ScatterFormer: Efficient Voxel Transformer with Scattered Linear Attention

Chenhang He[1], Ruihuang Li[1,2], Guowen Zhang[1], and Lei Zhang[1,2]

[1] The Hong Kong Polytechnic University, Hong Kong SAR, China
[2] OPPO Research, Shenzhen, China
`chenhang.he@polyu.edu.hk`
`csrhli@comp.polyu.edu.hk`
`guowen.zhang@connect.polyu.edu.hk`
`cslzhang@comp.polyu.edu.hk`

Abstract. Window-based transformers excel in large-scale point cloud understanding by capturing context-aware representations with affordable attention computation in a more localized manner. However, the sparse nature of point clouds leads to a significant variance in the number of voxels per window. Existing methods group the voxels in each window into fixed-length sequences through extensive sorting and padding operations, resulting in a non-negligible computational and memory overhead. In this paper, we introduce ScatterFormer, which to the best of our knowledge, is the first to directly apply attention to voxels across different windows as a single sequence. The key of ScatterFormer is a Scattered Linear Attention (SLA) module, which leverages the pre-computation of key-value pairs in linear attention to enable parallel computation on the variable-length voxel sequences divided by windows. Leveraging the hierarchical structure of GPUs and shared memory, we propose a chunkwise algorithm that reduces the SLA module's latency to less than 1 millisecond on moderate GPUs. Furthermore, we develop a cross-window interaction module that improves the locality and connectivity of voxel features across different windows, eliminating the need for extensive window shifting. Our proposed ScatterFormer demonstrates 73.8 mAP (L2) on the Waymo Open Dataset and 72.4 NDS on the NuScenes dataset, running at an outstanding detection rate of 23 FPS. The code is available at https://github.com/skyhehe123/ScatterFormer.

Keywords: 3D Object Detection · Voxel Transformer

1 Introduction

In the field of 3D object detection, the use of point clouds has become increasingly popular, especially for providing accurate and reliable perception results in autonomous systems. Unlike image data, point clouds obtained from LiDAR are often sparse and nonuniformly distributed, with varying density depending on their distances from the sensor. Earlier approaches utilize point-cloud operators
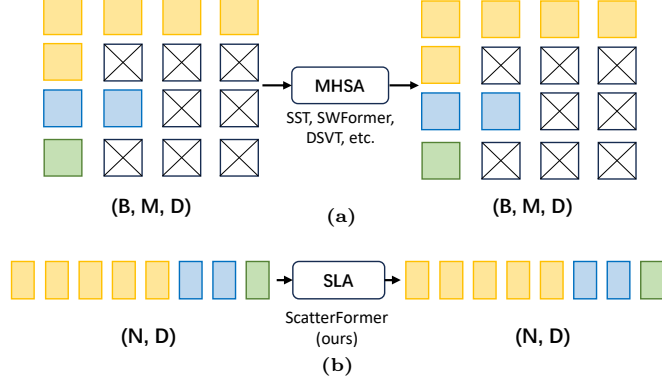
Fig. 1: Illustration of (a) group-based attention and (b) our scattered linear attention on variable-length sequences. The square in different colors represent voxels in different windows and the square with a cross represents the padding voxel.

like PointNet++ [31] for feature extraction in continuous space [37]. Some approaches [5, 9, 16, 18, 49, 50, 52, 60] have transformed point clouds into voxel grids, which are then efficiently processed by a sparse convolutional neural network (SpCNN) [49].

Recently, the success of vision transformers [7, 21] has motivated a number of attention-based methods to process indoor point clouds [12, 22, 24, 25, 27, 57]. Inspired by SwinTransformer [21], various studies [8, 14, 23, 47] have advanced the application of window-based Transformers in large-scale 3D detection tasks within outdoor environments, achieving outstanding performance and highlighting their potential as alternatives to SpCNN. However, due to the inherent sparsity of point clouds, the number of features grouped by windows can vary significantly, hampering the parallelism in attention computation. To resolve this issue, SST [8] and SWFormer [40] group the voxels within a window into different batches and manages the attention computation in a hybrid serial-parallel manner. More effectively, DSVT [47] alternatively sort the voxels within a window from different axes and partition them into sequences of fixed length, which allow parallel computation on sparse voxels. Albeit effective, these group-based methods, as shown in Figure 1(a), require extensive sorting and padding operations, incurring substantial memory and computational overhead.

In this paper, we delve into the window-based voxel transformer where the voxels grouped by windows form variable-length sequences $\{X_1 \in \mathbb{R}^{n_1 \times d}, X_2 \in \mathbb{R}^{n_2 \times d}, ..., X_k \in \mathbb{R}^{n_k \times d}\}$. Their corresponding self-attention matrices $\{A_1 \in \mathbb{R}^{n_1 \times n_1}, A_2 \in \mathbb{R}^{n_2 \times n_2}, ..., A_k \in \mathbb{R}^{n_k \times n_k}\}$ occupy irregular memory spaces, making it a challenge to perform attention in parallel for the voxels of the entire scene. Recent efforts on linear attention [1, 13, 17, 48] have emerged a promising alternative to traditional attention. By approximating softmax operation with the kernel function, $i.e.$, $\phi(Q) \cdot \phi(K)^{\mathsf{T}} \approx \mathrm{softmax}(QK^{\mathsf{T}})$, we can change the computation order from $(Q \cdot K^{\mathsf{T}}) \cdot V$ to $Q \cdot (K^{\mathsf{T}} \cdot V)$. This not only results in linear complexity but also yields a compressed hidden state $S \in \mathbb{R}^{d \times d}$ that is invariant to the sequence length. Another attractive property of linear attention is that it

can be converted into a "recurrent" form:

$$o_t = q_t S_t; \quad S_t = S_{t-1} + k_t^\mathsf{T} v_t \tag{1}$$

where the sequences can be divided into non-overlapping chunks, $\{q_j, k_j, v_j\}_{j=1:t}$. The output can be calculated by scanning over the sequence, based on an updated hidden state S_t. This enables the use of chunk-level computation to address the inability to parallelize at the sequence level due to varying lengths.

Inspired by this, we introduce the **Scattered Linear Attention** (SLA) module, which accommodates linear attention in a window-based voxel transformer. As shown in Figure 1(b), the SLA module treats the voxels of the entire scene into a single sequence and processes them directly without padding voxels. Using the recurrent form of linear attention, we develop an I/O-aware algorithm to further optimize matrix multiplication on voxel sequences. Specifically, we divide the voxel sequence into multiple chunks and loaded them into the shared memory (SRAM) of the GPU. The computation of the hidden state matrix in each window is then achieved through a series of chunk-wise matrix multiplications and cumulative sums. This optimized implementation significantly reduces I/O overhead and memory usage, resulting in extremely fast and memory-efficient attention computation.

In addition, current window-based transformer models use window shifting to propagate the information across the windows. After performing an instruction-level analysis of existing implementations, we observe that the voxel permutation operations caused by window shifting consume significant computational overhead. To address this issue, we propose a cross-window interaction (CWI) module, which is composed of depth-wise convolutions with small 2D kernels and lengthy 1D kernels that allow the voxel features in each window to fully interact with the features in other windows. As a result, the proposed CWI module can improve both the locality and connectivity of the voxel features while requiring minimal computational effort.

Building upon the SLA and CWI modules, we propose the **ScatterFormer**, an innovative voxel transformer for large-scale point cloud understanding. Our experiments show that ScatterFormer achieves linear complexity without compromising accuracy. It achieves superior results compared to the state-of-the-art model DSVT [47]. The latency of ScatterFormer is significantly lower than that of transformer-based detectors [8, 14, 47], which is comparable to that of sparse convolution-based detectors [52].

In conclusion, we delve into attention on voxels grouped by windows, highlighting the challenges in memory allocation and computation for variable-length sequences. Then, we introduce an SLA module, which can directly process the voxels grouped by windows by facilitating the linear attention formula. A chunk-wise matrix multiplication algorithm is proposed to further accelerate the attention computation in SLA. Finally, we present ScatterFormer, which has linear complexity and can efficiently process large-scale LiDAR scenes, achieving better accuracy with lower latency compared to existing transformer-based detectors.

2   Related Work

2.1   Point Cloud-based 3D Object Detection

There exist two primary point-cloud representations in 3D object detection, *i.e.*, point-based and voxel-based ones. In point-based methods [28, 29, 37, 39, 55], point clouds are first passed through a point-based backbone network [30], in which the points are gradually sampled and features are learned by point cloud operators. F-Pointnet [29] first employs PoinNet to detect 3D objects based on the frustums lifted by 2D proposals. PointRCNN [37] direct generates 3D proposals from point-based features with foreground segmentation. VoteNet [28] clusters objects from the surface points using a deep Hough voting method. Other point operators based on the point graph [39, 55] and the range view [10] have also been developed for point cloud processing.

Point-based methods are primarily limited by their inference efficiency and the absence of contextual features in a continuous space. On the other hand, voxel-based approaches [5, 16, 18, 49, 52, 58, 60] transform the entire point clouds into regular grids through voxelization, showcasing superior efficiency and context representation. VoxelNet [60] proposes a voxel feature encoding (VFE) module and combines it with 3D convolutions to extract voxel features in an end-to-end manner. SECOND [49] optimizes 3D convolution for sparse data, resulting in a significant reduction in both time and memory. PointPillars [18] demonstrates an efficient model by stacking voxels into vertical columns, subsequently processing them with naive 2D convolutions. There are also efforts [16, 35] that explore hybrid representations of point-based and voxel-based networks, demonstrating a better trade-off between speed and accuracy. However, a common limitation across these methods is their reliance on small convolution kernels with restricted receptive fields, making them less adept at capturing the global context for 3D object detection.

In this paper, we shed light on the voxel-based methods and introduce a Scattered Linear Attention module that enables dynamic modeling among voxels within each window with linear complexity, thereby efficiently extracting window-based contextual features.

2.2   Transformer on Point Cloud

Inspired by the significant success of self-attention in NLP [46] and CV [7, 45], Transformers have been adapted for 3D vision due to their ability to capture long-range dependencies. The Point Transformers [27, 57] employ attention to modulate point clouds for classification and segmentation tasks. PCT [12] presents an optimized offset attention module, which, when combined with the implicit Laplacian operator and normal estimation, becomes more adept at point-cloud processing. Some methods like [8, 22] opt for voxels or key points to optimize latency. 3DETR [25] proposes a query-based 3D object detection scheme. CT3D [33] enhances the region-based network using a channel-wise
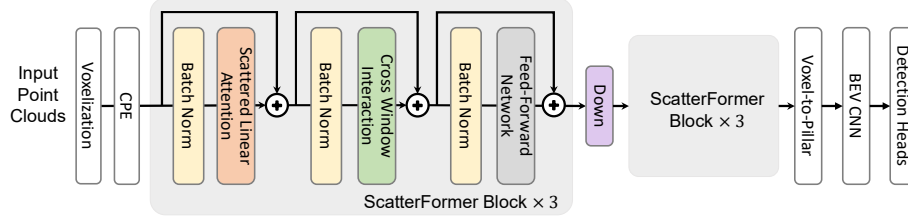
Fig. 2: The macro design of ScatterFormer. The backbone comprises a Conditional Position Encoding (CPE) and six transformer blocks. Each block is composed by a Scattered Linear Attention (SLA) module, a Cross-Window Interaction (CWI) module, and a Feed-Forward Network (FFN).

Transformer framework. To achieve context-rich representations, several studies [8, 14, 23, 24, 47] have integrated the attention module into point- or voxel-based encoders. For instance, VoTr [24] utilizes dilated attention for expanded receptive fields; VoxSet [14] applies set attention for extracting point-based features in set-to-set translation; SST [8] employs local attention with shifted windows; and OcTr [59] adopts an Octree-based attention for efficient hierarchical context learning. Nevertheless, how to efficiently leverage global context from attention remains a challenge due to the intrinsic sparsity of point clouds. DSVT [47] and FlatFormer [23] group the voxels within each window into a series of fixed-length voxel sets, thus extracting the features in a fully parallel manner. Recently, a group-free state-space model [54] was proposed to directly process voxels as a sequence. However, these approaches more or less lose the spatial proximity and incur extensive computational overhead in grouping and sorting the voxels.

3   Method

The overall architecture of our proposed ScatterFormer is depicted in Figure 2. It begins with the input point clouds, which are voxelized and transformed into high-dimensional embeddings using a VFE layer [60]. These embeddings are then processed through Conditional Positional Encoding (CPE) using a shallow convolutional network [6]. The encoded features enter the ScatterFormer backbone, consisting of six ScatterFormer blocks. Each block includes a Scattered Linear Attention (SLA) module, a Cross-Window Interaction (CWI) module, and a Feed-Forward Network (FFN), interspersed with Batch Normalization layers and skip connections. After three ScatterFormer blocks, the voxel features are downsampled via a sparse convolutional layer. The downsampled features are then converted into pillar features [47], generating compact BEV features for bounding-box prediction. ScatterFormer ScatterFormer stands out by not requiring voxel features to be organized into fixed-length sets [8, 23, 47], enabling flexible attention computation across windows. Additionally, the CWI module obviates the need for window shifting. These innovations allow ScatterFormer to avoid unnecessary memory allocation and permutation operations, achieving high efficiency comparable to CNN-based models.

3.1   Linear Attention

We begin by revisiting the self-attention introduced by [46]. Given an input matrix $x \in \mathbb{R}^{N \times d}$, where N denotes the number of tokens and d denotes the dimensionality. Self-attention first linearly projects the input to queries, keys, and values using weight matrices W_Q, W_K, and W_V, such that $Q = xW_Q, K = xW_K, V = xW_V$. Then it computes a Softmax-based attention map based on queries and keys $i.e.$, $A(Q, K) = \mathrm{Softmax}(QK^\mathsf{T})/\sqrt{d}$. The output of self-attention is defined as the weighted sum of N values with the weights corresponding to the attention map, $i.e.$, $O = A(Q, K)V$. This approach involves calculating the similarity for all query-key pairs, which yields a computational complexity of $O(N^2)$.

Linear attention [1, 17] offers a notable alternative to self-attention by significantly reducing its computational complexity from $O(N^2)$ to $O(N)$. This efficiency is achieved through the application of kernel functions on the query (Q) and key (K) matrices, which effectively approximates the original attention map without relying on Softmax, $i.e.$, $A(Q, K) = \phi(Q)\phi(K)^\mathsf{T}$. Taking advantage of the associative property of matrix multiplication, the computation shifts from $(\phi(Q)\phi(K)^\mathsf{T})V$ to $\phi(Q)(\phi(K)^\mathsf{T}V)$. This modification leads to a computational complexity proportional to the number of tokens, maintaining an order of $O(N)$.

3.2   Scattered Linear Attention (SLA)

Instead of organizing the voxels within a window into fixed-length subsets, we arrange the voxels of the entire scene into a single flattened matrix, as illustrated in Figure 3(a). These voxels are ordered based on their window coordinates, ensuring that voxels from the same window form a sub-matrix in contiguous memory. Initially, we use a shared projection layer to map all the voxels in the scene to query, key, and value representations, denoted Q, K, V. Subsequently, we perform attention computations on these submatrices separately, resulting in the scattered linear attention (SLA) formula as follows:

$$\mathrm{SLA}(Q, K, V) = \mathrm{Concat}[\mathrm{LA}(Q^j, K^j, V^j)]_{j=1:M}, \tag{2}$$

where M is the number of non-empty window in the current scene and LA is the linear attention formula used in [17]. Based on this formula, the output voxel features O^j in the jth window can be defined as:

$$\mathrm{LA} : O^j = \frac{\phi(Q) \sum_{i=1}^{m^j} \phi(K_i)^\top V_i}{\phi(Q) \sum_{i=1}^{m^j} \phi(K_i)^\top} \tag{3}$$

where m^j is the number of voxel in the window and $\phi(x)$ is the kernel function.

Hardware Efficient Implementation. It should be noted that the implementation of Equation 2 is not straightforward, as the sub-matrices have different numbers of rows. One naive implementation is to cache the "kv" matrix of each
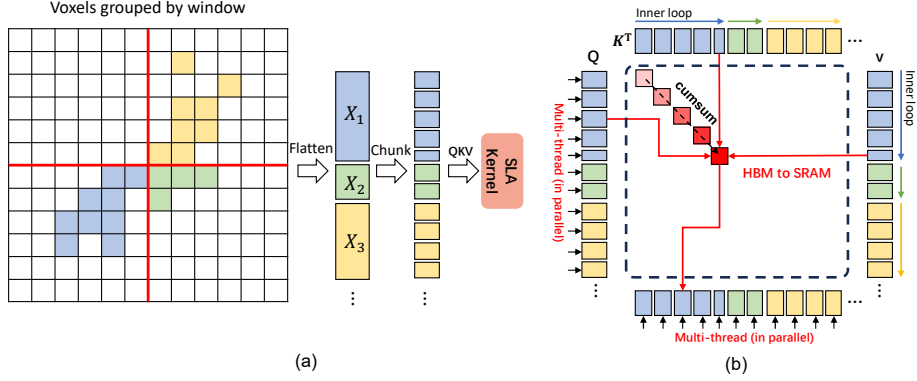
Fig. 3: (a) The voxel features are sorted by windows and flattened to a single matrix, which is further partitioned into multiple chunks of uniform length and sent to SLA kernel for attention computation. (b) In SLA, we first allocate individual threads to each window, each of them iterates over all corresponding key and value chunks, calculates, and accumulates their products to obtain the hidden state matrix of each window. Then, multiple threads are allocated to these query chunks $q_i \in Q$, calculating the chunk-wise output by multiplying q_i with the corresponding hidden state matrix.

voxel and apply $scatter^3$ operator to accumulate them within each window. However, this would consume a high memory overhead and IO latency. Considering that GPUs feature a memory hierarchy that includes larger, slower global GPU memory (high-bandwidth memory; HBM) and smaller, faster shared memory (SRAM), optimal utilization of SRAM to minimize HBM I/O costs can lead to significant speed-ups. Based on this, we partition the flattened Q, K, V into multiple chunks, load them from slow HBM to fast SRAM. Specifically, for each window, we allocate an single thread. Each thread iterates over all the key and value chunks, calculates, and accumulates their products to obtain the hidden state matrix of the current window. After calculating the hidden state matrix in window, we assign individual thread to each query chunk $q_i \in Q^j$ and calculate the chunk-wise output can then be following Equation 3. Figure 3(b) illustrates this chunk-wise matrix multiplication approach. As can be seen, it not only avoids outputting large matrices but also allows for more flexible processing of variable-length sequences.

In our implementation, we use Triton [44] to perform chunk-wise matrix multiplication. As illustrated in Figure 4, it outperforms the naive scattered-based operation, demonstrating improved speed and reduced memory usage when computing attention for extremely long sequences.

3.3   Cross Window Interaction

Traditional window-based transformers [8,40,47] utilize window shifting for inter-window connection. For point cloud sequences, this requires recalculating window coordinates and rearranging the voxels. In DSVT, the computation of voxel

---

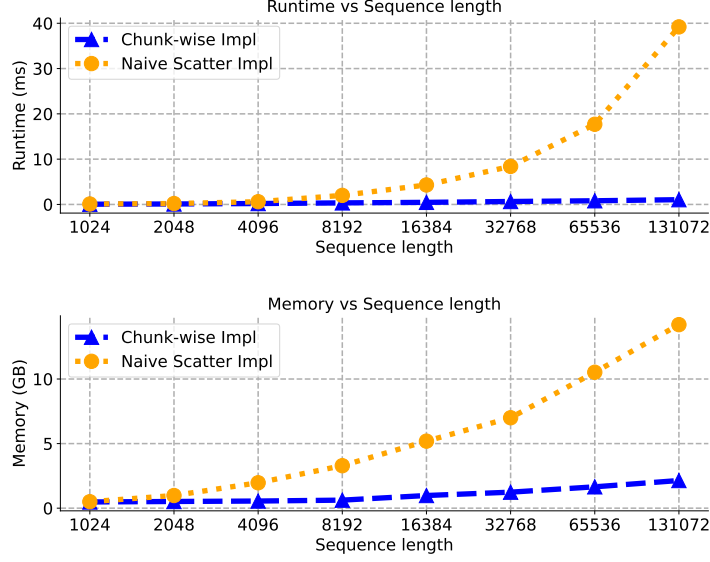^{3} https://github.com/rusty1s/pytorch_scatter

Fig. 4: Comparison between between scatter-based implementation (yellow) and our chunk-wise implementation (blue).

indices in subsets and the rearrangement of voxel matrices are particularly time consuming, accounting for **24%** of the total backbone latency, as shown in Figure 5. To alleviate this unnecessary computational overhead from window shifting, we introduce the Cross-Window Interaction (CWI) module, which employs convolutions with large kernel to blend voxel features across windows.

We adhere to the design principles of the Inception architecture [42], where the input features are divided along the channel dimension and processed through multiple branches using group-wise convolutions. Instead of employing a single large kernel for convolution, we adopt more efficient 1D kernels along different axes for feature aggregation. Specifically, we apply a $(S_h + 1) \times 1 \times 1$ convolution in one branch and a $1 \times (S_w + 1) \times 1$ convolution in another branch, where (S_w, S_h) denotes the window size. Additionally, we incorporate a standard $3 \times 3 \times 3$ convolution in one branch to enhance locality, while another branch performs an identity mapping. Given input X of channel $4c$, the output X' of the Cross Window Interaction module can be written as:

$$X' = \text{Concat}(X_k, X_w, X_h, X^{3c:4c}), \tag{4}$$

where

$$X_k = \text{DWConv}_{(S_h+1) \times 1 \times 1}(X^{:c}), \tag{5}$$

$$X_w = \text{DWConv}_{1 \times (S_w+1) \times 1}(X^{c:2c}), \tag{6}$$

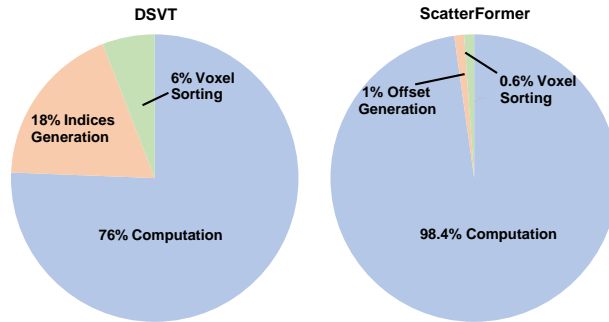$$X_h = \text{DWConv}_{3 \times 3 \times 3}(X^{2c:3c}). \tag{7}$$

Fig. 5: Runtime decomposition in DSVT [47] and ScatterFormer backbones. The offsets generation means computing the starting address of each chunk in the flattened matrix.

In our experiments, we found that this axis-decomposed large kernel design can achieve a better precision-latency trade-off. With these lengthy kernels, the voxel features can be efficiently blended among different windows.

3.4 Detection Head and Loss

To complement existing detection heads, ScatterFormer generates dense BEV feature maps from sparse voxel representations by placing them back to their spatial locations and filling the unoccupied positions with zeros. For the BEV network, we simply follow [53] by adopting a cascaded CNN with convolutional blocks at multiple levels of strides. In terms of detection head and loss function, we follow the design identical to that used in DSVT [47]. We train the detection model with heatmap estimation, bounding box regression, and incorporate an IoU loss for confidence calibration. On the nuScenes dataset, the detection head follows the architecture used in Transfusion [2].

4 Experiments

4.1 Datasets and Evaluation Metrics

Waymo Open Dataset (WOD). This dataset contains 230,000 annotated samples split into 160,000 for training, 40,000 for validation, and 30,000 for testing. It uses two metrics for 3D object detection: mean average precision (mAP) and mAP weighted by heading accuracy (mAPH), further categorized into Level 1 (L1) for objects detected by more than five LiDAR points and Level 2 (L2) for those detected with at least one point.

    NuScenes. This dataset comprises 40,000 annotated samples, with 28,000 for training, 6,000 for validation, and 6,000 for testing. On this dataset, the model performance is measured by mean average precision (mAP) across multiple distance thresholds (0.5, 1, 2, and 4 meters) and the nuScenes detection score (NDS), which combines mAP with a weighted sum of five additional metrics assessing true positive predictions in translation, scale, orientation, velocity, and attribute accuracy.
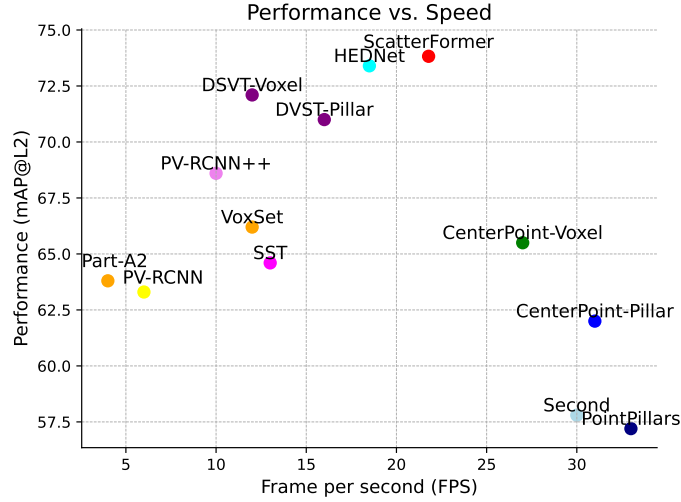
Fig. 6: The detection performance (mAPH/L2) vs. speed (FPS) of different methods on Waymo validation set. The speeds are measured on an NVIDIA A100 GPU.

4.2   Implementation Details

Our approach is implemented using the open-source framework OpenPCDet [43]. To construct ScatterFormer, we set the voxel size to (0.32m, 0.32m, 0.1875m) for the Waymo dataset and (0.3m, 0.3m, 8m) for the NuScenes dataset. The window sizes (S_w, S_h) for the two datasets are set to $(12, 12)$ and $(20, 20)$, respectively. We stack six building blocks for the backbone network. We configure our attention module to have 4 heads with a dimensionality of 128. ScatterFormer is trained for 24 epochs with a learning rate of 0.006 on Waymo Dataset and 20 epochs with a learning rate of 0.004 on NuScenes Dataset. In the last 4 epochs, we disabled the data augmentation strategy of ground-truth sampling. The model is trained on 8 RTX A6000 GPUs with a batch size of 32. Other settings for training and inference adhere strictly to DSVT [47].

4.3   Comparison with State-of-the-Arts

Results on Waymo Open Dataset (WOD). We conduct a detailed comparison of ScatterFormer against the published results on the validation set of the Waymo Open Dataset. In line with standard practices, we distinctly categorize and list the methods utilizing single-frame and multi-frame approaches. To ensure comprehensive coverage, we also include comparisons with methods incorporating long-term temporal modeling, such as those described in [4,15,61]. As demonstrated in Table 1, our single-stage model outperforms most two-stage methods and achieves better performance than state-of-the-art methods such as DSVT [47] and HEDNet [53]. Moreover,our model achieves 76.0 and 76.7 level 2 mAPH on 3 and 4 frame settings, respectively, outperforming previous multiframe methods by a margin of 1.1. Furthermore, as shown in Figure 6,

Table 1: Performance comparison on the validation set of Waymo Open Dataset. Symbol '*' denotes the methods with temporal modeling, and '-' means that the result is not available.

Method	Frames	ALL (3D mAPH)		Vehicle (AP/APH)		Pedestrian (AP/APH)		Cyclist (AP/APH)	
		L1	L2	L1	L2	L1	L2	L1	L2
SECOND [49]	1	63.05	57.23	72.27/71.69	63.85/63.33	68.70/58.18	60.72/51.31	60.62/59.28	58.34/57.05
PointPillar [18]	1	63.33	57.53	71.60/71.00	63.10/62.50	70.60/56.70	62.90/50.20	64.40/62.30	61.90/59.90
IA-SSD [56]	1	64.48	58.08	70.53/69.67	61.55/60.80	69.38/58.47	60.30/50.73	67.67/65.30	64.98/62.71
LiDAR R-CNN [20]	1	66.20	60.10	73.50/73.00	64.70/64.20	71.20/58.70	63.10/51.70	68.60/66.90	66.10/64.40
RSN [41]	1	-	-	75.10/74.60	66.00/65.50	77.80/72.70	68.30/63.70	-	-
Part-A2 [38]	1	70.25	63.84	77.05/76.51	68.47/67.97	75.24/66.87	66.18/58.62	68.60/67.36	66.13/64.93
Centerpoint [52]	1	-	65.50	-	-/66.20	-	-/62.60	-	-/67.60
VoTR [24]	1	-	-	74.95/74.25	65.91/65.29	-	-	-	-
VoxSeT [14]	1	72.24	66.22	74.50/74.03	65.99/65.56	80.03/72.42	72.45/65.39	71.56/70.29	68.95/67.73
SST-1f [8]	1	-	-	76.22/75.79	68.04/67.64	81.39/74.05	72.82/65.93	-	-
SWFormer-1f [40]	1	-	-	77.8/77.3	69.2/68.8	80.9/72.7	72.5/64.9	-	-
PV-RCNN [35]	1	69.63	63.33	77.51/76.89	68.98/68.41	75.01/65.65	66.04/57.61	67.81/66.35	65.39/63.98
PillarNet [34]	1	74.60	68.43	79.09/78.59	70.92/70.46	80.59/74.01	72.28/66.17	72.29/71.21	69.72/68.67
PV-RCNN++ [35]	1	75.21	68.61	79.10/78.63	70.34/69.91	80.62/74.62	71.86/66.30	73.49/72.38	70.70/69.62
FlatFormer [23]	1	-	67.2	-	69.0/68.6	-	71.5/65.3	-	68.6/67.5
PillarNext-1f [19]	1	75.74	69.74	78.40/77.90	70.27/69.81	82.53/77.14	74.90/69.80	73.21/72.20	70.58/69.62
VoxelNext [5]	1	76.3	70.1	78.2/77.7	69.9/69.4	81.5/76.3	73.5/68.6	76.1/74.9	73.3/72.2
FSD [9]	1	77.3	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
DSVT-1f [47]	1	78.2	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
HEDNet-1f [53]	1	79.5	73.4	**81.1/80.6**	**73.2/72.7**	84.4/**80.0**	76.8/**72.6**	78.7/77.7	75.8/74.9
ScatterFormer (ours)	1	**79.7**	**73.8**	81.0/80.5	73.1/72.7	**84.5**/79.9	**77.0**/72.6	**79.9/78.9**	**77.1/76.1**
SST-3f [8]	3	-	-	78.66/78.21	69.98/69.57	83.81/80.14	75.94/72.37	-	-
FlatFormer-3f [23]	3	-	72.0	-	71.4/71.0	-	74.5/71.3	-	74.7/73.7
SWFormer-3f [40]	3	-	-	79.4/78.9	71.1/70.6	82.9/79.0	74.8/71.1	-	-
PillarNext-3f [19]	3	80.0	74.5	80.6/80.1	72.9/72.4	85.0/82.1	78.0/75.2	78.9/77.9	76.7/75.7
DSVT-4f [47]	4	81.3	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
ScatterFormer-3f (ours)	3	81.7	76.0	82.0/81.4	75.0/74.1	85.7/83.2	79.0/76.1	80.6/79.5	78.1/77.7
ScatterFormer-4f (ours)	4	**81.9**	**76.7**	**82.4/81.9**	**75.2/74.7**	**86.1/83.4**	**79.3/76.6**	**81.2/80.4**	**78.9/78.1**
*3D-MAN [51]	16	-	-	74.53/74.03	67.61/67.14	71.7/67.7	62.6/59.0	-	-
*CenterFormer [61]	4	77.0	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
*CenterFormer [61]	8	77.3	73.7	78.8/78.3	74.3/73.8	82.1/79.3	77.8/75.0	75.2/74.4	73.2/72.3
*MPPNet [4]	4	79.83	74.22	81.54/81.06	74.07/73.61	84.56/81.94	77.20/74.67	77.15/76.50	75.01/74.38
*MPPNet [4]	16	80.40	74.85	82.74/82.28	75.41/74.96	84.69/82.25	77.43/75.06	77.28/76.66	75.13/74.52
*MSF [15]	4	80.20	74.62	81.36/80.87	73.81/73.35	85.05/82.10	77.92/75.11	78.40/77.61	76.17/75.40
*MSF [15]	8	80.65	75.46	82.83/82.01	75.76/75.31	85.24/82.21	78.32/75.61	78.52/77.74	76.32/75.47

our method significantly enhances the detection runtime. This improvement is attributed to our use of Linear Attention, which eliminates the need for extensive voxel sorting and padding operations. Furthermore, as presented in Table 2, ScatterFormer achieves the highest Level 1 and Level 2 mAPH scores in all categories of the Waymo test set, highlighting its superior detection quality and precision. Remarkably, our approach, as a one-shot method, delivers performance that is on par with methods that require temporal modeling. This finding underscores the critical importance of network design and architecture in achieving high-performance architecture.

Results on NuScenes. We compare ScatterFormer with the previous best performing methods on the nuScenes dataset. As shown in Table 3, ScatterFormer achieves the state-of-the-art performance in terms of *val* NDS (72.4) and mAP (68.3), surpassing HEDNet [53] by 1.0 and 1.6 respectively. Compared to TransFusion [2], ScatterFormer achieves a significant improvement of 2.8% in mAP, demonstrating that its backbone network is superior to the sparse convolutional network. ScatterFormer exhibits particularly strong performance on categories like Bus, Bike, Construction Vehicle and Trailer, suggesting its ability to effectively capture and encode detailed contextual features.

Table 2: Performance comparison on the test set of Waymo Open Dataset. '-' means that the result is not available.

Method	ALL (3D mAPH)		Vehicle (AP/APH)		Pedestrian (AP/APH)		Cyclist (AP/APH)	
	L1	L2	L1	L2	L1	L2	L1	L2
PointPillar [18]	-	-	68.10	60.10	68.00/55.50	61.40/50.10	-	-
StarNet [26]	-	-	61.00	54.50	67.80/59.90	61.10/54.00	-	-
M3DETR [11]	67.1	61.9	77.7/77.1	70.5/70.0	68.2/58.5	60.6/52.0	67.3/65.7	65.3/63.8
3D-MAN [51]	-	-	78.28	69.98	69.97/65.98	63.98/60.26	-	-
PV-RCNN++ [36]	75.7	70.2	81.6/81.2	73.9/73.5	80.4/75.0	74.1/69.0	71.9/70.8	69.3/68.2
CenterPoint [52]	77.2	71.9	81.1/80.6	73.4/73.0	80.5/77.3	74.6/71.5	74.6/73.7	72.2/71.3
RSN [41]	-	-	80.30	71.60	78.90/75.60	70.70/67.80	-	-
SST-3f [8]	78.3	72.8	81.0/80.6	73.1/72.7	83.3/79.7	76.9/73.5	75.7/74.6	73.2/72.2
HEDNet [53]	79.05	73.77	83.78/83.39	76.33/75.96	83.46/78.98	77.53/73.25	75.86/74.78	73.13/72.09
PillarNext-3f [19]	79.00	74.09	83.28/82.83	76.18/75.76	84.40/81.44	78.84/75.98	73.77/72.73	71.56/70.55
ScatterFormer-4f (ours)	**80.71**	**75.84**	**85.60/85.13**	**78.34/78.07**	**84.58/81.64**	**79.12/76.32**	**76.20/75.35**	**74.02/73.13**

Table 3: The performance on the validation set of NuScenes.

Method	NDS	mAP	Car	Truck	Bus	T.L.	C.V.	Ped.	M.T.	Bike	T.C.	B.R.
CenterPoint [52]	66.5	59.2	84.9	57.4	70.7	38.1	16.9	85.1	59.0	42.0	69.8	68.3
VoxelNeXt [5]	66.7	60.5	83.9	55.5	70.5	38.1	21.1	84.6	62.8	50.0	69.4	69.4
TransFusion-L [2]	70.1	65.5	86.9	60.8	73.1	43.4	25.2	87.5	72.9	57.3	77.2	70.3
PillarNext [19]	68.4	62.2	85.0	57.4	67.6	35.6	20.6	86.8	68.6	53.1	77.3	69.7
DSVT [47]	71.1	66.4	87.4	62.6	75.9	42.1	25.3	88.2	74.8	58.7	77.8	70.9
HEDNet [53]	71.4	66.7	87.7	60.6	77.8	**50.7**	28.9	87.1	74.3	56.8	76.3	66.9
ScatterFormer (ours)	**72.4**	**68.3**	**88.6**	**65.4**	**79.3**	45.4	**29.1**	**88.7**	**74.7**	**61.5**	**78.2**	**72.3**

4.4 Ablation Study

In this section, we conduct ablation experiments in ScatterFormer using 20% of the Waymo training and validation data. Each model was trained for 24 epochs and run on default parameters. Table 4 shows four different configurations of ScatterFormer: (a) removing the Scattered Linear Attention module (SLA), (b) removing the Cross Window Interaction (CWI) module, (c) replacing the CWI with Shifted Window (SW) approach, and (d) removing Conditional Position Encoding (CPE) module.

As can be seen in Table 4, configuration (a) results in ScatterFormer experiencing a decrease of (3.1%, 3.9%, and 3.2%) APH on three categories.This highlights the effectiveness of the SLA, as it successfully captures long-range contexts and facilitates dynamic feature extraction for object detection. In configuration (b), ScatterFormer, which lacks cross-window interaction capability, exhibits a performance drop of (1.9%, 1.0%, and 2.0%) APH across the three categories. While the Shifted Window approach in configuration (c) enhances performance, the improvement is not as substantial as that achieved by the proposed CWI module. The results in configuration (d) indicate that CPE contributes to a certain degree of performance enhancement.

Window Size. In Table 5, We tested the impact of different window sizes on the detection performance of various target types. It can be observed that ScatterFormer is not highly sensitive to window size. Slightly increasing the window size can enhance the detection performance for small targets, indicating that for objects like pedestrians and bicycles, where the point cloud is overly sparse, appropriate contextual information can enhance the recognition of these

Table 4: Ablation experiments on the validation set of Waymo Open Dataset. "SLA" and "CWI" refer to the proposed Scattered Linear Attention and Cross-Window Interaction modules, respectively. "CPE" refers to Conditional Position Embedding. AP and APH scores on LEVEL2 are reported.

	Ablation	Veh.	Ped.	Cyc.
-	baseline	71.6/71.1	76.0/70.9	74.2/73.1
(a)	w/o SLA module	68.5/68.0	73.3/67.0	71.0/69.9
(b)	w/o CWI module	69.6/69.2	74.0/69.9	72.1/71.1
(c)	CWI → SW	69.9/69.5	75.0/70.4	73.8/72.4
(d)	w/o CPE	70.2/69.8	73.3/67.2	72.4/70.5

Table 5: Comparison of model performance using different window sizes at various stages. The APH (L2) scores on different categories are reported.

Window Size	Region Size	Veh.	Ped.	Cyc.
10	3.20 m	70.2	69.2	71.0
12	3.84 m	71.1	70.9	73.1
14	4.48 m	71.3	69.9	72.5
16	5.12 m	71.0	69.2	72.3

objects. However, excessively large windows can degrade the performance. We speculate that this is because larger windows encompass more tokens, thereby mitigating the focusing power of linear attention.

Kernel Design in CWI. Table 6 presents the performance under various kernel configurations in the Cross-Window Interaction (CWI) module. Increasing the number of parameters in the kernel significantly enhances performance. For example, using a large 7×7 kernel brings notable performance improvements, but also introduces more latency. Additionally, larger kernel strides enhance the exchange rate of tokens across windows. For example, using dilated convolutions in CWI performs slightly better than using non-dilated convolutions. Instead of scaling up large, we decompose large kernel into more efficient 1D kernels, which achieves the optimal latency while keeping comparable performance. As the current Spconv library lacks support for depth-wise convolution, we have developed a modified version to address this limitation.[4]

4.5   Comparison with Different Linear Attentions

We further explored the performance of ScatterFormer by conducting experiments with several other linear attention mechanisms, including Gated Linear Attention [32], Cross-covariance Attention (XCA) [1], and Focused Linear Attention [13]. The results are summarized in Table 7. Our findings suggest that the window-based attention framework employed by ScatterFormer is not highly sensitive to the specific form of linear attention used. Notably, Efficient Attention [3] performs slightly worse than XCA [1] in the Vehicle class but better in the

---

[4] https://github.com/skyhehe123/spconv

Table 6: Performance with various kernel configurations in the Cross-Window Integration (CWI) module. Each configuration is specified by the notation "KxDy", where "Kx" refers to the kernel size, and "Dy" denotes the dilation rate. The APH (L2) scores on different categories are reported.

Kernel	Latency	Veh.	Ped.	Cyc.
$\text{Conv1D}_{K13} \times 2$	47ms	71.1	70.9	73.1
Conv2D_{K5D1}	49ms	69.5	69.0	71.1
Conv2D_{K5D2}	49ms	70.7	70.4	72.8
Conv2D_{K7D1}	65ms	71.5	71.0	72.5

Table 7: Performance with different linear attention designs. The APH (L2) scores on different categories are reported.

Linear Attention	Veh.	Ped.	Cyc.
Efficient Attn [3]	71.1	70.9	73.1
Gated Linear Attn [32]	69.9	69.0	72.5
Focused Linear Attn [13]	70.4	69.5	72.3
XCA [1]	71.5	70.4	72.8

Pedestrian and Cyclist classes. Focused Linear Attention introduces additional computational overhead, while Gated Linear Attention adds more learnable parameters to the model.

4.6   Limitations

ScatterFormer relies on our customized operators, which have not yet been implemented as plugins in TensorRT. Therefore, deploying ScatterFormer on in-vehicle devices will require additional engineering efforts. Despite this, ScatterFormer is more efficient than current models based on sparse convolution and traditional attention when used on consumer-grade GPU cards. Furthermore, ScatterFormer can be optimized by dynamically partitioning matrices according to different GPU architectures to leverage TensorCore for hardware acceleration.

5   Conclusion

We introduced ScatterFormer, an innovative architecture designed for 3D object detection using point clouds, specifically targeting the challenges associated with processing sparse and unevenly distributed data from LiDAR sensors. The cornerstone of our approach is the Scattered Linear Attention (SLA) module, which effectively addresses the limitations of conventional attention mechanisms in managing voxel features of varying lengths. SLA ingeniously combines linear attention with a chunk-wise matrix multiplication algorithm, tailored to meet the distinct requirements of processing voxels grouped by windows. By integrating SLA with a novel cross-window interaction (CWI) module, ScatterFormer achieves higher accuracy and lower latency, surpassing traditional transformer-based and sparse CNN-based detectors in extensive 3D detection tasks.

References

1. Ali, A., Touvron, H., Caron, M., Bojanowski, P., Douze, M., Joulin, A., Laptev, I., Neverova, N., Synnaeve, G., Verbeek, J., et al.: Xcit: Cross-covariance image transformers. Advances in neural information processing systems **34**, 20014–20027 (2021)
2. Bai, X., Hu, Z., Zhu, X., Huang, Q., Chen, Y., Fu, H., Tai, C.L.: Transfusion: Robust lidar-camera fusion for 3d object detection with transformers. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1090–1099 (2022)
3. Cai, H., Gan, C., Han, S.: Efficientvit: Enhanced linear attention for high-resolution low-computation visual recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2023)
4. Chen, X., Shi, S., Zhu, B., Cheung, K.C., Xu, H., Li, H.: Mppnet: Multi-frame feature intertwining with proxy points for 3d temporal object detection. In: European Conference on Computer Vision. pp. 680–697. Springer (2022)
5. Chen, Y., Liu, J., Zhang, X., Qi, X., Jia, J.: Voxelnext: Fully sparse voxelnet for 3d object detection and tracking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 21674–21683 (2023)
6. Chu, X., Tian, Z., Zhang, B., Wang, X., Shen, C.: Conditional positional encodings for vision transformers. ICLR (2023)
7. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. ICLR (2021)
8. Fan, L., Pang, Z., Zhang, T., Wang, Y.X., Zhao, H., Wang, F., Wang, N., Zhang, Z.: Embracing single stride 3d object detector with sparse transformer. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 8458–8468 (2022)
9. Fan, L., Wang, F., Wang, N., Zhang, Z.: Fully sparse 3d object detection. Advances in Neural Information Processing Systems **35**, 351–363 (2022)
10. 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 (ICCV). pp. 2918–2927 (October 2021)
11. Guan, T., Wang, J., Lan, S., Chandra, R., Wu, Z., Davis, L., Manocha, D.: M3detr: Multi-representation, multi-scale, mutual-relation 3d object detection with transformers. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision. pp. 772–782 (2022)
12. Guo, M.H., Cai, J.X., Liu, Z.N., Mu, T.J., Martin, R.R., Hu, S.M.: Pct: Point cloud transformer. Computational Visual Media **7**(2), 187–199 (Apr 2021). `https://doi.org/10.1007/s41095-021-0229-5`, `http://dx.doi.org/10.1007/s41095-021-0229-5`
13. Han, D., Pan, X., Han, Y., Song, S., Huang, G.: Flatten transformer: Vision transformer using focused linear attention. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5961–5971 (2023)
14. He, C., Li, R., Li, S., Zhang, L.: Voxel set transformer: A set-to-set approach to 3d object detection from point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8417–8427 (2022)
15. He, C., Li, R., Zhang, Y., Li, S., Zhang, L.: Msf: Motion-guided sequential fusion for efficient 3d object detection from point cloud sequences. In: Proceedings of the

IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5196–5205 (2023)

16. He, C., Zeng, H., Huang, J., Hua, X.S., Zhang, L.: Structure aware single-stage 3d object detection from point cloud. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11873–11882 (2020)

17. Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F.: Transformers are rnns: Fast autoregressive transformers with linear attention. In: International conference on machine learning. pp. 5156–5165. PMLR (2020)

18. Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: Pointpillars: Fast encoders for object detection from point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 12697–12705 (2019)

19. Li, J., Luo, C., Yang, X.: Pillarnext: Rethinking network designs for 3d object detection in lidar point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 17567–17576 (2023)

20. Li, Z., Wang, F., Wang, N.: Lidar r-cnn: An efficient and universal 3d object detector. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2021)

21. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 10012–10022 (October 2021)

22. Liu, Z., Zhang, Z., Cao, Y., Hu, H., Tong, X.: Group-free 3d object detection via transformers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 2949–2958 (October 2021)

23. Liu, Z., Yang, X., Tang, H., Yang, S., Han, S.: Flatformer: Flattened window attention for efficient point cloud transformer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1200–1211 (2023)

24. Mao, J., Xue, Y., Niu, M., Bai, H., Feng, J., Liang, X., Xu, H., Xu, C.: Voxel transformer for 3d object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 3164–3173 (October 2021)

25. 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 (ICCV). pp. 2906–2917 (October 2021)

26. Ngiam, J., Caine, B., Han, W., Yang, B., Chai, Y., Sun, P., Zhou, Y., Yi, X., Alsharif, O., Nguyen, P., et al.: Starnet: Targeted computation for object detection in point clouds. arXiv preprint arXiv:1908.11069 (2019)

27. 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 (CVPR). pp. 7463–7472 (June 2021)

28. Qi, C.R., Litany, O., He, K., Guibas, L.J.: Deep hough voting for 3d object detection in point clouds. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)

29. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum pointnets for 3d object detection from rgb-d data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 918–927 (2018)

30. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 652–660 (2017)

31. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in neural information processing systems. pp. 5099–5108 (2017)

32. Qin, Z., Li, D., Sun, W., Sun, W., Shen, X., Han, X., Wei, Y., Lv, B., Yuan, F., Luo, X., Qiao, Y., Zhong, Y.: Scaling transnormer to 175 billion parameters (2023)

33. Sheng, H., Cai, S., Liu, Y., Deng, B., Huang, J., Hua, X.S., Zhao, M.J.: Improving 3d object detection with channel-wise transformer. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 2743–2752 (October 2021)

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

35. Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., Li, H.: Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)

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

37. Shi, S., Wang, X., Li, H.: Pointrcnn: 3d object proposal generation and detection from point cloud. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–779 (2019)

38. Shi, S., Wang, Z., Wang, X., Li, H.: Part-aˆ 2 net: 3d part-aware and aggregation neural network for object detection from point cloud. arXiv preprint arXiv:1907.03670 (2019)

39. Shi, W., Rajkumar, R.: Point-gnn: Graph neural network for 3d object detection in a point cloud. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1711–1719 (2020)

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

41. Sun, P., Wang, W., Chai, Y., Elsayed, G., Bewley, A., Zhang, X., Sminchisescu, C., Anguelov, D.: 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)

42. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)

43. Team, O.D.: Openpcdet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet (2020)

44. Tillet, P., Kung, H.T., Cox, D.: Triton: An intermediate language and compiler for tiled neural network computations. In: Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. p. 10–19. MAPL 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3315508.3329973, https://doi.org/10.1145/3315508.3329973

45. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jegou, H.: Training data-efficient image transformers & distillation through attention. In: International Conference on Machine Learning. vol. 139, pp. 10347–10357 (July 2021)

46. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)

47. Wang, H., Shi, C., Shi, S., Lei, M., Wang, S., He, D., Schiele, B., Wang, L.: Dsvt: Dynamic sparse voxel transformer with rotated sets. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13520–13529 (2023)
48. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity (2020)
49. Yan, Y., Mao, Y., Li, B.: Second: Sparsely embedded convolutional detection. Sensors **18**(10), 3337 (2018)
50. Yang, B., Luo, W., Urtasun, R.: Pixor: Real-time 3d object detection from point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7652–7660 (2018)
51. Yang, Z., Zhou, Y., Chen, Z., Ngiam, J.: 3d-man: 3d multi-frame attention network for object detection. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1863–1872 (2021)
52. 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)
53. Zhang, G., Chen, J., Gao, G., Li, J., Hu, X.: Hednet: A hierarchical encoder-decoder network for 3d object detection in point clouds. arXiv preprint arXiv:2310.20234 (2023)
54. Zhang, G., Fan, L., He, C., Lei, Z., Zhang, Z., Zhang, L.: Voxel mamba: Group-free state space models for point cloud based 3d object detection. arXiv preprint arXiv:2406.10700 (2024)
55. Zhang, Y., Huang, D., Wang, Y.: Pc-rgnn: Point cloud completion and graph neural network for 3d object detection. In: Proceedings of the AAAI conference on artificial intelligence. vol. 35, pp. 3430–3437 (2021)
56. Zhang, Y., Hu, Q., Xu, G., Ma, Y., Wan, J., Guo, Y.: Not all points are equal: Learning highly efficient point-based detectors for 3d lidar point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 18953–18962 (2022)
57. Zhao, H., Jiang, L., Jia, J., Torr, P.H., Koltun, V.: Point transformer. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 16259–16268 (October 2021)
58. Zheng, W., Tang, W., Jiang, L., Fu, C.W.: Se-ssd: Self-ensembling single-stage object detector from point cloud. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14494–14503 (2021)
59. Zhou, C., Zhang, Y., Chen, J., Huang, D.: Octr: Octree-based transformer for 3d object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5166–5175 (2023)
60. Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4490–4499 (2018)
61. Zhou, Z., Zhao, X., Wang, Y., Wang, P., Foroosh, H.: Centerformer: Center-based transformer for 3d object detection. In: European Conference on Computer Vision. pp. 496–513. Springer (2022)