# SWFormer: Sparse Window Transformer for 3D Object Detection in Point Clouds

Pei Sun(✉), Mingxing Tan, Weiyue Wang, Chenxi Liu, Fei Xia, Zhaoqi Leng, and Dragomir Anguelov

Waymo LLC, Palo Alto, USA
{peis,tanmingxing,weiyuewang,cxliu,feixia,lengzhaoqi,
dragomir}@waymo.com

**Abstract.** 3D object detection in point clouds is a core component for modern robotics and autonomous driving systems. A key challenge in 3D object detection comes from the inherent *sparse* nature of point occupancy within the 3D scene. In this paper, we propose Sparse Window Transformer (*SWFormer*), a scalable and accurate model for 3D object detection, which can take full advantage of the sparsity of point clouds. Built upon the idea of window-based Transformers, SWFormer converts 3D points into sparse voxels and windows, and then processes these variable-length sparse windows efficiently using a bucketing scheme. In addition to self-attention within each spatial window, our SWFormer also captures cross-window correlation with multi-scale feature fusion and window shifting operations. To further address the unique challenge of detecting 3D objects accurately from sparse features, we propose a new voxel diffusion technique. Experimental results on the Waymo Open Dataset show our SWFormer achieves state-of-the-art *73.36* L2 mAPH on vehicle and pedestrian for 3D object detection on the official test set, outperforming all previous single-stage and two-stage models, while being much more efficient.

## 1 Introduction

3D point cloud representation learning is critical for autonomous driving, especially for core tasks like 3D object detection. The challenges of learning from 3D point clouds mainly come from two aspects. The first aspect is that 3D points are sparsely distributed in the 3D space due to the nature of LiDAR sensors. This forces 3D models to be different from dense models in natural language processing (where words in a sentence are dense) or image understanding (where pixels in an image are dense). The second aspect is that both the number of points in a point cloud frame and the point cloud sensing region are increasing along with the improvement of the LiDAR sensor hardware. Some of the latest commercial LiDARs can sense up to 250 m [14] and 300 m [41] in all directions around the vehicle, leading to a large range of point clouds.

**Supplementary Information** The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-20080-9_25.


S. Avidan et al. (Eds.): ECCV 2022, LNCS 13670, pp. 426–442, 2022.
https://doi.org/10.1007/978-3-031-20080-9_25

To address these challenges, previous works have proposed many methods that can be roughly organized as five categories. **PointNet** [27,29,35] based method treats 3D point clouds as unordered sets and encodes them with MLPs and max pooling. Hierarchical structure is introduced to deal with the large input space and to better capture local information. These methods usually have inferior representation capacity compared with more recent methods. **PointPillars**-style methods [17] divide the space into grids of fixed sizes to convert the sparse 3D problem to a dense 2D problem. This method scales quadratically with the range, making it hard to scale with the advancement of LiDAR hardwares. **Sparse submanifold convolutions** [13,33,37] based method can handle the sparse input efficiently. Usually these methods use small $3 \times 3$ convolution kernels which cannot connect features that are sparsely disconnected without adding normal sparse convolution and striding. This weakness limits its representation capacity. Another weakness of this method is their need for heavily optimized custom ops to be efficient on the modern GPUs and incompatibility with matmul optimized accelerators such as TPUs. **Range image** is a compact representation of point cloud. Multi-view methods [2,37,40,47] run dense convolutions in this view to extract features and fuse with BEV features learned in the PointPillars-style to improve 3D representation learning. It is hard to regress 3D objects directly from the range image due to its lack of 3D information encoding in the dense 2D perspective convolutions. To tackle this weakness, graph-style kernels [4,11] replace convolutions to make use of the range information in range images to capture 3D information which greatly improves the accuracy but is still inferior to the state of the art. **Transformer** [38] is designed to process sequences of data. The challenge in applying it to a point cloud is to solve the quadratic complexity on the number of inputs. Recent methods tackle this problem by attending to neighboring points [26], neighboring voxels [22] or voxels in fixed windows [10]. A generic and efficient transformer-only model without limitations like limited receptive field, irregular memory access pattern, and lack of scalability is still to be designed.

In this paper, we adapt window-based Transformers to 3D point clouds. The Transformer [38] architecture has been hugely successful in modeling language sequences and image patches. In particular, on 2D images, Swin Transformer [21] proposed to partition images into windows and merge context information in a hierarchical manner. Our *Sparse Window Transformer (SWFormer)* builds upon similar ideas, but with several key adaptations for sparse windows. Our first adaption is to add a bucketing-based window partition for sparse windows. Although each window has the same spatial size, such as a $10 \times 10$ voxel grid, the number of non-empty voxels in each window can vary significantly, so we group these windows into buckets with different effective sequence lengths. Our second adaptation is to limit the expensive window shifting. Swin Transformer [21] uses window shifting once per Transformer layer to connect features between windows and increases receptive fields, but this shifting operation is expensive in the sparse world as it needs to re-order all the sparse features with gather operations. Moreover, it is extremely slow on matmul optimized accelerators such

as TPUs. To address this issue, SWFormer employs a new hierarchical backbone architecture, where each SWFormer block has many Transformer layers but only one shifting operation, as shown in Fig. 3. It relies on multi-scale features to achieve large receptive fields for context information, and a multi-scale fusion network to effectively combine these features. The model uses additional custom downsample and upsample algorithms to properly handle the sparse features during feature fusion.

Our innovation continues from the backbone into the 3D object detection head. Existing 3D object detection methods [4,12,17,22,33,37,40,43,47,48] can mostly be viewed as either anchor based methods with implicit or explicit anchors or DETR [3] based methods [24]. The detection performance is closely related with the distribution of the difference between anchor and groundtruth. Methods with inaccurate anchors [4,22] have poor performance in detecting large objects such as vehicles though they can have reasonable performance on pedestrians. One way to solve this problem is to have a two-stage model to refine the boxes [22,33] which greatly improves the detection accuracy. CenterNet-style detection methods [12,37,43] strive to define anchors in the center of the groundtruth boxes only which enforces distributions of closer to zero mean and smaller variance. However, when detecting objects directly from sparse features (e.g. features from PointNet, Submanifold convolutions, sparse Transformers), there are not necessarily features close to the object centers. To alleviate this issue, [37] applies normal sparse convolutions to insert points in the convolution output; [10] scatters the sparse features to a dense BEV grid and runs dense convolutions to expand features to missing positions. These methods are expensive. In this paper, we propose a voxel diffusion module to address this issue efficiently in a scalable way by segmenting and diffusing foreground voxels to their nearby regions as described in Sect. 3.4.

Extensive experiments are conducted on the challenging Waymo Open Dataset [36] to show state of the art results of SWFormer on 3D object detection. We summarize our contributions as follows:

- We propose a hierarchical Sparse Window Transformer (SWFormer) backbone for 3D representation learning. Its flexible receptive fields and multi-scale features make it suitable for different self-driving tasks like object detection and semantic segmentation.
- We propose a generic voxel diffusion module to address the unique challenge of anchor placement in 3D object detection from sparse features.
- We conduct extensive experiments on Waymo Open Dataset [36] to demonstrate the state of the art performance of our SWFormer model.

## 2 Related Work

### 2.1 3D Object Detection

As one of the most important tasks in autonomous driving, 3D object detection has been extensively studied in prior works. Early works like PointNet [27] and

PointNet++ [29] directly apply multilayer perceptions on individual points, but it is difficult to scale them to large point clouds with good accuracy. The current mainstream 3D object detectors often convert point clouds into bird eye view 3D [48] or 2D voxels [17] (2D voxels are also referred as pillars), where each voxel aggregates the information from points it contains. In this way, regular 2D or 3D convolutional neural networks can be applied to process these bird-eye-view representations. The pseudo image of voxels also makes it easier to reuse the rich research advancements in 2D object detection, such as two-stage or anchor-based detection heads [43]. The downside is that the pseudo image of voxels grows cubically/quadratically with the voxelization granularity and detection range, not to mention that many of the voxels are effectively empty. Therefore, another type of approach is to perform 3D object detection without voxelization. This includes methods that detect objects from the perspective view [4,11,23], or lookup nearest neighbors for each point [25]. However, the detection accuracy is typically inferior to the voxelization route.

To have the best of both worlds, recent approaches [33,37,42] start to explore multi-view approaches and make use of *sparse* convolutions on the *voxelized* point cloud. For example, the recent range sparse net (RSN [37]) adopts a two-step approach, where the first step performs class-specific segmentation on the range image view, and the second step applies sparse 3D convolutions on the voxel view for specific classes. However, submanifold sparse convolutions cannot connect features that are sparsely disconnected without adding normal sparse convolutions and striding, and they often require heavily optimized customized ops to be efficient on modern accelerators.

Our work aims to learn the 3D representations from sparse point clouds without using any dense or sparse convolutions. Instead, we resort to a hierarchical Transformer to achieve our goal.

### 2.2 Transformers

Transformers [38] have shown great success in natural language processing [7]. Recently, researchers have brought this architecture to computer vision [1,6, 30,39]. ViT [8] partitions images into patches, which greatly advanced the use of Transformers for image classification. Swin Transformer [21] further demonstrated better ways to fuse contextual information through window shifting and hierarchy, and also generalized to other tasks such as segmentation and detection.

Interestingly, Transformers are naturally suitable for sparse point clouds, because they can take any length of sequences as inputs and do not require dense 2D/3D image representations. Therefore, recent works have attempted to adopt Transformers for 3D representation learning, but they are primary developed for object scans and indoor applications [9,24,26,44]. Voxel Transformer [22] is the submanifold sparse convolution [13] counterpart in the Transformer world, by replacing the convolution kernel with attention. Its irregular memory access pattern is computationally inefficient, and its accuracy is worse than state of the art methods. Recently, SST [10] proposes a single-stride transformer for 3D object

detection and achieved impressive results on Waymo Open Datasets especially for pedestrian object detection. However, due to its single stride nature, SST has a limited receptive field and thus has difficulty dealing with large objects, making it ineffective in important tasks like large vehicle detection, large object segmentation (e.g. buildings), lane detection, and trajectory prediction. It needs to scatter features to a dense BEV grid to run several dense convolutions which limits its scalability. It is also computationally expensive as it needs to run many layers of transformers on the high resolution feature map which limits its applications in realtime systems.

Our work is inspired by window-based Transformers (e.g., SwinTransformer [21]) in the sense that we also adopt the hierarchical window-based Transformer backbone, but to address the unique challenges of 3D sparse point clouds, we propose several novel techniques such as the improved SWFormer blocks, multi-scale feature fusion, and voxel diffusion.

## 3 Sparse Window Transformer

### 3.1 Overall Architecture

SWFormer is a pure Transformer-based model without any convolutions. Figure 1 shows the overall network architecture: given a sequence of point cloud frames as inputs, each point is augmented with per-frame voxel features [17] and an auxiliary frame timestamp offset [37]. It uses dynamic voxelization [47] and a point net [17,27] based feature embedding net to get sparse voxel features. Note, our voxels are also referred as pillars in other works [17]. These sparse voxels are then processed by a hierarchical sparse window Transformer network described in Sect. 3.2. The resulting multi-scale features are then fused with a Transformer based feature fusion blocks. To address the unique challenge of detecting 3D boxes from sparse features, we first segment the foreground voxels and then apply a voxel diffusion module to expand foreground voxels to neighboring locations with pseudo voxels. In the end, we apply a center net [37,43,46] style detection head to regress 3D boxes.

### 3.2 Hierarchical Sparse Window Transformer Encoder

A key concept of our SWFormer is the *sparse window* in the birds eye view. After points are converted to a grid of 2D voxels on bird eye view, the voxel grid is further partitioned into a list of non-overlapping windows with fixed size $H \times W$ (e.g., $10 \times 10$), similar to Swin Transformer [21]; however, since points are often sparse, many voxels are empty with no valid points. Therefore, the number of non-empty voxels in each window may vary from 0 to $HW$. As we will explain later, all non-empty voxels within the same window will be flattened to a single variable-length sequence and fed into Transformer layers. In practice, these variable-length sequences prevent us from batch training, causing lower training efficiency. To solve this issue, we borrow a widely used ideas from natural language processing [7,38] and recent works [10], which group these sparse

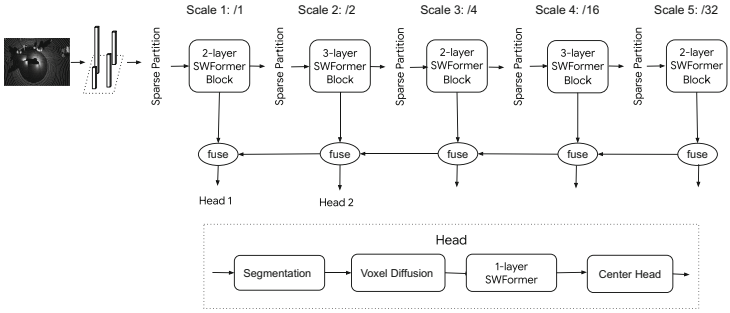


**Fig. 1.** Overview of SWFormer model architecture. Given a sparse point cloud, we first perform voxelization to generate a grid of 2D voxels. These voxels are then processed with a 5-scale sequence of hierarchical SWFormer blocks (Fig. 3), with strides $\{1, 2, 4, 16, 32\}$. The output features are combined with a multi-scale feature fusion network (Sect. 3.3). The fused features are fed to a head, which performs foreground segmentation and voxel diffusion (Sect. 3.4), and computes center net style classification and box regression loss (Sect. 3.5). Different object classes (e.g. vehicles and pedestrians) may use a separate head on different feature scales.

windows into different buckets based on their sequence lengths. Concretely, we divide sparse windows into at most $k$ buckets $\{B_0, B_1, ..., B_k\}$, where windows in $B_i$ are always padded to a maximum sequence length of $HW/2^i$. All padded tokens are masked in Transformer layers.

Based on the aforementioned sparse windows, our encoder adopts hierarchical Transformers to process the inputs and produce a list of multi-scale BEV features. As shown in Fig. 1, each scale starts with a sparse window partition layer followed by a multi-layer SWFormer block.

**Sparse Window Partition:** We divide the BEV voxels into non-overlapping windows with fixed size $H \times W$, which are then grouped into buckets $\{B_0, B_1, ..., B_k\}$. For each bucket $B_i$, we flatten all voxels within the same window into a sequence and zero-pad the sequence length to $HW/2^i$. These sequences are then batched and fed to the Transformer blocks, where the self-attention shares the keys and values for all query voxels coming from the same window [21]. Since SWFormer processes inputs in a hierarchical fashion with multiple feature scales, we need to apply strided window partitions at the beginning of each scale. The strided window partition is similar to traditional strided convolutions, except that it always picks the closest voxel to the center of the window with deterministic rules to break ties. Notably, no max or average pooling operations are applied because they are not friendly to sparse implementations. Figure 2 illustrates an example of a stride-4 window partition.

**Sparse Window Transformer Block:** Transformer [38] is inherently suitable for sparse point clouds, as it does not require the dense 2D/3D inputs as in convolutional networks; unfortunately, due to the quadratic complexity of self-attention with respect to the input sequence length, it is prohibitively expensive

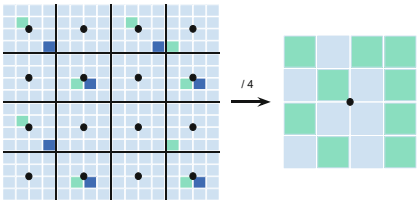


**Fig. 2.** Strided Sparse Window Partition. Left shows a grid of $16 \times 16$ BEV voxels, where grey voxels are empty and others are non-empty. Right shows the results of stride-4 window partition, leading to a grid of $4 \times 4$ voxels. For each striding window, it picks the nearest neighbor non-empty voxel feature (light green) from the center (black dot) with any deterministic rule to break ties; if all voxels are empty in the striding window, then the corresponding voxel after striding is also empty. Best viewed in color.

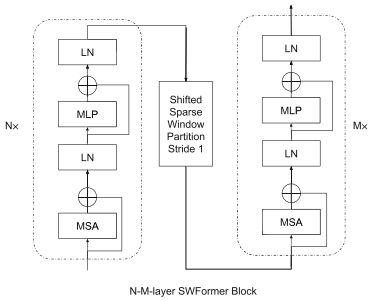


**Fig. 3.** Sparse Window Transformer Block. Given a sequence of sparse features, it first applies a multi-head self-attention (MSA) on all valid voxel within the same window, followed by a MLP and layer norm. After repeating the Transformer layer $N$ times, it performs a shifted sparse window partition to re-generate the sparse windows, and then process the shifted windows with another $M$ Transformer layers. If $N$ and $M$ are the same, we name it as N-layer SWFormer block for simplicity.

to feed the whole point cloud (with millions of points) or voxel features (with tens of thousands valid voxels) as a single input sequence to Transformer. In this paper, we adopt the idea of Swin Transformer [21]: the sparse BEV voxels are first partitioned into windows, and Transformer is applied to each window separately. To increase the receptive field and connect the features across windows, SwinTransformer uses a window shifting technique to re-partition the window for every layer of Transformer. However, as we are operating on sparse voxel features, such shift-window operation is memory-read/write intensive, especially for matrix-optimized accelerators like TPUs. To alleviate this problem, we propose to limit the shift-window operation to once per stride rather than per layer. Figure 3 shows the detailed architecture of a SWFormer block: it largely follows the same style of SwinTransformer to perform self-attention within a local window, except it only performs shift-window operation once in the middle.

Formally, our SWFormer block can be described as follows:

$$\mathbf{z}^0 = [\mathbf{x};\ \text{mask}_z] + \text{PE}_z \tag{1}$$
$$\hat{\mathbf{z}}^l = \text{LN}\left(\mathbf{z}^{l-1} + \text{MSA}(\mathbf{z}^{l-1})\right) \qquad l = 1...N$$
$$\mathbf{z}^l = \text{LN}\left(\hat{\mathbf{z}}^l + \text{MLP}(\hat{\mathbf{z}}^l)\right) \qquad l = 1...N$$
$$\mathbf{u}^0 = [\text{shift-window}(\mathbf{z}^N);\ \text{mask}_u] + \text{PE}_u$$
$$\hat{\mathbf{u}}^l = \text{LN}\left(\mathbf{u}^{l-1} + \text{MSA}(\mathbf{u}^{l-1})\right) \qquad l = 1...M$$
$$\mathbf{u}^l = \text{LN}\left(\hat{\mathbf{u}}^l + \text{MLP}(\hat{\mathbf{u}}^l)\right) \qquad l = 1...M \tag{2}$$

where $\mathbf{x}$ is the input features after sparse window partition, $\text{mask}_z$ is the mask for input padding, $\text{PE}_z$ is the positional encoding. The process contains two stages: (1) the first stage applies $N$ Transformer layers to $\mathbf{z}^0$ and output $\mathbf{z}^N$. Each Transformer layer consists of a standard multi-head self-attention (MSA) and multilayer perceptron (MLP), but slightly different from the standard version, here we adopt the post-norm scheme where layer norm (LN) is added after MSA and MLP. For simplicity, we use the standard sine/cosine absolute positional encoding in this paper. (2) The second stage first applies window-shift to $\mathbf{z}^N$, and adds the updated $\text{mask}_u$ and positional encoding $\text{PE}_u$ based on $\mathbf{z}^N$; afterwards, $M$ Transformer layers are added to process $\mathbf{u}^0$ and generate the final output $\mathbf{u}^M$. Notably, each SWFormer block has $N + M$ Transformer layers but only one window-shift operation.

By restricting window-shift operations, our SWFormer block is more efficient than the conventional Swin Transformer; however, it also limits the receptive field, since each Transformer layer is only applied to a small window. To address this challenge, SWFormer is designed as a hierarchical network with multiple scales, where the strides are gradually increased: for simplicity, this paper uses strides $\{1, 2, 4, 16, 32\}$ for the five scales. For each scale, we always keep the window size fixed (e.g., $10 \times 10$); however, as the later scales have larger strides, the same window in later scales will cover much larger area. As an example, for the last scale with stride 32, a $10 \times 10$ window would cover $320 \times 320$ area on the original BEV voxel grid, and a single window-shift would connect all features within an area as large as $480 \times 480$.

### 3.3 Multi Scale Feature Fusion

Inspired by feature pyramid network (FPN [19]), SWFormer adopts Transformer-based multi-scale feature network to effectively combine all features from the hierarchical Transformer encoder. Figure 4 shows the overall architecture of the feature network: given a list of encoder features $\{P_0, P_1, ..P_5\}$, it iteratively fuses $(P_{i+1}, P_i)$ from large-stride $P_5$ to small-stride $P_0$. Formally, our feature fusion process can be described as:

$$\hat{P}_5 = P_5 \tag{3}$$
$$\hat{P}_i = \text{SWFormer}(\text{Concat}(P_i, \text{Upsample}(\hat{P}_{i+1}))) \qquad i = 0, ..., 4 \tag{4}$$

Starting from the last feature map $P_5$, we first upsample it to have the same stride as $P_4$ such that they can be concatenated into a single feature map; afterwards, we simple apply a 1-layer SWFormer block to process the concatenated feature and generate the new $\hat{P}_4$. The process is iterated until all fused features $\{\hat{P}_0, ..., \hat{P}_5\}$ have been generated, which have the same strides as $\{P_0, ..., P_5\}$ features. The fused features are further used in voxel diffusion and box regression as described in the following sections.

One challenge in sparse upsamping is that one cannot naively duplicate the feature to all upsampled locations (like commonly done in dense upsampling), which will cause unnecessary excessive feature duplication and significantly reduce the sparsity. In this paper, we restrict features in $P_{i+1}$ to only duplicate to locations that have non-empty features in $P_i$, as shown in Fig. 4. In this way, we can ensure $\hat{P}_i$ has the same sparsity as $P_i$.

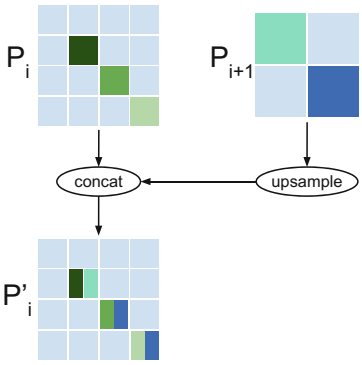


**Fig. 4.** Feature Fusion. Feature $P_{i+1}$ is upsampled and concatenated with $P_i$ to generate $P_i'$ and the final $P_i$. During upsampling, we only duplicate $P_{i+1}$ features to locations that are non-empty in $P_i$.

### 3.4 Voxel Diffusion

To detect 3D objects from sparse voxel features, a unique challenge is that there might be no valid voxel feature near object centers which are the best positions to place implicit [43] or explicit anchors [31]. Prior works have attempted to resolve this issue by: 1) second-stage box refinement [33], 2) sparse convolutions [37] or coordinate refinement [26] that can expand features to empty voxels close to the object centers, 3) scattering sparse voxel features to dense and applying dense convolutions [10]. In this paper, we propose a novel *voxel diffusion* module to effectively and efficiently address this challenge.

Voxel diffusion is based on two simple ideas: First, we segment all foreground voxels by jointly performing foreground/background segmentation, thus effectively filtering out the majority of background voxels. Second, we expand all foreground voxels by zero-initializing their features into neighboring locations with a simple $k \times k$ max pooling operations on the dense BEV grid, where $k$ is the detection head specific diffusion factor to control the magnitude of expansion. The diffused voxel features are further connected and processed with a few Transformer layers. Combining these two ideas, we can simultaneously keep voxel features sparse (by filtering out background voxels) and features filled (by voxel diffusion) for voxels closer to the object center. Figure 5 illustrates an example of voxel diffusion.

Our foreground segmentation is jointly trained with object detection. Specifically, for each voxel, we assign a binary groundtruth label: 0 (background, voxel does not overlap with any objects) and 1 (foreground, voxel overlaps with at

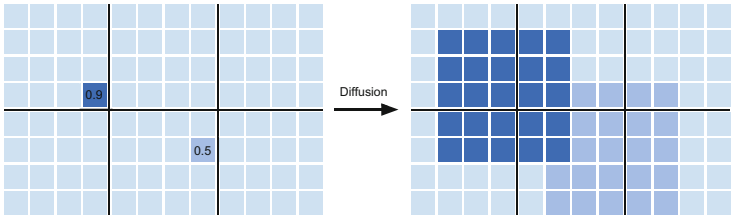


**Fig. 5.** Voxel Diffusion. After foreground segmentation, each voxel receives a segmentation score $s \in [0, 1]$. All voxels with scores greater than a threshold $\gamma = 0.05$ are scattered to a dense BEV grid, and then we apply a $k \times k$ max pooling on the dense BEV grid to expand valid voxel features to their neighboring locations where k is set to 5 in this example. (Left) before diffusion, there are only two foreground voxels with segmentation scores $\{0.5, 0.9\}$ greater than $\gamma$; (Right) after voxel diffusion, 47 voxels become valid. Best viewed in color.

least one object). The foreground segmentation is trained with a two-class focal loss [20] for each object class $c$:

$$L^c_{\text{seg}} = \frac{1}{N} \sum_i L_i \tag{5}$$

where $N$ is the total number of valid voxels and $L_i$ is the focal loss for voxel $i$. At inference time, we keep voxels as foreground if their foreground scores are greater than a threshold $\gamma$.

### 3.5 Box Regression

SWFormer follows [37] to use a modified CenterNet [12,37,43,46] head to regress boxes from voxel features. The heatmap loss is computed as a penalty-reduced focal loss [20,46] per object class.

$$\begin{aligned} L^c_{\text{hm}} = -\frac{1}{N} \sum_i \{ & (1 - \tilde{h}_i)^\alpha \log(\tilde{h}_i) I_{h_i > 1-\epsilon} + \\ & (1 - h_i)^\beta \tilde{h}_i^\alpha \log(1 - \tilde{h}_i) I_{h_i \leq 1-\epsilon} \}, \end{aligned} \tag{6}$$

where $\tilde{h}_i$ and $h_i$ are the predicted and ground truth heatmap values for object class $c$ respectively at voxel $i$. $N$ is the number of boxes in class $c$. We use $\epsilon = 1e-3$, $\alpha = 2$ and $\beta = 4$ in all experiments, following [18,37,46]. SWFormer parameterize 3D boxes as $\boldsymbol{b} = \{d_x, d_y, d_z, l, w, h, \theta\}$ where $d_x, d_y, d_z$ are the box center offsets relative to the voxel centers. $l, w, h, \theta$ are box length, width, height and box heading. We follow [37] to apply a bin loss [35] to regress heading $\theta$, smooth L1 to regress other box parameters, and an IoU loss [45] to improve overall box accuracy on the voxels with ground truth heatmap values above a

threshold $\delta_1$.

$$L^c_{\theta_i} = L_{bin}(\theta_i, \tilde{\theta}_i), \tag{7}$$

$$L^c_{\boldsymbol{b_i}\setminus\theta_i} = \text{SmoothL1}(\boldsymbol{b_i}\setminus\theta_i - \tilde{\boldsymbol{b_i}}\setminus\tilde{\theta}_i), \tag{8}$$

$$L^c_{\text{box}} = \frac{1}{N}\sum_i (L_{\theta_i} + L_{\boldsymbol{b_i}\setminus\theta_i} + L_{\text{iou}_i}) I_{h_i>\delta_1}, \tag{9}$$

where $\tilde{b}_i$, $b_i$ are the predicted and ground truth box parameters respectively, $\tilde{\theta}_i$, $\theta_i$ are the predicted and ground truth box heading respectively.

The net is trained end to end with the total loss defined as

$$L = \sum_c (\lambda_1 L^c_{\text{seg}} + \lambda_2 L^c_{\text{hm}} + L^c_{\text{box}}) \tag{10}$$

When decoding prediction boxes, we first filter voxels with heatmap less than a threshold $\delta_2$, then run max pool on the heatmap to select boxes corresponding to the local heatmap maximas without any non-maximum-suppression.

## 4 Experiments

We describe the SWFormer implementation details, and demonstrate its efficiency and accuracy in multiple experiments. Ablation studies are conducted to understand the importance of various design choices.

### 4.1 Waymo Open Dataset

Our experiments are primary based on the challenging Waymo Open Dataset (WOD) [36], which has been adopted in many recent state of the art 3D detection methods [10,28,33,37,43]. The dataset contains 1150 scenes, split into 798 training, 202 validation, and 150 test. Each scene has about 200 frames, where each frame captures the full 360° around the ego-vehicle. The dataset has one long range LiDAR with range capped at 75 m, four near range LiDARs and five cameras. SWFormer uses all five LiDARs in the experiments.

### 4.2 Implementation Details

We normalize intensity and elongation in the raw point cloud with the tanh function. The dynamic voxelization uses 0.32 m voxel size in $x$, $y$ and infinite size in $z$. During training, we ignore all ground truth boxes with fewer than five points inside. The voxel feature embedding net has two layers of MLPs with channel size of 128. All of the transformer layers have channel size of 128, 8 heads, and inner MLP ratio of 2. We also use stochastic depth [15] with survival probability 0.6. The segmentation cutoff $\gamma$ in Sect. 3.4 is set to 0.05. The heatmap threshold $\delta_1$, $\delta_2$ are set to 0.2, 0.1 respectively for both vehicle and pedestrian heads. For training efficiency, we cap the number of regression targets in each

frame by 1024 for vehicle and 800 for pedestrian sorted by ground truth heatmap values. $\lambda_1$, $\lambda_2$ are set to 200 and 10 in Eq. 10.

**Data Augmentation.** We have adopted the several popular 3D data augmentation techniques described in [5] during training: randomly rotating the world by yaws uniformly chosen from $[-\pi, \pi]$ with probability 0.74, randomly flipping the world along y-axis with probability 0.5, randomly scaling the world with scaling factor uniformly chosen within $[0.95, 1.05)$, randomly dropping points with probability of 0.05.

**Training and Inference.** The SWFormer models are trained end-to-end with 32 TPUv3 cores using the Adam optimizer [16] for a total number of 128 epochs with an initial learning rate set to 1e−3. We apply cosine learning rate decay and 8 epoch warmup with initial warmup learning rate set to 5e−4.

### 4.3 Main Results

We measured the detection results using the official WOD detection metrics: BEV and 3D average precision (AP), heading error weighted BEV, and 3D average precision (APH) for L1 (easy) and L2 (hard) difficulty levels [36]. The official metrics used to rank in the leaderboard uses IoU cutoff of 0.7 for vehicle, 0.5 for pedestrian. We report additional AP results at IoU of 0.8 for vehicle, 0.6 for pedestrian. Large vehicles that have max dimension greater than 7 m are also reported. Table 1 reports the main results on validation set, Table 2 reports additional results for high IoU and large vehicles on the validation set, and Table 3 shows the test set results by submitting our predictions to the official test server. Results from methods with test time augmentation or ensemble are not included.

As shown in Table 1, SWFormer achieves new state-of-the-art results for vehicle detection on the WOD *validation set*: it has 1.5 APH/L2 higher than the prior best single-stage model RSN [37]. SWFormer even outperforms the prior best performing two-stage method PVRCNN++ [34] by 0.42 APH/L2. Importantly, SWFormer performs very well at detecting large vehicles, 6.35 AP/L2 higher than the prior art of RSN [37] as shown in Table 2. SWFormer slightly outperforms the state of the art single stage method SST_3f [10] by 0.12 APH/L2. Notably, the single frame single stage SWFormer_1f also outperforms all prior single frame methods.

We have compiled the model with XLA [32] and ran inference for the 15th frame in scene 8907419590259234067_1960_000_1980_000 that has 68 vehicles and 69 pedestrians on a Nvidia T4 GPU with fused transformer kernels. The latency is 20 ms, more efficient than the popular realtime detector PointPillars [17] which takes about 100 ms on the same GPU with our own implementation.

Table 3 shows vehicle and pedestrian detection result comparison with published results on the WOD *test set*, which shows SWFormeroutperforms all previous single-stage or two-stage methods on the official ranking method mAPH/L2.

**Table 1.** WOD *validation set* results. † is from [37]. Top methods are highlighted. Top one-frame (cyan), single-stage (blue) are colored. TS: two-stage. BEV: BEV L1 AP.

| Method | TS | AP/APH Vehicle | | | AP/APH Pedestrian | | |
|---|---|---|---|---|---|---|---|
| | | 3D L1 | 3D L2 | BEV | 3D L1 | 3D L2 | BEV |
| PVRCNN++ [34] | ✓ | 79.3/78.8 | 70.6/70.2 | - | 81.8/76.3 | 73.2/68.0 | - |
| VoTr-TSD[22] | ✓ | 75.0/74.3 | 65.9/65.3 | - | - | - | - |
| SST_TS_3f [10] | ✓ | 78.7/78.2 | 70.0/69.6 | - | **83.8/80.1** | **75.9/72.4** | - |
| CenterPoint_TS [43] | ✓ | 76.6/76.1 | 68.9/68.4 | - | 79.0/73.4 | 71.0/65.8 | - |
| PointPillars [17] † | ✗ | 63.3/62.7 | 55.2/54.7 | 82.5 | 68.9/56.6 | 60.0/49.1 | 76.0 |
| MVF++_1f [28] | ✗ | 74.6/- | - | 87.6 | 78.0/- | - | 83.3 |
| RSN_1f [37] | ✗ | 75.1/74.6 | 66.0/65.5 | 88.5 | 77.8/72.7 | 68.3/63.7 | 83.4 |
| RSN_3f [37] | ✗ | 78.4/78.1 | 69.5/69.1 | 91.3 | 79.4/76.2 | 69.9/67.0 | 85.0 |
| SST_1f [10] | ✗ | 74.2/73.8 | 65.5/65.1 | - | 78.7/69.6 | 70.0/61.7 | - |
| SST_3f [10] | ✗ | 77.0/76.6 | 68.5/68.1 | - | 82.4/78.0 | 75.1/70.9 | |
| SWFormer_1f (Ours) | ✗ | 77.8/77.3 | 69.2/68.8 | 91.7 | 80.9/72.7 | 72.5/64.9 | 86.1 |
| SWFormer_3f (Ours) | ✗ | **79.4/78.9** | **71.1/70.6** | **92.6** | 82.9/79.0 | 74.8/71.1 | 87.5 |

**Table 2.** Additional WOD *validation set* results. Top methods are highlighted.

| Method | Vehicle L1 AP | | | Pedestrian L1 AP |
|---|---|---|---|---|
| | 3D IoU = 0.8 | BEV Large | 3D Large | 3D IoU = 0.6 |
| MVF++_1f [28] | 43.3 | - | - | 56.0 |
| RSN_3f [37] | 46.4 | 53.1 | 45.2 | - |
| SWFormer_3f (Ours) | **47.5** | **60.1** | **51.5** | **62.1** |

**Table 3.** WOD *test set* results. † is from [37]. Top methods are highlighted. mAPH/L2 is the official ranking metric on the WOD leaderboard. TS is short for two-stage.

| Method | TS | mAPH | Vehicle AP/APH 3D | | Pedestrian AP/APH 3D | |
|---|---|---|---|---|---|---|
| | | L2 | L1 | L2 | L1 | L2 |
| CenterPoint [43] | ✓ | 69.1 | 80.20/79.70 | 72.20/71.80 | 78.30/72.10 | 72.20/66.40 |
| SST_TS_3f [10] | ✓ | 72.94 | 80.99/80.62 | 73.08/72.74 | 83.05/79.38 | 76.65/73.14 |
| PVRCNN++ [34] | ✓ | 71.24 | 81.62/81.20 | 73.86/73.47 | 80.41/74.99 | 74.12/69.00 |
| P.Pillars [17] † | ✗ | 55.10 | 68.60/68.10 | 60.50/60.10 | 68.00/55.50 | 61.40/50.10 |
| RSN_3f [37] | ✗ | 69.70 | 80.70/80.30 | 71.90/71.60 | 78.90/75.60 | 70.70/67.80 |
| SWFormer_3f (Ours) | ✗ | **73.36** | 82.89/82.49 | 75.02/74.65 | 82.13/78.13 | 75.87/72.07 |

### 4.4 Ablation Study

**Voxel diffusion** is one of the primary contributions of this paper. We study its impacts by varying the diffusion window size $k$ introduced in Sect. 3.4. The result in Table 4 shows the significance of voxel diffusion. Disabling voxel diffusion (i.e. setting $k = 1$) results in 6.37 and 3.22 3D AP drop compared with $k = 9$ on

**Table 4.** Impact of Voxel Diffusion. Compared to the baseline (window size = 1), our voxel diffusion improves accuracy, especially with large diffusion window size.

| Diffusion Window Size | 1 | 3 | 5 | 9 |
|---|---|---|---|---|
| Vehicle 3D AP/L1 | 72.13 | 78.07 | 78.58 | 78.50 |
| Pedestrian 3D AP/L1 | 79.23 | 82.28 | 82.44 | 82.45 |
| Vehicle BEV AP/L1 | 82.42 | 91.19 | 92.09 | 92.03 |
| Pedestrian BEV AP/L1 | 83.65 | 87.01 | 87.15 | 87.47 |

**Table 5.** Impact of Multi-Scale and Window Shifting. Compared to single scale, multi-scale have much better accuracy. Window shifting is also important for performance.

| | Number of Scales | | | | Window Shift | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 5 | ✗ | ✓ |
| Vehicle 3D AP/L1 | 74.96 | 77.68 | 78.88 | 79.36 | 76.74 | 79.36 |
| Pedestrian 3D AP/L1 | 81.24 | 82.39 | 82.19 | 82.91 | 81.19 | 82.91 |
| Vehicle BEV AP/L1 | 89.55 | 91.83 | 92.23 | 92.60 | 90.74 | 92.60 |
| Pedestrian BEV AP/L1 | 86.48 | 87.30 | 87.13 | 87.54 | 86.46 | 87.54 |

vehicle and pedestrian detection respectively. Increasing $k$ can slightly improve the detection accuracy especially on vehicle.

**Multi-scale feature** improves the model accuracy as shown in Table 5 especially going from one scale to two scales. The impact is larger on vehicle detection (+2.72 3D AP) than pedestrian detection (+1.15 3D AP). The 3-scale model has pretty close accuracy as the full 5-scale model. In practice, we can trade-off between accuracy and latency by adjusting the number of scales. Note that some autonomous driving tasks such as lane detection, behavior prediction require larger receptive field. The success of training a deep five-scale SWFormer model shows its potential in those tasks.

**Window shifting** is introduced in SwinTransformer [21] to connect the features among windows. We have limited its usage to one per scale. What happens if we completely remove it? Table 5 shows clear accuracy drop especially on vehicles if the window-shift operations are removed from the SWFormer blocks. This meets our intuition that it is important to keep one window shift operation per scale to make sure every voxel gets the similar receptive field in all directions.

## 5 Conclusion

This paper presents *SWFormer*, a scalable and accurate sparse window transformer-only model, to effectively learn 3D point cloud representations for object detection. Built upon window-based Transformers, it addresses the unique

challenges brought by the sparse 3D point clouds, and proposes a bucketing-based multi-scale Transformer neural network. SWFormer takes full advantage of the sparsity of point clouds, and can effectively processes sparse windows of point clouds using pure Transformer layers without any convolutions. It also proposes a novel voxel diffusion module to further detect 3D objects from sparse features. Experiments show state-of-the-art results on the challenging Waymo Open Dataset.

## References


1. Bello, I., Zoph, B., Vaswani, A., Shlens, J., Le, Q.V.: Attention augmented convolutional networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 3286–3295 (2019)
2. Bewley, A., Sun, P., Mensink, T., Anguelov, D., Sminchisescu, C.: Range conditioned dilated convolutions for scale invariant 3D object detection. In: Conference on Robot Learning (2020)
3. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12346, pp. 213–229. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58452-8_13
4. Chai, Y., et al.: To the point: efficient 3D object detection in the range image with graph convolution kernels. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 16000–16009 (2021)
5. Cheng, S., et al.: Improving 3D object detection through progressive population based augmentation. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12366, pp. 279–294. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58589-1_17
6. Dai, Z., Liu, H., Le, Q., Tan, M.: CoatNet: marrying convolution and attention for all data sizes. In: Advances in Neural Information Processing Systems, vol. 34 (2021)
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
8. Dosovitskiy, A., et al.: An image is worth $16 \times 16$ words: transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
9. Engel, N., Belagiannis, V., Dietmayer, K.: Point transformer. IEEE Access **9**, 134826–134840 (2021)
10. Fan, L., et al.: Embracing single stride 3D object detector with sparse transformer. arXiv preprint arXiv:2112.06375 (2021)
11. Fan, L., Xiong, X., Wang, F., Wang, N., Zhang, Z.: RangeDet: in defense of range view for lidar-based 3D object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 2918–2927 (2021)
12. Ge, R., et al.: AFDet: anchor free one stage 3D object detection. arXiv preprint arXiv:2006.12671 (2020)
13. Graham, B., van der Maaten, L.: Submanifold sparse convolutional networks. arXiv preprint arXiv:1706.01307 (2017)
14. Guizilini, V., Ambrus, R., Pillai, S., Raventos, A., Gaidon, A.: 3D packing for self-supervised monocular depth estimation. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2020)

15. Huang, G., Sun, Yu., Liu, Z., Sedra, D., Weinberger, K.Q.: Deep networks with stochastic depth. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 646–661. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_39
16. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
17. Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: PointPillars: fast encoders for object detection from point clouds. In: CVPR (2019)
18. Law, H., Deng, J.: CornerNet: detecting objects as paired keypoints. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 734–750 (2018)
19. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2117–2125 (2017)
20. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2980–2988 (2017)
21. Liu, Z., et al.: Swin transformer: hierarchical vision transformer using shifted windows. In: CVPR (2021)
22. Mao, J., et al.: Voxel transformer for 3D object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 3164–3173 (2021)
23. Meyer, G.P., Laddha, A., Kee, E., Vallespi-Gonzalez, C., Wellington, C.K.: LaserNet: an efficient probabilistic 3D object detector for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12677–12686 (2019)
24. Misra, I., Girdhar, R., Joulin, A.: An end-to-end transformer model for 3D object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 2906–2917 (2021)
25. Ngiam, J., et al.: StarNet: targeted computation for object detection in point clouds. arXiv preprint arXiv:1908.11069 (2019)
26. Pan, X., Xia, Z., Song, S., Li, L.E., Huang, G.: 3D object detection with pointformer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7463–7472 (2021)
27. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: deep learning on point sets for 3D classification and segmentation. In: CVPR (2017)
28. Qi, C.R., et al.: Offboard 3D object detection from point cloud sequences. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 6134–6144 (2021)
29. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: deep hierarchical feature learning on point sets in a metric space. In: NeurIPS (2017)
30. Ramachandran, P., Parmar, N., Vaswani, A., Bello, I., Levskaya, A., Shlens, J.: Stand-alone self-attention in vision models. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
31. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1137–1149 (2016)
32. Sabne, A.: XLA: compiling machine learning for peak performance (2020)
33. Shi, S., et al.: PV-RCNN: point-voxel feature set abstraction for 3D object detection. In: CVPR (2020)
34. Shi, S., et al.: PV-RCNN++: point-voxel feature set abstraction with local vector representation for 3D object detection. arXiv preprint arXiv:2102.00463 (2021)

35. Shi, S., Wang, X., Li, H.: PointRCNN: 3D object proposal generation and detection from point cloud. In: CVPR (2019)
36. Sun, P., et al.: Scalability in perception for autonomous driving: Waymo open dataset. In: CVPR (2020)
37. Sun, P., et al.: RSN: range sparse net for efficient, accurate lidar 3d object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5725–5734 (2021)
38. Vaswani, A., et al.: Attention is all you need. In: NeurIPS (2017)
39. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7794–7803 (2018)
40. Wang, Y., et al.: Pillar-based object detection for autonomous driving. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12367, pp. 18–34. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58542-6_2
41. Waymo: Waymo's 5th generation driver. https://blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html
42. Yan, Y., Mao, Y., Li, B.: Second: sparsely embedded convolutional detection. Sensors (2018)
43. Yin, T., Zhou, X., Krahenbuhl, P.: Center-based 3D object detection and tracking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11784–11793 (2021)
44. Zhao, H., Jiang, L., Jia, J., Torr, P.H., Koltun, V.: Point transformer. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 16259–16268 (2021)
45. Zhou, D., et al.: IoU loss for 2D/3D object detection (2019)
46. Zhou, X., Wang, D., Krähenbühl, P.: Objects as points. arXiv preprint arXiv:1904.07850 (2019)
47. Zhou, Y., et al.: End-to-end multi-view fusion for 3D object detection in lidar point clouds. In: CORL (2019)
48. Zhou, Y., Tuzel, O.: VoxelNet: end-to-end learning for point cloud based 3d object detection. In: CVPR (2018)