can have:

$$q_{\tau-1}^0 = l - 1, \; q_{\tau-1}^1 = 2 \times l - 1, \; ..., \; q_{\tau-1}^{S-1} = S \times l - 1, \tag{16}$$

which indicates for any j, we obtain $|\mathcal{Q}_j| = l = \lfloor \frac{N}{S} \rfloor$. In the case of $res \neq 0$, we compute the difference between $\widetilde{q}_0^j$ and $\widetilde{q}_0^{j+1}$:

$$\begin{aligned} \widetilde{q}_0^{j+1} - \widetilde{q}_0^j &= \frac{(j+1) \times \tau}{S \times \tau} \times N - \frac{j \times \tau}{S \times \tau} \times N \\ &= \frac{N}{S} = l + \frac{res}{S}. \end{aligned} \tag{17}$$

By Eq. (10) and Eq. (17), we have

$$\begin{aligned} q_0^{j+1} &= \left\lfloor \frac{(j+1) \times N}{S} \right\rfloor = \left\lfloor \frac{j \times N}{S} + \frac{N}{S} \right\rfloor \\ &= \left\lfloor N_0^j + \frac{k_0^j}{S} + l + \frac{res}{S} \right\rfloor = N_0^j + l + \left\lfloor \frac{res + k_0^j}{S} \right\rfloor. \end{aligned} \tag{18}$$

Note that $0 \leq k_0^j \leq S-1$ and $0 < res \leq S-1$, therefore we can easily obtain:

$$N_0^j + l \leq q_0^{j+1} \leq N_0^j + l + 1. \tag{19}$$

Finally, following the proof of Theorem A.2, we have:

$$N_0^j + l - 1 \leq q_{\tau-1}^j \leq N_0^j + l, \tag{20}$$

which implies for any j, we have $l \leq |\mathcal{Q}_j| \leq l + 1$ and concludes the proof.

B. Implementation Details

In this section, we provide more implementation details about network architecture (§B.1), ablation baselines (§B.2) and training schemes (§B.3).

B.1. Network Architecture.

B.1.1 3D Perception on Waymo

As mentioned in the main paper, our detection approach follows the framework of CenterPoint-Pillar [53] and only appends our DSVT before BEV backbone while other components remained unchanged.

DSVT-P is a single-stride pillar-based sparse backbone, which adopts the pillar size of (0.32m, 0.32m, 6m) with four DSVT blocks. Each block is equipped with two DSVT layers with different set partitioning configurations, (*i.e*, X-Axis Partition and Y-Axis Partition). The DSVT layers contains a rotated set based Multi-Head Self-Attention (MHSA) module, followed by a 2-layer MLP with GELU nonlinearity in between. A LayerNorm (LN) layer is applied after each MHSA module and each MLP, and a residual connection is applied after each module. All the attention modules are equipped with 8 heads, 192 input channels, and 384 hidden channels. The hybrid window sizes are set to (12, 12, 1) and (24, 24, 1) by default, and the maximum number of voxels belonging to each set (τ) is 36, as introduced in main paper.

DSVT-V is a voxel-based variant of our proposed backbone, which follows the pillar-based framework and splits along the Z-Axis. The input voxel size is (0.32m, 0.32m, 0.1875m). Moreover, its backbone also has four stages with block numbers $\{1, 1, 1, 1\}$ and the number of voxels along the Z-Axis is reduced by our attention-style 3D pooling module with the stride $\{4, 4, 2\}$. The window sizes along the Z-Axis are $\{32, 8, 2, 1\}$, which covers all of the Z-Axis. Different from DSVT-P, to adapt more voxels, τ is set to 48.

B.1.2 3D Perception on nuScenes

3D object detection and BEV Map Segmentation both utilizes DSVT-P in nuScenes benchmark. We set window size and set the maximum number of tokens assigned to each set (τ) to (30, 30, 1) and 90, respectively. The attention modules in use were equipped with 8 heads, 128 input channels, and 256 hidden channels.

B.2. Ablation Baselines.

B.2.1 ResBackbone1x

ResBackbone1x is built upon sparse convolution (Spconv 2.0) [8], a widely used auto-differentiation library for sparse tensors. This baseline adopts the same network designs (*i.e.*, depth, width, and kernel size) as VoxelResBackBone8x implemented by OpenPCDet [42] except for replacing all the downsampling SparseConv blocks with conventional SubMConv to hold the single stride architecture. The input voxel size is set to (0.32m, 0.32m, 6m), which is the same as our DSVT pillar version. For a fair comparison, this variant only substitutes the DSVT sparse backbone with ResBackbone1x while other settings remained unchanged, (*e.g.*, detection head, loss functions, and post-processing). As shown in Table 10, thanks to the single stride design, this baseline is very strong, which is +4.89 better than the original CenterPoint-Voxel(8x) [53] and +2.80 higher than its residual modification version on L2 mAPH. Even on such a strong baseline, our DSVT performs +1.78 better, which demonstrates its powerful modeling ability.

Backbone	#param.	LEVEL_2 (3D) mAP	LEVEL_2 (3D) mAPH
VoxelBackBone8x†	58M	64.51	61.92
VoxelResBackBone8x†	80M	66.47	64.01
ResBackbone1x	88M	69.61	66.81
DSVT(Pillar, dim128)	71M	**71.14**	**68.59**

Table 10. Comparison with sparse convolution. † denotes the results implemented by OpenPCDet [42]. All models are trained on 20% Waymo data with 30 epochs.

B.2.2 3D Pooling

Linear. As for a specific downsampling sparse region, we first convert it into dense and flatten it to a vector with fixed length. Then a one-layer MLP is applied to project it. Finally, a layer normalization is adopted after MLP module.

Max Pooling. Similar to the linear variant, after being converted into a dense format, the native max-pooling operation is applied on voxel dimension for processing downsampling.

Attention+Mask. This variant follows the same design as our attention-style 3D pooling module except for adding key padding masks of the empty space in the pooling region.

B.3. Training and Inference Schemes.

B.3.1 Waymo

One Stage Detection. As mentioned in the main paper, we follow the same training schemes as [53] to optimize the model using Adam [21] optimizer with weight decay 0.05, one-cycle learning rate policy [17], and max learning rate 3e-3. All the models are trained with batch size 24 for 24 epochs on 8 NVIDIA A100 GPUs. During inference, following [20, 34], we use class-specific NMS with the IoU threshold of 0.7, 0.6 and 0.55 for vehicle, pedestrian and cyclist, respectively. Besides, we also use the ground-truth copy-paste data augmentation during training and disable this data augmentation in the last one epoch following [13] (*e.g.*, using the fade strategy).

DSVT-P	DSVT-V	Size	LEVEL_2 (3D) mAP	mAPH
✓		24	70.71	68.14
✓		36	**71.14**	**68.59**
✓		48	70.95	68.43
	✓	36	71.65	69.31
	✓	48	**72.01**	**69.67**
	✓	60	71.90	69.60

Table 11. Effect of set size.

Two Stage Detection. The two-stage of our DSVT is built upon CT3D [33] and trained separately. We fix the 1^{st}-stage model and finetune the 2^{nd}-stage refinement module for 12 epochs with the same training schedule.

B.3.2 NuScenes

3D Object Detection. We follow the same training scheme adopted in Transfusion-L [1]. All the models are trained by AdamW optimizer with weight decay 0.05, one-cycle learning rate policy, max learning rate 5e-3, and batch size 32 for 20 epochs.We adopt the same fade strategy in [1] in last 5 epochs.

BEV Map Segmentation. We also adopt the same training strategy as BEVFusion [25], including training epoch, learning rate and hyper-parameter of optimizer.

C. Hyper-parameter Analyses

Our DSVT also works well in a wide range of hyper-parameters, such as the set size and network depth. All the experiments are trained on 20% Waymo training data with 30 epochs.

Set Size. Table 11 shows the performance of our approach with different set sizes. With the increase of the set size (from 24 to 36 in DSVT-P, 36 to 48 in DSVT-V), the performance gradually improves. However, a very large set size will also slightly decrease the mAP/mAPH. We argue that our regional local set attention can better encode the part-aware geometric information, which enhances the performance of tiny objects. The large set coverage may involve lots of noise points. Thus, we set the set sizes to 36 and 48 for our DSVT-P and DSVT-V respectively.

Network Depth. DSVT is relatively shallow by design thanks to the large receptive fields of the transformer architecture. As shown in Table 12, we provide the results with a greater number of DSVT blocks for investigating the influence of network depth. We observe that the performance is gradually saturated with the increase of block number. A deeper network will decrease the running speed. Considering the trade-off between the computation cost and performance improvement, we choose 4 blocks as the default setting.

# of Blocks	LEVEL_2 (3D) mAP	mAPH
2	70.66	68.10
4	71.14	68.59
6	71.12	68.56
8	**71.24**	**68.68**

Table 12. Effect of network depth.

D. Qualitative Results

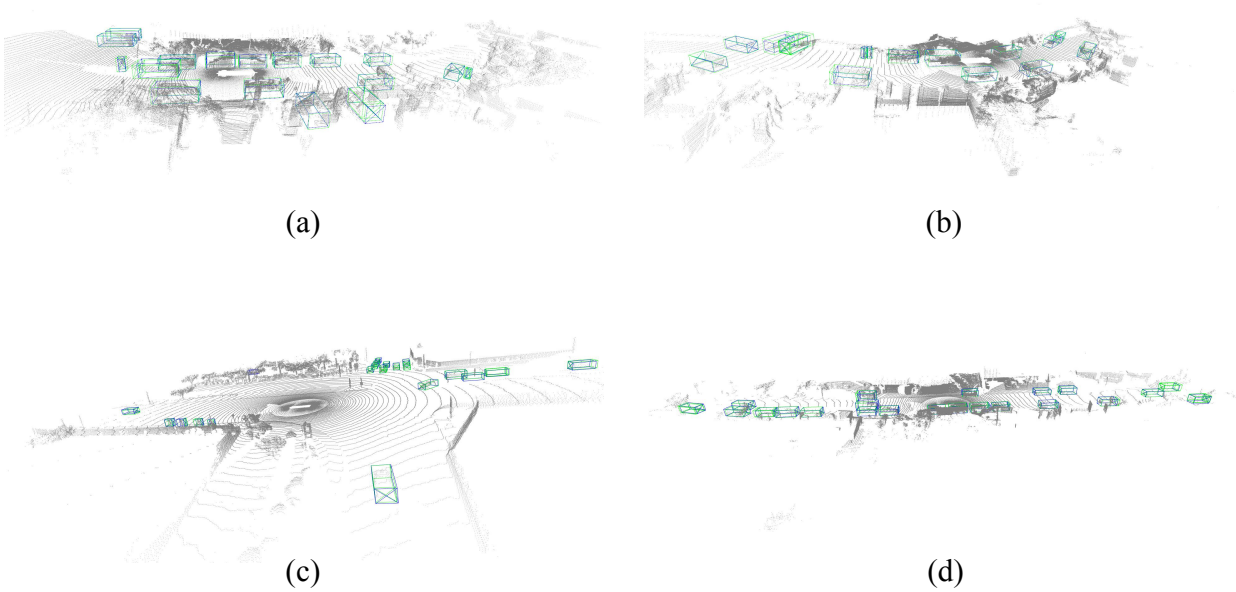
We visualize the qualitative results on Waymo Validation Set in Figure 4. Thanks to the large receptive field of Transformer and fine-grained geometric information provided by the attention-style 3D pooling module, our DSVT performs well on the large scenes and can locate 3D objects with sparse points accurately.

E. Compared to Axial-attention

Our method cannot be considered as an extension of Axial-attention [49]. DSVT is specifically designed for efficiently processing sparse data in parallel with dynamically assigned and size-equivalent local sets. The axis-based rotated partitioning is a replaceable strategy for intra-window fusion (please see Table 6 in the main paper for alternative strategies). In contrast, axial-attention [3, 49] aims to reduce the attention computation cost due to the dense data (*e.g.*, 2D image, or video) and enlarge receptive field by axis-based factorization.

F. Limitation

Although our method achieves promising performance and running speed on Waymo Open dataset, there are still some limitations. DSVT mainly focuses on point cloud processing for 3D object detection in outdoor autonomous driving scenarios, where objects (*i.e.*, car, pedestrian and cyclist) are distributed on a 2D ground plane. It is still an open problem to design more general-purposed backbones for the 3D community.



(a) (b) (c) (d)

Figure 4. Qualitative visualization on Waymo validation set. Blue boxes and green boxes are ground-truth and predictions, respectively.