A. Overview

In this document, we explain more details to the main paper and provide more ablation experiments on hyperparameters.

In Section B., we describe the implementation of sparse operation (Section 3.1). In Section C., we provide more details of our network architecture (Section 4.1). In Section D., we conduct ablation studies on some critical hyperparameters to prove the effectiveness of the model design. In Section E., we provide more visualization results on Waymo [2].

B. Sparse Operation

In this section, we describe in detail how to implement window-based attention on sparse voxels.

Hash Table Establishment Given the input sparse voxel $\boldsymbol{\mathcal{V}} = \{\boldsymbol{v}_i | \boldsymbol{v}_i = (\boldsymbol{x}_i, \boldsymbol{f}_i)\}_{i=1}^{|\boldsymbol{\mathcal{V}}|}$, we first build a hash table for convenient voxel searching. The key of hash table is the flattened voxel coordinate $bx_{max}y_{max}z_{max} + xy_{max}z_{max} + yz_{max} + z$ while the value stores the voxel features, where b, x, y, z denote batch index and voxel coordinates, and $x_{max}, y_{max}, z_{max}$ are the max range of voxel space. The hash function adopts the modulus strategy, and utilizes the linear probing method to find the empty address when hash collision occurs. In this way, we can only reserve the non-empty voxels, and establish the connection between voxel coordinates and their features. Once a voxel coordinate is given, we can rapidly check whether it is empty and extract the feature data according to the coordinate for non-empty location. It is worth noting that the hash table can be processed by CUDA in parallel so that the voxel searching process can be further accelerated.

Sparse Window Partition In Section 3.1.1, we partition the voxels into non-overlap 3D windows according to window size $\boldsymbol{s}_0$ and obtain their window centers $\{\boldsymbol{c}_i | \boldsymbol{c}_i \in \mathbb{Z}^3\}_{i=0}^{L}$ as:

$$\boldsymbol{c}_i = (\lfloor \boldsymbol{x}_i / \boldsymbol{s}_0 \rfloor + 0.5) \times \boldsymbol{s}_0, \tag{1}$$

Note that we further merge the repeated centers and generate a unique set of centers to save computation cost.

Voxel Gathering With ready of the window centers, we further search for non-empty voxels around the centers within the query or key windows. Benefit from the pre-built hash table, the voxel searching process can be converted to finding the existing hash keys via the hash function. In this way, we first get the non-empty voxel coordinates within all non-empty windows then leverage balanced multi-window sampling to obatin the final sampled voxel coordinates. Finally we can quickly gather the voxel features from the hash table with the sampled voxel coordinates. These voxel coordinates as well as their features are then fed to the attention mechanism as mentioned in the main paper (Section 3.1.2).

C. More Architectural Details

Following the classic 3D detection paradigm in OpenPCDet [3], our single-stage network consists of a VFE layer, a 3D backbone, a 2D backbone and a detection head. We also add a ROI head implemented by CT3D [1] on our single-stage architecture to build the two-stage network. In Section 4.1, we elaborate 3D backbone and detection head, and now we will detail the implementation of VFE layer and 2D backbone.

VFE. We apply the dynamic VFE [5] to convert N irregular points into L voxels with 128 channels, since it can reduce the information loss caused by voxelization. For the original point cloud in each voxel, we obtain the feature $\boldsymbol{P}_0 \in \mathbb{R}^{N \times (C_0 + 6)}$ by concatenating the relative coordinates offset to voxel centers and cluster centers with the original channels C_0. After that, the features are transformed into $\boldsymbol{F}_1 \in \mathbb{R}^{N \times 64}$ through linear projection as:

$$\boldsymbol{P}_1 = \textbf{MLP}(\boldsymbol{P}_0). \tag{2}$$

Then we compute the average of features in each voxel and cat it back to all points within the voxel as:

$$\boldsymbol{F}_1 = \textbf{Average}(\boldsymbol{P}_1), \tag{3}$$

Table 1: Ablations on grouping setup. s_1 represents the key window of $3 \times 3 \times 5$ and s_2 represents the key window of $7 \times 7 \times 7$.

Group	Veh / Ped / Cyc
$8s_1$	71.26/74.19/65.35
$6s_1 + 2s_2$	71.96/75.65/67.65
$4s_1 + 4s_2$	72.37/75.99/67.90
$2s_1 + 6s_2$	72.23/75.81/68.08
$8s_2$	72.06/74.40/66.73

Table 2: Ablations on window size setup. s_1 represents the size of query window and the inner key window. s_2 represents the size of query window and the outer key window. "3,3,5", "7,7,7" and other numbers represent the window size alone xyz axes.

Window Size	Veh / Ped / Cyc
$s_1 = 3, 3, 5, s_2 = 7, 7, 7$	72.37/75.99/67.90
$s_1 = 5, 5, 5, s_2 = 9, 9, 9$	72.04/75.37/67.75
$s_1 = 7, 7, 5, s_2 = 11, 11, 10$	70.48/73.84/65.71

$$\boldsymbol{P}_2 = \textbf{Concat}(\boldsymbol{P}_1, \boldsymbol{F}_1), \tag{4}$$

where $\boldsymbol{F}_1 \in \mathbb{R}^{L \times 64}$, and $\boldsymbol{P}_2 \in \mathbb{R}^{N \times 128}$. A new linear layer is used to transform the features into $\boldsymbol{P}_3 \in \mathbb{R}^{N \times 128}$ and finally we can obtain the voxelwise encoded features $\boldsymbol{F}_2 \in \mathbb{R}^{L \times 128}$ through another average pooling operation:

$$\boldsymbol{P}_3 = \textbf{MLP}(\boldsymbol{P}_2), \tag{5}$$

$$\boldsymbol{F}_2 = \textbf{Average}(\boldsymbol{P}_3). \tag{6}$$

2D Backbone. Following the implementation of SECOND [4] on OpenPCDet, the input 2D BEV features go through six 2D convolution layers at the original resolution and another six 2D convolution layers at 1/2 resolution, then are upsampled to the original resolution by the trans-convolution layers to obtain the final output.

D. Ablation experiments of hyperparameters

We conduct a series of ablation experiments for the important hyperparameters in MsSVT and report the L1 mAP for three categories. All ablation experiments used 20% waymo data and were trained for 12 epochs.

Effect of Window Grouping. In Table 1, we study the impact of different window head grouping settings. Note that the size of the query window is always $3, 3, 5$. Generally, our model is robust to different window head grouping settings, except for $8s_1$ and $8s_2$, our model degenerates into a single-scale model in this case, which causes the decline of the performance.

Effect of Window Size. We investigate the effects of window size in Table 2. It can be seen that too large window size will lead to performance degradation. This may be because the key voxels are too sparse in a large window. A possible solution is to increase the number of sampled key voxels,

Table 3: Ablations on number of blocks.

Blocks	Veh / Ped / Cyc	Mem(G)	Lat(ms)
2	71.79/74.66/66.36	8.3	91
3	72.06/75.24/66.41	10.2	103
4	72.37/75.99/67.90	12.2	121
5	72.69/75.74/66.88	14.1	145

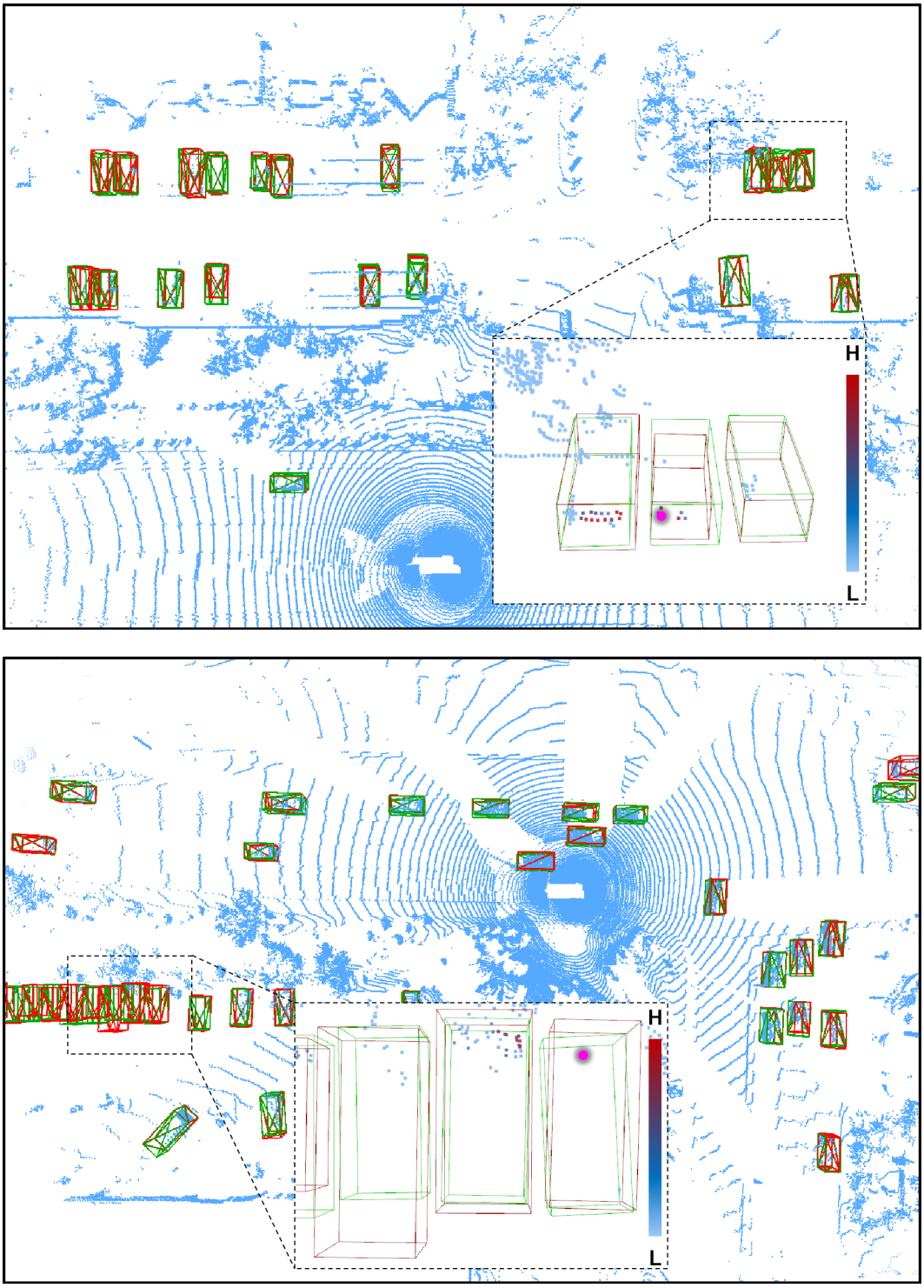

Figure 1: More visualization results on Waymo. The red and green boxes denote ground-truth and prediction, respectively. The pink dots represent query positions. The attention weight is reflected in the color of the point.

which however causes a huge computing cost and can not be adopted in implementation. Therefore, we select $\boldsymbol{s}_1 = 3, 3, 5, \boldsymbol{s}_2 = 7, 7, 7$ for our model.

Effect of Block Number. Table 3 shows the effect of different block numbers. Generally, our model is robust to the number of blocks. More blocks lead to the improvement of vehicle category, while other categories rise first then fall. Notably, more blocks can bring larger receptive field and help to extract higher-level semantic information, which is beneficial to the larger objects. To make a trade off between efficiency and performance, we finally use 4 blocks in our MsSVT by default.

E. More Visualization

We show more visualization results of our prediction bounding boxes and the attention weights in Fig. 1. It can be seen that most of the objects in Fig. 1 are accurately detected. Some "hard" bounding boxes are indicated with dotted rectangles, i.e., only 1~5 points are in the boxes. These boxes are still correctly regressed and even have a high confidence score of more than 0.5. By visualizing the attention weight, we find that in these hard cases, the few points left in the bounding boxes pay more attention to their surrounding objects to search for more information as the points within boxes themselves are too sparse to provide confident information, which demonstrates that MsSVT can gather a long range of contextual information to infer the location and size of the box when the fine-grained details are missing. A representative example is shown in Fig. 1: in a row of side-by-side vehicles, even if one of the vehicles is covered by very few points, its bounding box can still be predicted by understanding the scene semantics and analysing the locations and sizes of the surrounding vehicles.

References

[1] 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.

[2] 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.

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

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

[5] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *Conference on Robot Learning*, pages 923–932. PMLR, 2020.