Stratified Transformer for 3D Point Cloud Segmentation

Xin Lai[1*] Jianhui Liu[2,3*] Li Jiang[4] Liwei Wang[1] Hengshuang Zhao[2,5]
Shu Liu[3] Xiaojuan Qi[2†] Jiaya Jia[1,3]
[1]CUHK [2]HKU [3]SmartMore [4]MPI Informatics [5]MIT

Abstract

3D point cloud segmentation has made tremendous progress in recent years. Most current methods focus on aggregating local features, but fail to directly model long-range dependencies. In this paper, we propose ***Stratified Transformer*** *that is able to capture long-range contexts and demonstrates strong generalization ability and high performance. Specifically, we first put forward a novel key sampling strategy. For each query point, we sample nearby points densely and distant points sparsely as its keys in a stratified way, which enables the model to enlarge the effective receptive field and enjoy long-range contexts at a low computational cost. Also, to combat the challenges posed by irregular point arrangements, we propose first-layer point embedding to aggregate local information, which facilitates convergence and boosts performance. Besides, we adopt contextual relative position encoding to adaptively capture position information. Finally, a memory-efficient implementation is introduced to overcome the issue of varying point numbers in each window. Extensive experiments demonstrate the effectiveness and superiority of our method on S3DIS, ScanNetv2 and ShapeNetPart datasets. Code is available at https://github.com/dvlab-research/Stratified-Transformer.*

1. Introduction

Nowadays 3D point clouds can be conveniently collected. They have demonstrated great potential in various applications, such as autonomous driving, robotics and augmented reality. Unlike regular pixels in 2D images, 3D points are arranged irregularly, hampering direct adoption of well-studied 2D networks to process 3D data. Therefore, it is imperative to explore advanced methods that are tailored for 3D point cloud data.

Abundant methods [7, 15, 16, 34, 35, 41, 52, 61, 62] have explored 3D point cloud segmentation and obtained decent performance. Most of them focus on aggregating local features, but fail to explicitly model long-range dependencies, which has been demonstrated to be crucial in capturing contexts from a long distance [49]. Along another line of research, Transformer [44] can naturally harvest long-range information via the self-attention mechanism. However, only limited attempts [31, 62] have been made to apply Transformer to 3D point clouds. Point Transformer [62] proposes "vector self-attention" and "subtraction relation" to aggregate local features, but it is still difficult to directly capture long-range contexts. Voxel Transformer [31] is tailored for object detection and performs self-attention over the voxels, but it loses accurate position due to voxelization.

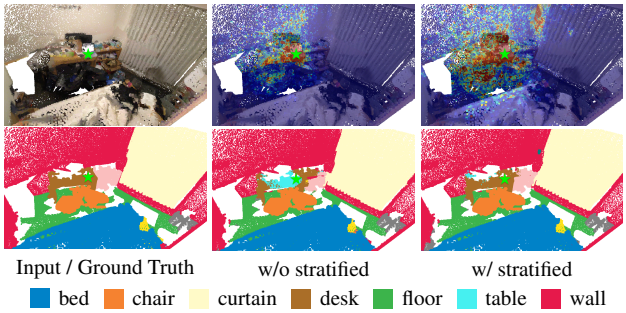


Figure 1. Visualization of Effective Receptive Field (ERF) [29], given the feature of interest (shown with green star) in the output layer. Red region corresponds to high contribution. **Left**: Input point cloud and the ground truth. **Middle**: The ERF and prediction of the model without stratified strategy and by only attending to its own window. **Right**: The ERF and prediction of the model with direct long-range dependency, using the stratified strategy. More illustrations are shown in the supplementary file.

Differently, we develop an efficient segmentation network to capture long-range contexts using the standard multi-head self-attention [44], while keeping position information intact. To this end, we propose a simple and powerful framework, namely, *Stratified Transformer*.

Specifically, we first partition the 3D space into non-overlapping cubic windows, inspired by Swin Transformer [26]. However, in Swin Transformer, different windows work independently, and each query token only chooses the tokens within its window as keys, thus attend-

*Equal Contribution
†Corresponding Author

ing to a limited local region. Instead, we propose a stratified strategy for sampling keys. Rather than only selecting nearby points in the same window as keys, we also sparsely sample distant points. In this way, for each query point, both denser nearby points and sparser distant points are sampled to form the keys all together, achieving a significantly enlarged effective receptive field while incurring negligible extra computations. For instance, we visualize the Effective Receptive Field (ERF) [29] in Fig. 1 to show the importance of modeling long-range contexts. In the middle of the figure, due to incapability to model the direct long-range dependency, the *desk* merely attends to the local region, leading to false predictions. Contrarily, with our proposed stratified strategy, the *desk* is able to aggregate contexts from distant objects, such as the *bed* or *curtain*, which helps to correct the prediction.

Moreover, it is notable that irregular point arrangements pose significant challenges in designing 3D Transformer. In 2D images, patch-wise tokens can be easily formed with spatially regular pixels. But 3D points are completely different. In our framework, each point is deemed as a token and we perform point embedding for each point to aggregate local information in the first layer, which is beneficial for faster convergence and stronger performance. Furthermore, we adopt effective relative position encoding to capture richer position information. It can generate the positional bias dynamically with contexts, through the interaction with the semantic features. Also, considering that 3D point numbers in different windows vary a lot and cause unnecessary memory occupation for windows with a small number of points, we introduce a memory-efficient implementation to significantly reduce memory consumption.

In total, our contribution is threefold:

- We propose Stratified Transformer to additionally sample distant points as keys but in a sparser way, enlarging the effective receptive field and building direct long-range dependency while incurring negligible extra computations.
- To handle irregular point arrangements, we design first-layer point embedding and effective contextual position encoding, along with a memory-efficient implementation, to build a strong Transformer tailored for 3D point cloud segmentation.
- Experiments show our model achieves state-of-the-art results on widely adopted large-scale segmentation datasets, *i.e.*, S3DIS [1], ScanNetv2 [10] and ShapeNetPart [5]. Extensive ablation studies verify the benefit of each component.

2. Related Work

Vision Transformer. Recently, vision Transformer [44] becomes popular in 2D image understanding [4, 9, 12, 13, 26, 31, 38, 42, 43, 47, 48, 57, 60, 63]. ViT [13] treats each patch as a token, and directly uses a Transformer encoder to extract features for image classification. Further, PVT [48] proposes a hierarchical structure to obtain a pyramid of features for semantic segmentation and also presents Spatial Reduction Attention to save memory. Alternatively, Swin Transformer [26] uses a window-based attention, and proposes a shifted window operation in the successive Transformer block. Methods of [9, 12, 57] further propose different designs to incorporate long-range and global dependencies. Transformer is already popular in 2D, but remains under-explored on point clouds. Inspired by Swin Transformer, we adopt hierarchical structure and shifted window operation for 3D point cloud. On top of that, we propose a stratified strategy for sampling keys to harvest long-range contexts, and put forward several essential designs to combat the challenges posed by irregular point arrangements.

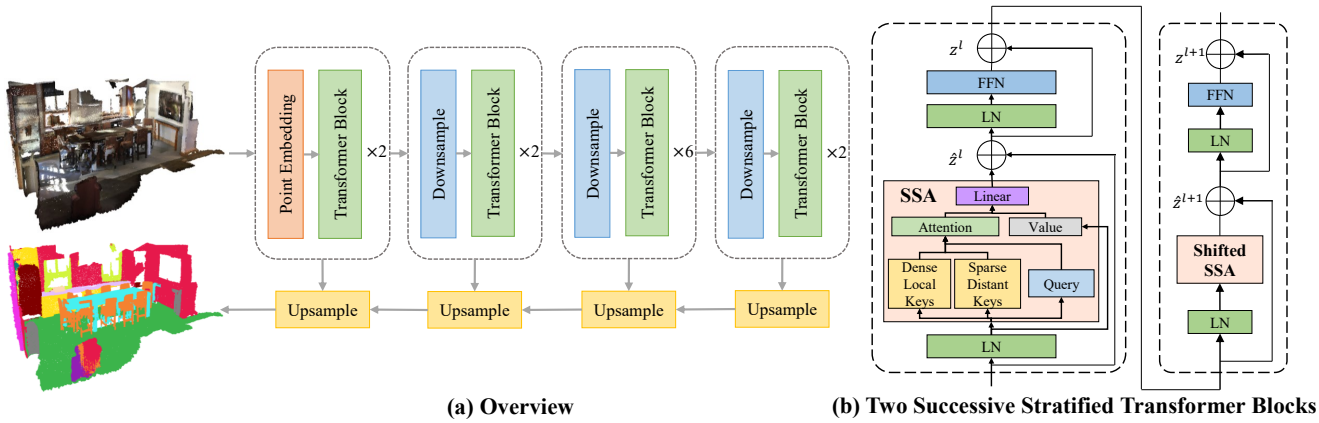
Point Cloud Segmentation. Approaches for point cloud segmentation can be grouped into two categories, *i.e.*, voxel-based and the point-based methods. The voxel-based solutions [7, 15, 16] first divide the 3D space into regular voxels, and then apply sparse convolutions upon them. They yield decent performance, but suffer from inaccurate position information due to voxelization. Point-based methods [2,6,8,11,14,17–24,28,30,33–35,37,39–41,45,46,50,52,54–56,58,59,61,62] directly adopt the point features and positions as inputs, thus keeping the position information intact. Following this line of research, different ways for feature aggregation are designed to learn high-level semantic features. PointNet and its variants [34,35] use max pooling to aggregate features. PointConv [52] and KPConv [41] try to use an MLP or discrete kernel points to mimic a continuous convolution kernel. Point Transformer [62] uses the "vector self-attention" operator to aggregate local features and the "subtraction relation" to generate the attention weights, but it suffers from lack of long-range contexts and insufficient robustness upon various perturbations in testing.

Our work is pointed-based and closely related point transformer yet with a fundamental difference: ours overcomes the limited effective receptive field issue and makes the best of Transformer for modeling long-range contextual dependencies instead of merely local aggregation.

3. Our Method

3.1. Overview

The overview of our model is illustrated in Fig. 2. Our framework is point-based, and we use both *xyz* coordinates and *rgb* colors as input. The encoder-decoder structure is adopted where the encoder is composed of multiple stages connected by downsample layers. At the beginning of the encoder, the first-layer point embedding module is used for local aggregation. Then, there are several Transformer



(a) Overview (b) Two Successive Stratified Transformer Blocks

Figure 2. (a) Framework Overview. (b) Structure of Stratified Transformer Block. Hierarchical structure is employed to obtain multi-level features. Input point clouds firstly go through the Point Embedding module to aggregate local structure information. After several downsample layers and transformer blocks, the features are upsampled for segmentation. SSA: Stratified Self-attention. Shifted SSA: SSA with shifted window. Best viewed in color.

blocks at each stage. As for the decoder, the encoder features are upsampled to become denser layer by layer in the way similar to U-Net [36].

3.2. Transformer Block

The Transformer block is composed of a standard multi-head self-attention module and a feed-forward network (FFN). With tens of thousands of points as inputs, directly applying global self-attention incurs unacceptable $O(N^2)$ memory consumption, where N is the input point number.

Vanilla Version. To this end, we employ window-based self-attention. The 3D space is firstly partitioned into non-overlapping cubic windows, where the points are scattered in different windows. Instead of attending to all the points as in global self-attention, each query point only needs to consider neighbors in the same window. Multi-head self-attention is performed in each window independently. Since different windows may contain varying numbers of points, we denote k_t as the number of points within the t-th window. Formally, given that N_h is the number of heads, N_d is the dimension of each head and $N_c = N_h \times N_d$ is the feature dimension, for the input points in the t-th window $\mathbf{x} \in \mathbb{R}^{k_t \times (N_h \times N_d)}$, the multi-head self-attention in the t-th window is formulated as

$$\begin{aligned} \mathbf{q} = Linear_q(\mathbf{x}), \quad \mathbf{k} = Linear_k(\mathbf{x}), \quad \mathbf{v} = Linear_v(\mathbf{x}), \\ \mathbf{attn}_{i,j,h} &= \mathbf{q}_{i,h} \cdot \mathbf{k}_{j,h}, \\ \hat{\mathbf{attn}}_{i,.,h} &= softmax(\mathbf{attn}_{i,.,h}), \\ \mathbf{y}_{i,h} &= \sum_{j=1}^{k_t} \hat{\mathbf{attn}}_{i,j,h} \times \mathbf{v}_{j,h}, \\ \hat{\mathbf{z}} = Linear(\mathbf{y}), \end{aligned} \tag{1}$$

where $\mathbf{q}, \mathbf{k}, \mathbf{v} \in \mathbb{R}^{k_t \times N_h \times N_d}$ are obtained from $\mathbf{x}$ by three linear layers, and $\cdot$ means dot product between vectors $\mathbf{q}_{i,h}$ and $\mathbf{k}_{j,h}$. $\mathbf{attn} \in \mathbb{R}^{k_t \times k_t \times N_h}$ is the attention map, and $\mathbf{y} \in \mathbb{R}^{k_t \times N_h \times N_d}$ is the aggregated feature, which is further projected to the output feature $\hat{\mathbf{z}} \in \mathbb{R}^{k_t \times (N_h \times N_d)}$.

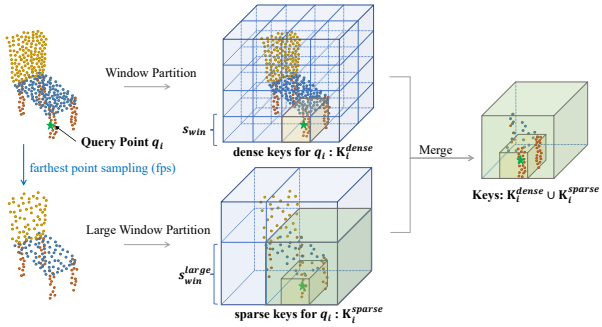


Figure 3. Illustration of the stratified strategy for keys sampling. The green star denotes the given query point.

Note that the above equations only show the calculation in a single window, and different windows work in the same way independently. In this way, the memory complexity is dramatically reduced to $O(\frac{N}{k} \times k^2) = O(N \times k)$, where k is the average number of points scattered in each window.

To facilitate cross-window communication, we also shift the window by half of the window size between two successive Transformer blocks, similar to [26]. The illustration of shifted window is given in the supplementary file.

Stratified Key-sampling Strategy. Since every query point only attends to the local points in its own window, the vanilla version Transformer block suffers from limited effective receptive field even with shifted window, as shown in Fig. 1. Therefore, it fails to capture long-range contextual dependencies over distant objects, causing false predictions.

A simple solution is to enlarge the size of cubic window. However, the memory would grow as the window size increases. To effectively aggregate long-range contexts at a low cost of memory, we propose a stratified strategy

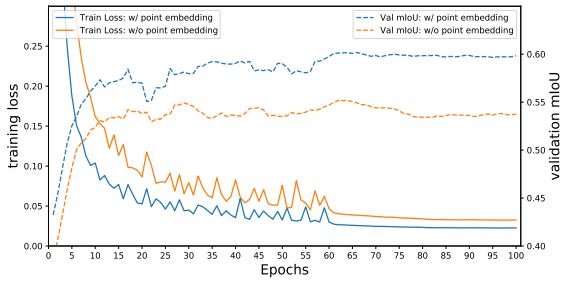


Figure 4. Plot of training loss (solid line) and validation mIoU (dotted line) in the training process. The models w/ (blue curve) and w/o (orange curve) first-layer point embedding are compared.

for sampling keys. As shown in Fig. 3, we partition the space into non-overlapping cubic windows with the window size s_{win}. For each query point q_i (shown with green star), we find the points $\mathbf{K}_i^{dense}$ in its window, same as the vanilla version. Additionally, we downsample the input points through farthest point sampling (fps) at the scale of s, and find the points $\mathbf{K}_i^{sparse}$ with a larger window size s_{win}^{large}. In the end, both dense and sparse keys form the final keys, *i.e.*, $\mathbf{K}_i = \mathbf{K}_i^{dense} \cup \mathbf{K}_i^{sparse}$. Note that duplicated key points are only counted once.

The complete structure of Stratified Transformer block is shown in Fig. 2 (b). Following common practice, we use LayerNorm [3] before each self-attention module or feed-forward network. To further complement the information interaction across windows, the original window is shifted by $\frac{1}{2}s_{win}$ while the large window is shifted by $\frac{1}{2}s_{win}^{large}$ in the successive Transformer block. This further boosts the performance as listed in Table 7.

Thanks to the stratified strategy for key sampling, the effective receptive field is enlarged remarkably and the query feature is able to effectively aggregate long-range contexts. Compared to the vanilla version, we merely incur the extra computations on the sparse distant keys, which only takes up about 10% of the final keys $\mathbf{K}_i$.

3.3. First-layer Point Embedding

In the first layer, we build a point embedding module. An intuitive choice is to use a linear layer or MLP to project the input features to a high dimension. However, we empirically observe relatively slow convergence and poor performance by using a linear layer in the first layer, as shown in Fig. 4. We note that the point feature from a linear layer or MLP merely comprises the raw information of its own *xyz* position and the *rgb* color, but it lacks local geometric and contextual information. As a result, in the first Transformer block, the attention map could not capture high-level relevance between the queries and keys that only contain raw *xyz* and *rgb* information. This negatively affects representation power and generalization ability of the model.

We contrarily propose to aggregate the features of local neighbors for each point in the Point Embedding module. We try a variety of methods for local aggregation, such as max pooling and average pooling, and find KPConv performs the best, as shown in Table 5. Surprisingly, this minor modification to the architecture brings about considerable improvement as suggested in Exp.I and II as well as Exp.V and VI of Table 4. It proves the importance of initial local aggregation in the Transformer-based networks. Note that a single KPConv incurs negligible extra computations (merely 2% FLOPs) compared to the whole network.

3.4. Contextual Relative Position Encoding

Compared to 2D spatially regular pixels, 3D points are in a more complicated continuous space, posing challenges to exploit the *xyz* position. [32] claims that position encoding is unnecessary for 3D Transformer-based networks because the *xyz* coordinates have already been used as the input features. However, although the input of the Transformer block has already contained the *xyz* position, fine-grained position information may be lost in high-level features when going deeper through the network. To make better use of the position information, we adopt a context-based adaptive relative position encoding scheme inspired by [51].

Particularly, for the point features $\mathbf{x} \in \mathbb{R}^{k_t \times (N_h \times N_d)}$ in the t-th window, we denote the *xyz* coordinates as $\mathbf{p} \in \mathbb{R}^{k_t \times 3}$. So, the relative xyz coordinates $\mathbf{r} \in \mathbb{R}^{k_t \times k_t \times 3}$ between the queries and keys are formulated as

$$\mathbf{r}_{i,j,m} = \mathbf{p}_{i,m} - \mathbf{p}_{j,m}, \quad 1 \le i, j \le k_t, m \in \{1, 2, 3\}. \tag{2}$$

To map relative coordinates to the corresponding position encoding, we maintain three learnable look-up tables $\mathbf{t}_x, \mathbf{t}_y, \mathbf{t}_z \in \mathbb{R}^{L \times (N_h \times N_d)}$ corresponding to x, y and z axis, respectively. As the relative coordinates are continuous floating-point numbers, we uniformly quantize the range of $\mathbf{r}_{i,j,m}$, *i.e.*, $(-s_{win}, s_{win})$ into L discrete parts and map the relative coordinates $\mathbf{r}_{i,j,m}$ to the indices of the tables as

$$\mathbf{idx}_{i,j,m} = \lfloor \frac{\mathbf{r}_{i,j,m} + s_{win}}{s_{quant}} \rfloor, \tag{3}$$

where s_{win} is the window size and $s_{quant} = \frac{2 \cdot s_{win}}{L}$ is the quantization size, and $\lfloor \cdot \rfloor$ denotes floor rounding.

We look up the tables to retrieve corresponding embedding with the index and sum them up to obtain the position encoding of

$$\mathbf{e}_{i,j} = \mathbf{t}_x[\mathbf{idx}_{i,j,1}] + \mathbf{t}_y[\mathbf{idx}_{i,j,2}] + \mathbf{t}_z[\mathbf{idx}_{i,j,3}], \tag{4}$$

where $\mathbf{t}[idx] \in \mathbb{R}^{N_h \times N_d}$ means the idx-th entry of the table $\mathbf{t}$, and $\mathbf{e} \in \mathbb{R}^{k_t \times k_t \times N_h \times N_d}$ is the position encoding.

Practically, the tables for query, key and value are not shared. So we differentiate among them by adding a superscript, where $\mathbf{t}_x^q$ denotes the x-axis table for the query. Similarly, the position encoding corresponding to query, key and value is denoted by $\mathbf{e}^q$, $\mathbf{e}^k$ and $\mathbf{e}^v$, respectively.

Then the position encoding performs dot product with the query and key feature to obtain the positional bias

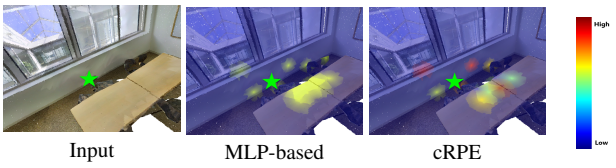


Figure 5. Visualization of the positional bias of each key at the first head of the last transformer block given the query point (shown with green star). The color map is shown on the right.

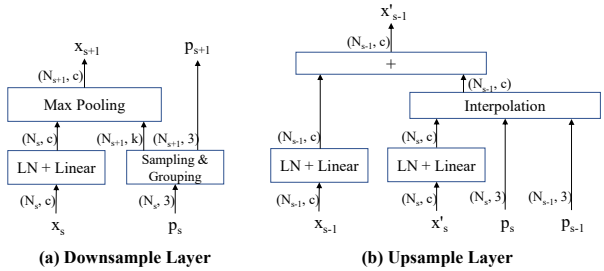


Figure 6. Structural illustration of (a) Downsample Layer and (b) Upsample Layer.

$\mathbf{pos\_bias} \in \mathbb{R}^{k_t \times k_t \times N_h}$, which is then added to the attention map. Also, we add the value feature with its corresponding position encoding, followed by the weighted sum aggregation. Finally, the original equations Eq. (1) are updated to the contextual Relative Position Encoding (cRPE) version of

$$\begin{aligned}
\mathbf{pos_bias}^{cRPE}_{i,j,h} &= \mathbf{q}_{i,h} \cdot \mathbf{e}^{q}_{i,j,h} + \mathbf{k}_{j,h} \cdot \mathbf{e}^{k}_{i,j,h}, \\
\mathbf{attn}^{cRPE}_{i,j,h} &= \mathbf{q}_{i,h} \cdot \mathbf{k}_{j,h} + \mathbf{pos_bias}^{cRPE}_{i,j,h}, \\
\hat{\mathbf{attn}}^{cRPE}_{i,.,h} &= softmax(\mathbf{attn}^{cRPE}_{i,.,h}), \\
\mathbf{y}^{cRPE}_{i,h} &= \sum_{j=1}^{k_t} \hat{\mathbf{attn}}^{cRPE}_{i,j,h} \times (\mathbf{v}_{j,h} + \mathbf{e}^{v}_{i,j,h}).
\end{aligned}$$

Compared to the MLP-based position encoding, where the relative *xyz* coordinates $\mathbf{r} \in \mathbb{R}^{k_t \times k_t \times 3}$ are directly projected to the positional bias $\mathbf{pe\_bias} \in \mathbb{R}^{k_t \times k_t \times N_h}$ via an MLP, cRPE adaptively generates the positional bias through the dot product with queries and keys, thus providing semantic information. The positional bias of the MLP-based and cRPE are visualized in Fig. 5. It reveals the fact that the positional bias generated by the MLP-based model is similar among the keys. So it makes little difference to the attention weights. But for cRPE, the positional bias varies a lot for different keys. Besides, Exp. III and IV and Exp. V and VIII of Table 4 also show the superiority of cRPE.

3.5. Downsample and Upsample Layers

The Downsample Layer is shown in Fig. 6 (a). First, the *xyz* coordinates $\mathbf{p}_s$ go through the Sampling & Grouping module, where we first sample centroid points $\mathbf{p}_{s+1}$ by fathest point sampling (fps) [35] and then use kNN to query the original points to get the grouping index $\mathbf{idx_{group}} \in \mathbb{R}^{N_{s+1} \times k}$. The number of centroid points is $\frac{1}{4}$ of the original points, *i.e.*, $N_{s+1} = \lceil \frac{1}{4} N_s \rceil$. Meanwhile, the point features $\mathbf{x}_s$ are fed into a Pre-LN [53] linear projection layer. Further, we exploit max pooling to aggregate the projected features using the grouping index, yielding the output features $\mathbf{x}_{s+1}$.

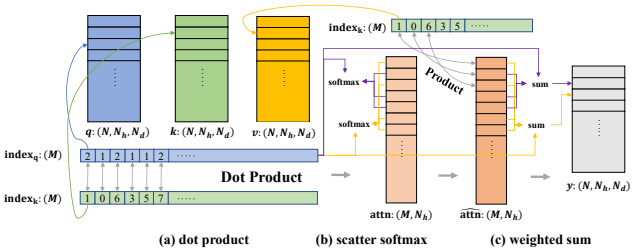


Figure 7. Memory-efficient implementation includes three steps: (a) dot product; (b) scatter softmax; (c) weighted sum. It is best viewed in color and by zoom-in.

For the upsample layer, as shown in Fig. 6 (b), the decoder features $\mathbf{x}'_s$ are firstly projected by a Pre-LN linear layer. We perform interpolation [35] between current *xyz* coordinates $\mathbf{p}_s$ and the previous ones $\mathbf{p}_{s-1}$. The encoder point features in the previous stage $\mathbf{x}_{s-1}$ go through a Pre-LN linear layer. Finally, we sum them up to yield the next decoder features $\mathbf{x}'_{s-1}$.

4. Memory-efficient Implementation

In 2D Swin Transformer, it is easy to implement the window-based attention because the number of tokens is fixed in each window. Nevertheless, due to the irregular point arrangements in 3D, the number of the tokens in each window varies a lot. A simple solution is to pad the tokens in each window to the maximum token number k_{max} with dummy tokens, and then apply a masked self-attention. But this solution wastes much memory and computations.

Instead, we first pre-compute all pairs of query and key that need to perform dot product. As shown in Fig. 7 (a), we use two indices of $\mathbf{index}_q, \mathbf{index}_k \in \mathbb{R}^M$, to index the $\mathbf{q}$ and $\mathbf{k}$ of shape (N, N_h, N_d), respectively, where N denotes the total number of input points. Then, we perform dot product between the entries indexed by $\mathbf{index}_q$ and $\mathbf{index}_k$, yielding the attention map $\mathbf{attn}$ of the shape (M, N_h). Afterwards, as shown in Fig. 7 (b), we perform the scatter softmax directly on $\mathbf{attn}$ with the query index $\mathbf{index}_q$, where the softmax function is applied on the entries in $\mathbf{attn}$ with the same index in $\mathbf{index}_q$. Further, as shown in Fig. 7 (c), we use $\mathbf{index}_k$ to index the values $\mathbf{v}$ and multiply them with the attention map $\mathbf{attn}$. We finally sum up the entries with the same index in $\mathbf{index}_q$ and save the results into the output features $\mathbf{y}$. Note that each of the steps is implemented by a single CUDA kernel. So the intermediate variables inside each step hardly occupy memory. In this way, we reach the memory complexity of $O(M \cdot N_h)$, much less than that used in vanilla implementation. More detailed memory complexity analysis and discussion of po-

sition encoding implementation are given in the supplementary file. Our implementation saves 57% memory compared to the vanilla one.

5. Experiments

5.1. Experimental Setting

Network Architecture. The main architecture is shown in Fig. 2. Both the *xyz* coordinates and *rgb* colors are used as inputs. We set the initial feature dimension and number of heads to 48 and 3 respectively, and they will double in each downsample layer. As for S3DIS, four stages are constructed with the block depths [2, 2, 6, 2]. In contrast, for ScanNetv2, we note that the point number is larger. So we add an extra downsample layer on top of the first-layer point embedding module. Then, the later four stages with block depths [3, 9, 3, 3] are added. So a total of five stages are constructed for ScanNetv2.

Implementation Detail. For S3DIS, following previous work [62], we train for $76,500$ iterations with 4 RTX 2080Ti GPUs. The batch size is set to 8. Following common practice, the raw input points are firstly grid sampled with the grid size set to 0.04m. During training, the maximum input points number is set to $80,000$, and all extra ones are discarded if points number reaches this number. The window size is set to 0.16m initially, and it doubles after each downsample layer. The downsample scale for the stratified sampling strategy is set to 8. Unless otherwise specified, we use z-axis rotation, scale, jitter and drop color as data augmentation.

For ScanNetv2, we train for 600 epochs with weight decay and batch size set to 0.1 and 8 respectively, and the grid size for grid sampling is set to 0.02m. At most $120,000$ points of a point cloud are fed into the network during training. The initial window size is set to 0.1m. And the downsample scale for the stratified sampling is set to 4. Except random jitter, the data augmentation is the same as that on S3DIS. The implementation details for ShapeNetPart and the datasets descriptions are given in the supplementary file.

5.2. Results

We make comparisons with recent state-of-the-art semantic segmentation methods. Tables 1 and 2 show the results on S3DIS and ScanNetv2 datasets. Our method achieves state-of-the-art performance on both challenging datasets. On S3DIS, ours outperforms others significantly, even higher than Point Transformer [62] by 1.6% mIoU. On ScanNetv2, the validation mIoU of our method surpasses others including voxel-based methods, with a gap of 2.1% mIoU. On the test set, ours achieves slightly higher results than MinkowskiNet [7]. The potential reason may be the points in ScanNetv2 are relatively sparse. So the loss of accurate position in voxelization is negligible for voxel-based methods. But on S3DIS where points are denser, our method outperforms MinkowskiNet with a huge gap, *i.e.*, 6.6% mIoU. Also, ours outperforms MinkowskiNet by 2.1% mIoU on the validation set and is much more robust than MinkowskiNet when encountering various perturbations in testing, as shown in Table 9. Notably, it is the first time for the point-based methods to achieve higher performance compared with voxel-based methods on ScanNetv2.

Also, in Table 3, to show the generalization ability, we also make comparison on ShapeNetPart [5] for the task of part segmentation. Our method outperforms previous ones and achieves new state of the art in terms of both category mIoU and instance mIoU. Although the instance mIoU of ours is comparable to Point Transformer, ours outperforms

Method	Input	OA	mAcc	mIoU
PointNet [34]	point	-	49.0	41.1
SegCloud [40]	point	-	57.4	48.9
TangentConv [39]	point	-	62.2	52.6
PointCNN [22]	point	85.9	63.9	57.3
PointWeb [61]	point	87.0	66.6	60.3
HPEIN [19]	point	87.2	68.3	61.9
GACNet [45]	point	87.8	-	62.9
PAT [58]	point	-	70.8	60.1
ParamConv [46]	point	-	67.0	58.3
SPGraph [20]	point	86.4	66.5	58.0
SegGCN [21]	point	88.2	70.4	63.6
MinkowskiNet [7]	voxel	-	71.7	65.4
PAConv [54]	point	-	-	66.6
KPConv [41]	point	-	72.8	67.1
PointTransformer [62]	point	90.8	76.5	70.4
Ours	point	**91.5**	**78.1**	**72.0**

Table 1. Results on S3DIS Area5 for semantic segmentation.

Method	Input	Val mIoU	Test mIoU
PointNet++ [35]	point	53.5	55.7
3DMV [11]	point	-	48.4
PanopticFusion [33]	point	-	52.9
PointCNN [22]	point	-	45.8
PointConv [52]	point	61.0	66.6
JointPointBased [6]	point	69.2	63.4
PointASNL [56]	point	63.5	66.6
SegGCN [21]	point	-	58.9
RandLA-Net [17]	point	-	64.5
KPConv [41]	point	69.2	68.6
JSENet [18]	point	-	69.9
FusionNet [59]	point	-	68.8
PointTransformer [62]	point	70.6	-
SparseConvNet [15]	voxel	69.3	72.5
MinkowskiNet [7]	voxel	72.2	73.6
Ours	point	**74.3**	**73.7**

Table 2. Results on ScanNetv2 for semantic segmentation. More results and analysis are included in the supplementary file.

Method	Cat. mIoU	Ins. mIoU
PointNet [34]	80.4	83.7
PointNet++ [35]	81.9	85.1
PCNN [2]	81.8	85.1
SpiderCNN [55]	82.4	85.3
SPLATNet [37]	83.7	85.4
DGCNN [50]	82.3	85.2
SubSparseCNN [15]	83.3	86.0
PointCNN [22]	84.6	86.1
PointConv [52]	82.8	85.7
Point2Sequence [23]	-	85.2
PVCNN [27]	-	86.2
RS-CNN [25]	84.0	86.2
KPConv [41]	85.0	86.2
InterpCNN [30]	84.0	86.3
DensePoint [24]	84.2	86.4
PAConv [54]	84.6	86.1
PointTransformer [62]	83.7	**86.6**
Ours	**85.1**	**86.6**

Table 3. Results on ShapeNetPart for part segmentation.

ID	PointEmb	Aug	cRPE	Stratified	S3DIS	ScanNet
I					56.8	56.8
II	✓				61.3	69.6
III	✓	✓			67.2	70.6
IV	✓	✓	✓		70.1	72.5
V	✓	✓	✓	✓	**72.0**	**73.7**
VI		✓	✓	✓	70.0	69.7
VII	✓		✓	✓	66.1	72.3
VIII	✓	✓		✓	68.0	71.4

Table 4. Ablation study. **PointEmb**: First-layer Point Embedding. **Aug**: Data Augmentation. **cRPE**: contextual Relative Position Encoding. **Stratified**: Stratified Transformer Block. Metric: mIoU.

Point Transformer by a large margin in category mIoU.

5.3. Ablation Study

We conduct extensive ablation studies to verify the effectiveness of each component in our method, and show results in Table 4. To make our conclusions more convincing, we make evaluations on both S3DIS and ScanNetv2 datasets. From Exp.I to V, we add one component each time. Also, from Exp.VI to VIII, we make double verification by removing each component from the final model, *i.e.*, Exp.V.

Stratified Transformer. In Table 4, comparing Exp.IV and V, we notice that with the stratified strategy, the model improves with 1.9% mIoU on S3DIS and 1.2% mIoU on ScanNetv2. Combining the visualizations in Fig. 1, we note that the stratified strategy is able to enlarge the effective receptive field and boost the performance. Besides, we also show the effect when setting different downsample scales, *i.e.*, 4, 8 and 16, in the supplementary file.

First-layer Point Embedding. We compare Exp.I with II, and find the model improves by a large margin with first-layer point embedding. Also, we compare Exp.VI and V,

Method	Linear	PointTrans block	Max pool	Avg pool	KPConv
mIoU	68.9	69.7	70.3	71.0	72.0
Δ	-	+0.8	+1.4	+2.1	+3.1

Table 5. Comparison among different ways of first-layer point embedding on S3DIS. PointTrans block: Point Transformer block.

Query			✓			✓	✓		✓
Key				✓		✓		✓	✓
Value					✓		✓	✓	✓
MLP	✓								
mIoU	68.0	68.0	70.2	70.5	70.8	70.8	71.0	70.8	72.0

Table 6. Ablation study on cRPE. We evaluate on the S3DIS dataset. **Query, Key and Value**: applying the cRPE on the corresponding features to get positional bias. **MLP**: MLP-based relative position encoding.

where the model gets 2.0% mIoU gain on S3DIS and 4.0% mIoU gain on ScanNetv2 with the equipment of first-layer point embedding. This minor modification in the architecture brings considerable benefit.

To further explore the role of local aggregation in first-layer point embedding, we compare different ways of local aggregation with linear projection in Table 5. Obviously, all listed local aggregation methods are better than linear projection for the first-layer point embedding.

Contextual Relative Position Encoding. From Exp.III to IV, the performance increases by 2.9% mIoU on S3DIS and 1.9% mIoU on ScanNetv2 after using cRPE. Moreover, when also using the stratified Transformer, the model still improves with 4.0% mIoU gain on S3DIS and 2.3% gain on ScanNetv2 equipped with cRPE, through the comparison between Exp.VIII and V.

Further, we testify the contribution of applying cRPE on each of the query, key or value features. Table 6 shows that applying cRPE in either feature can make improvement. When applying cRPE on query, key and value simultaneously, the model achieves the best performance.

In addition, we compare our approach with the MLP-based method as mentioned in Sec. 3.4. As shown in Table 6, we find the MLP-based method (the first column) actually makes no difference with the model without any position encoding (the second column). Combining the visualization in Fig. 5, we conclude that the relative position information purely based on *xyz* coordinates is not helpful, since input point features to the network have already incorporated the *xyz* coordinates. In contrast, cRPE is based on both *xyz* coordinates and contextual features.

Shifted Window. Shifted window is adopted to complement information interaction across windows. In Table 7, we compare the models w/ and w/o shifted window for both our vanilla version and Stratified Transformer on S3DIS. Evidently, shifted window is effective in our framework.

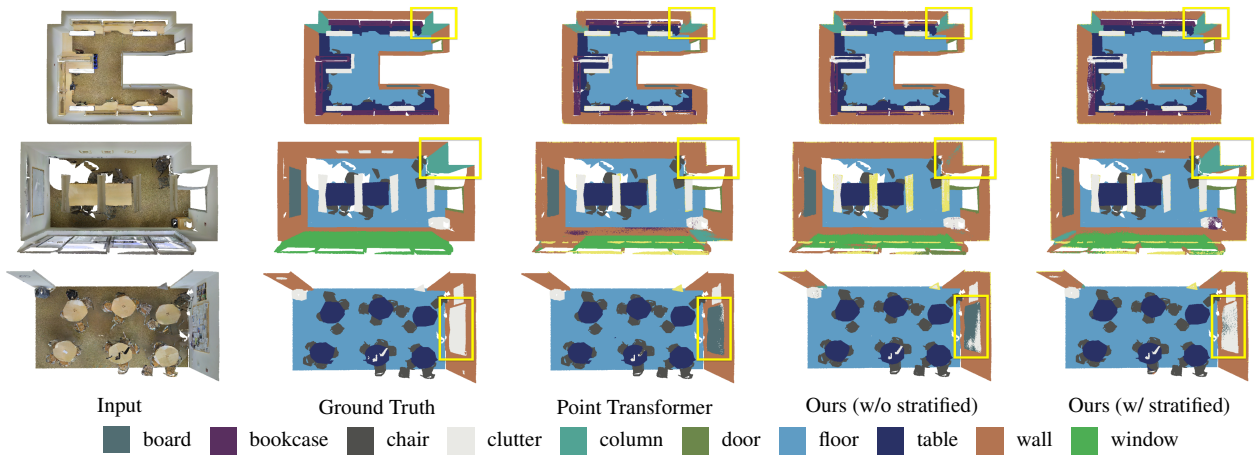


Figure 8. Visual comparison between Point Transformer, our baseline model (w/o stratified) and our proposed Stratified Transformer.

Method	w/ shift	w/o shift	shift (original)	shift (large)
vanilla	70.1	69.4	N/A	N/A
stratified	72.0	70.1	71.0	70.3

Table 7. Ablation study on shifted window. shift (original): apply shifted window only on original windows. shift (large): apply shifted window only on large windows. vanilla: vanilla version Transformer block.

aug type	no aug	jitter	rotate	drop color	scale	all
mIoU	66.1	66.3	66.4	67.0	67.3	72.0
Δ	-	+0.2	+0.3	+0.9	+1.2	+5.9

Table 8. Ablation study on data augmentation evaluated on S3DIS.

Moreover, even without shifted window, Stratified Transformer still yields higher performance, *i.e.*, 70.1% mIoU, compared to the vanilla version. Also, shifting on both original and large windows is beneficial.

Data Augmentation. Data augmentation plays an important role in training Transformer-based network. It is also the case in our framework as shown in Exp.V and VII as well as Exp.II and III. We also investigate the contribution of each augmentation in Table 8.

5.4. Robustness Study

To show the anti-interference ability of our model, we measure the robustness by applying a variety of perturbations in testing. Following [54], we make evaluations in aspects of permutation, rotation, shift, scale and jitter. As shown in Table 9, our method is extremely robust to various perturbations, while previous methods fluctuate drastically under these scenarios. It is notable that ours performs even better (+0.63% mIoU) with 90° z-axis rotation.

Although Point Transformer also employs the self-attention mechanism, it yields limited robustness. A potential reason may be Point Transformer uses special operator designs such as "vector self-attention" and "subtraction relation", rather than standard multi-head self-attention.

Method	None	Perm.	90°	180°	270°	+0.2	−0.2	×0.8	×1.2	jitter
PointNet++	59.75	59.71	58.15	57.18	58.19	22.33	29.85	56.24	59.74	59.05
Minkowski	64.68	64.56	63.45	63.83	63.36	64.59	64.96	59.60	61.93	58.96
PAConv	65.63	65.64	61.66	63.48	61.80	55.81	57.42	64.20	63.94	65.12
PointTrans	70.36	70.45	65.94	67.78	65.72	70.44	70.43	65.73	66.15	59.67
Ours	**71.96**	**72.02**	**72.59**	**72.37**	**71.86**	**71.99**	**71.93**	**70.42**	**71.21**	**72.02**

Table 9. Robustness study on S3DIS. We apply the perturbations of permutation (Perm.), z-axis rotation ($90°, 180°, 270°$), shifting (± 0.2), scaling ($\times 0.8, \times 1.2$) and jitter in testing. PointTrans: Point Transformer [62].

5.5. Visual Comparison

In Fig. 8, we visually compare Point Transformer, the baseline model and ours. It clearly shows the superiority of our method. Due to the awareness of long-range contexts, our method is able to recognize the objects highlighted with yellow box, while others fail.

6. Conclusion

We propose Stratified Transformer and achieve state-of-the-art results. The stratified strategy significantly enlarges the effective receptive field. Also, first-layer point embedding and an effective contextual relative position encoding are put forward. Our work answers two questions. First, it is possible to build direct long-range dependencies at low computational costs and yield higher performance. Second, standard Transformer can be applied to 3D point cloud with strong generalization ability and powerful performance.

Acknowledgements

The work is supported in part by Hong Kong Research Grant Council - Early Career Scheme (Grant No. 27209621), HKU Startup Fund, HKU Seed Fund for Basic Research, and SmartMore donation fund.

References

[1] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *CVPR*, 2016. 2

[2] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *TOG*, 2018. 2, 7

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *Stat*, 2016. 4

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 2

[5] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv:1512.03012*, 2015. 2, 6

[6] Hung-Yueh Chiang, Yen-Liang Lin, Yueh-Cheng Liu, and Winston H Hsu. A unified point-based framework for 3d segmentation. In *3DV*, 2019. 2, 6

[7] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019. 1, 2, 6

[8] Ruihang Chu, Yukang Chen, Tao Kong, Lu Qi, and Lei Li. Icm-3d: Instantiated category modeling for 3d instance segmentation. *IEEE Robotics and Automation Letters*, 2021. 2

[9] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. *arXiv:2104.13840*, 2021. 2

[10] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 2

[11] Angela Dai and Matthias Nießner. 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In *ECCV*, 2018. 2, 6

[12] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. *arXiv:2107.00652*, 2021. 2

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021. 2

[14] Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. Dilated point convolutions: On the receptive field size of point convolutions on 3d point clouds. In *ICRA*, 2020. 2

[15] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *CVPR*, 2018. 1, 2, 6, 7

[16] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv:1706.01307*, 2017. 1, 2

[17] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *CVPR*, 2020. 2, 6

[18] Zeyu Hu, Mingmin Zhen, Xuyang Bai, Hongbo Fu, and Chiew-lan Tai. Jsenet: Joint semantic segmentation and edge detection network for 3d point clouds. In *ECCV*, 2020. 2, 6

[19] Li Jiang, Hengshuang Zhao, Shu Liu, Xiaoyong Shen, Chi-Wing Fu, and Jiaya Jia. Hierarchical point-edge interaction network for point cloud semantic segmentation. In *ICCV*, 2019. 2, 6

[20] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, 2018. 2, 6

[21] Huan Lei, Naveed Akhtar, and Ajmal Mian. Seggcn: Efficient 3d point cloud segmentation with fuzzy spherical kernel. In *CVPR*, 2020. 2, 6

[22] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *NeurIPS*, 2018. 2, 6, 7

[23] Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. In *AAAI*, 2019. 2, 7

[24] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *ICCV*, 2019. 2, 7

[25] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *CVPR*, 2019. 7

[26] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *ICCV*, 2021. 1, 2, 3

[27] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. In *NeurIPS*, 2019. 7

[28] Ze Liu, Zheng Zhang, Yue Cao, Han Hu, and Xin Tong. Group-free 3d object detection via transformers. In *ICCV*, 2021. 2

[29] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *NeurIPS*, 2016. 1, 2

[30] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3d point cloud understanding. In *ICCV*, 2019. 2, 7

[31] Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu. Voxel transformer for 3d object detection. In *ICCV*, 2021. 1, 2

[32] Ishan Misra, Rohit Girdhar, and Armand Joulin. An end-to-end transformer model for 3d object detection. In *ICCV*, 2021. 4

[33] Gaku Narita, Takashi Seno, Tomoya Ishikawa, and Yohsuke Kaji. Panopticfusion: Online volumetric semantic mapping at the level of stuff and things. In *IROS*, 2019. 2, 6

[34] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 1, 2, 6, 7

[35] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 1, 2, 5, 6, 7

[36] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 3

[37] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *CVPR*, 2018. 2, 7

[38] Shuyang Sun, Xiaoyu Yue, Song Bai, and Philip Torr. Visual parser: Representing part-whole hierarchies with transformers. *arXiv:2107.05790*, 2021. 2

[39] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *CVPR*, 2018. 2, 6

[40] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *3DV*, 2017. 2, 6

[41] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*, 2019. 1, 2, 6, 7

[42] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 2

[43] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv:2103.17239*, 2021. 2

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1, 2

[45] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *CVPR*, 2019. 2, 6

[46] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *CVPR*, 2018. 2, 6

[47] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvtv2: Improved baselines with pyramid vision transformer. *arXiv:2106.13797*, 2021. 2

[48] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, 2021. 2

[49] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018. 1

[50] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *TOG*, 2019. 2, 7

[51] Kan Wu, Houwen Peng, Minghao Chen, Jianlong Fu, and Hongyang Chao. Rethinking and improving relative position encoding for vision transformer. In *ICCV*, 2021. 4

[52] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, 2019. 1, 2, 6, 7

[53] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *ICML*, 2020. 5

[54] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *CVPR*, 2021. 2, 6, 7, 8

[55] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *ECCV*, 2018. 2, 7

[56] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *CVPR*, 2020. 2, 6

[57] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers. In *NeurIPS*, 2021. 2

[58] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. In *CVPR*, 2019. 2, 6

[59] Feihu Zhang, Jin Fang, Benjamin Wah, and Philip Torr. Deep fusionnet for point cloud semantic segmentation. In *ECCV*, 2020. 2, 6

[60] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *CVPR*, 2020. 2

[61] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *CVPR*, 2019. 1, 2, 6

[62] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer. In *ICCV*, 2021. 1, 2, 6, 7, 8

[63] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *ICLR*, 2020. 2