Fast Point Transformer
—Supplementary Material—

Chunghyun Park Yoonwoo Jeong Minsu Cho Jaesik Park
POSTECH GSAI & CSE
http://cvlab.postech.ac.kr/research/FPT

In this supplementary material, we provide additional experimental results and their detailed settings.

1. Experimental Details

In this section, we clarify the experimental settings for training models, latency evaluation, and model architectures in detail. Each experiment has been conducted with a fixed random seed for the reproducibility.

Training details. For 3D semantic segmentation, we use the same training configuration except the batch size and training iterations for both ScanNet [3] and S3DIS [1]. We use the SGD optimizer with momentum and weight decay as 0.9 and 0.0001, respectively. The learning rate is scheduled by the linear warm-up and cosine annealing policy from the initial learning rate 0.1 to the final learning rate 0. We train models with batch size 8 both for ScanNet and S3DIS. We train models with 100k and 40k iterations for ScanNet and S3DIS, respectively.

Latency evaluation. We describe the detailed setups that have been used during the inference time evaluation on Table 2 of the main paper. We measure the latency of each model with batch size 1 under the following environments:

1. CUDA version: 11.0
2. cuDNN version: 8.2.1
3. PyTorch version: 1.7.1
4. MinkowskiEngine version: 0.5.4
5. GPU: single NVIDIA Geforce RTX 3090
6. CPU: Intel(R) Core(TM) i7-5930K CPU @ 3.50GHz

Network architectures. Figure A1 illustrates detailed model designs of MinkowskiNet42 [2] and our Fast Point Transformer. To set the total parameter numbers to be similar, we adjust the feature dimensions as Hu et al. [4] does, resulting in similar parameter numbers; 37.9M for both models. For small models used in both Table 4 of the main paper and Table A4 of this supplementary material, we modify the number of residual blocks as the official code of MinkowskiNet [2] does. Table A3 provides the exact number of residual blocks.

Table A1. **Color reconstruction results.** We compare RGB color reconstruction quality in PSNR with the same backbone architecture as MinkowskiNet [2] except the vox/devoxelization modules. We conduct the experiments on ScanNet [3] validation set (10cm).

Method	PSNR (↑)
Conventional [2]	21.76
Our centroid-aware	23.03

Table A2. **Comparison of robustness to voxel size.** We compare mIoU scores of MinkowskiNet42† and Fast Point Transformer using a larger voxel size (5cm) for inference than the voxel size (4cm) used for training on S3DIS [1] dataset. Note that both models are trained with voxel size as 4cm.

Method	mIoU (4cm)	mIoU (5cm)
MinkowskiNet42†	67.2	64.0 (↓ 3.2)
Fast Point Transformer	68.7	67.5 (↓ 1.2)

2. Analysis on Centroid-aware Voxelization

Color reconstruction. We conduct a experiment to evaluate the effeciveness of our centroid-aware voxelization. We compare ours and the conventional voxelization [2] with the same setting from Table 5 of the main paper on ScanNet [3] validation set. We reconstruct colors (RGB) of input point clouds with MinkowskiNet [2], optimized by $l2$-difference between input colors and reconstructed colors. As shown in Table A1, ours achieves a higher PSNR by 1.27 than the conventional one, showing the effectiveness of the centroid-aware property to mitigate quantization artifact.

Robustness to voxel size. We evaluate the robustness of both MinkowskiNet42† and Fast Point Transformer to the voxel size for inference by using a larger voxel size than one used for training. For both models trained with voxel size 4cm, we measure the performance drop of each method

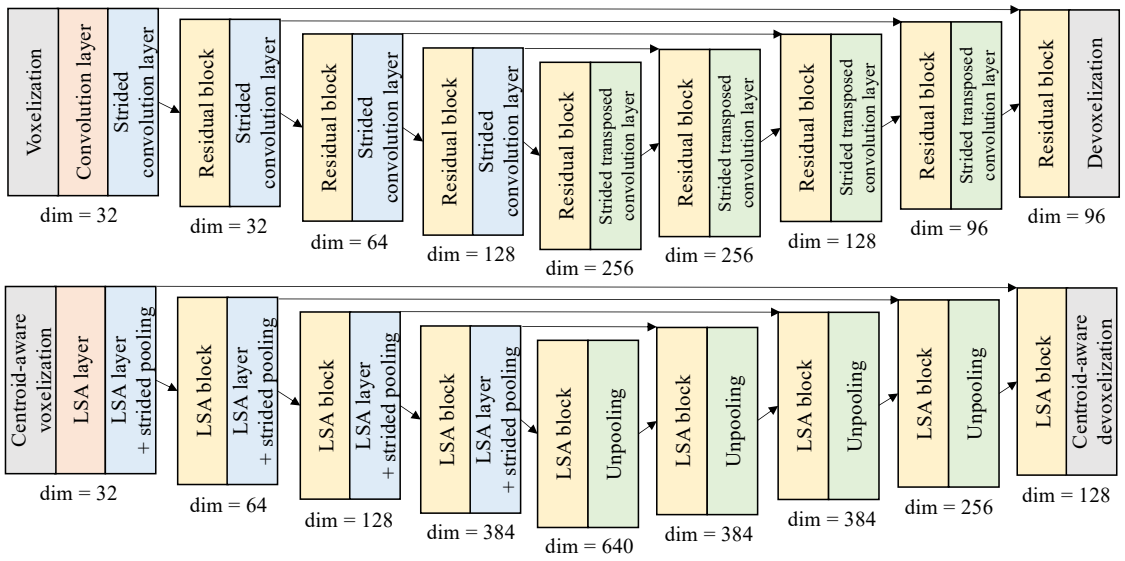

Figure A1. **Network architectures.** (*Top*) MinkowskiNet42 [2] and (*Bottom*) our Fast Point Transformer. LSA denotes the proposed lightweight self-attention. Note that both models have the same number of learnable parameters.

Table A3. **The number of residual blocks.** We apply the same configuration for both MinkowskiNet [2] and Fast Point Transformer. S1,$\cdots$, S16 denote the tensor stride in the feature map hierarchy.

Method	Encoder				Decoder			
	S2	S4	S8	S16	S8	S4	S2	S1
baseline	2	3	4	6	2	2	2	2
small	2	2	2	2	2	2	2	2
smaller	1	1	1	1	1	1	1	1

when it use voxel size 5cm for inference. The results show that the Fast Point Transformer is more robust to inference voxel size than MinkowskiNet42† as shown in Table A2.

3. Additional Experimental Results

In this section, we show further experimental results about the effect of model size on its performance, the proposed decomposition of positional encodings, and the class-wise IoU scores of both MinkowskiNet42† and our Fast Point Transformer on S3DIS [1] Area 5 test dataset.

mIoU vs. model size. We provide additional results with voxel size 2cm in Table A4. Since Fast Point Transformer shows its robustness to the number of parameters with voxel size as 5cm and 10cm, Fast Point Transformer (smaller) still achieves 70.5% of mIoU score while MinkowskiNet (smaller) only shows 68.6% with voxel size as 2cm. Interestingly, both MinkowskiNet and Fast Point Transformer show the largest performance drop with voxel size 2cm. We hypothesize that this is because the reduction of residual blocks reduces the receptive field, and the reduced receptive field is not sufficient for the model to recognize a 3D scene.

Table A4. **mIoU vs. model size.**

Method	# Param. (M)		mIoU (%)	
		Rel. (%)		Δ
*Voxel size: 10*cm				
MinkowskiNet42†	37.9	±0.0	60.5±0.2	±0.0
MinkowskiNet (small)	21.7	↓ 42.7	59.9±0.6	↓ 0.6
MinkowskiNet (smaller)	11.6	↓ 69.4	58.2±0.9	↓ 2.3
FastPointTrans. (ours)	37.9	±0.0	65.9±0.6	±0.0
FastPointTrans. (small)	20.2	↓ 46.7	66.0±0.3	↑ 0.1
FastPointTrans. (smaller)	10.8	↓ 71.5	65.7±0.1	↓ 0.2
*Voxel size: 5*cm				
MinkowskiNet42†	37.9	±0.0	66.7±0.3	±0.0
MinkowskiNet (small)	21.7	↓ 42.7	66.0±0.1	↓ 0.7
MinkowskiNet (smaller)	11.6	↓ 69.4	64.2±0.4	↓ 2.5
FastPointTrans. (ours)	37.9	±0.0	70.0±0.1	±0.0
FastPointTrans. (small)	20.2	↓ 46.7	70.3±0.2	↑ 0.3
FastPointTrans. (smaller)	10.8	↓ 71.5	69.7±0.2	↓ 0.3
*Voxel size: 2*cm				
MinkowskiNet42†	37.9	±0.0	71.9±0.2	±0.0
MinkowskiNet (small)	21.7	↓ 42.7	71.2±0.2	↓ 0.7
MinkowskiNet (smaller)	11.6	↓ 69.4	68.6±0.6	↓ 3.3
FastPointTrans. (ours)	37.9	±0.0	72.1±0.3	±0.0
FastPointTrans. (small)	20.2	↓ 46.7	71.3±0.1	↓ 0.8
FastPointTrans. (smaller)	10.8	↓ 71.5	70.5±0.1	↓ 1.6

Decomposition of positional encodings. We quantitatively measure how much memory the proposed decomposition of positional encodings can reduce. We measure the peak memory usage of both models with and without the decomposition as varying the local window size for neighbor

Table A5. **Effect of the decomposition on memory usage.** k denotes the local window size which defines the maximum number of neighbor points, $K := k^3$, within the kernel volume. We conduct the experiments on ScanNet [3] validation set (2cm).

k	Peak Memory Usage (GB)	
	Decomposition (ours)	Exact - *parallel*
3	3.613	9.519
5	3.892	23.245
7	4.494	Out of Memory

Table A6. **Sequential computation vs. Decomposition.** We conduct the experiments on ScanNet [3] validation set (2cm).

Method	Memory (GB)	Latency (sec)	mIoU (%)
Exact - *parallel*	9.52	**0.15**	**72.1**
Exact - *sequential*	**3.40**	0.48	**72.1**
Decomposition	3.61	0.17	72.0

Table A7. **Voxel-based vs. Hybrid vs. Fast Point Transformer.** We compare MinkowskiNet42 [2], SPVCNN [8] and our Fast Point Transfomer on ScanNet [3] validation set with voxel size 10cm.

Method	Memory (GB)	Latency (sec)	mIoU (%)
MinkowskiNet42†	**1.93**	**0.04**	60.4
SPVCNN	3.62	0.07	62.8
Ours	2.73	0.08	**65.3**

points on ScanNet [3]. We keep the voxel size as 2cm for the all measurements. As shown in Table A5, the models with the proposed decomposition which has the space complexity of $\mathcal{O}(ID + KD)$ show an almost constant memory usage since the number of voxel centroids I is much bigger than the number of neighbor points K. However, the models without the decomposition which has the space complexity of $\mathcal{O}(IKD)$ show a growing usage of memory. Moreover, the model with local window size 7 raises the out-of-memory error in single NVIDIA Geforce RTX 3090 GPU whose VRAM capacity is 24GB. This results show the memory-efficient property of the proposed *lightweight* self-attention (LSA). Furthermore, Table A6 shows that the LSA layer saves memory consumption and preserves fast inference time. The mIoU is almost identical to the exact approaches, indicating the effectiveness of the decomposition.

Comparison with hybrid methods. For a fair comparison, we re-implement SPVCNN [8] with MinkowskiEngine-0.5.4 since MinkowskiEngine-0.5.4 is faster than TorchSparse. As shown in Table A7, Fast Point Transformer outperforms SPVCNN by 2.5 mIoU on ScanNet [3] validation set with voxel size 10cm, consuming a less GPU memory.

Detailed experimental results on S3DIS [1]. We report the class-wise IoU scores of both MinkowskiNet42† and the proposed Fast Point Transformer on S3DIS [1] Area 5 in Table A9. We report the performance of the best model among three different experiments with the same training configuration except random seed numbers both for MinkowskiNet42† and Fast Point Transformer. There is a large gap in the latency between point-based methods [5, 7, 9–12] and voxel hashing-based methods [2] including our Fast Point Transformer as shown in Table A9. Specifically, Fast Point Transformer (4cm) outperforms MinkowskiNet42† (4cm) with rotation average by 0.4 mIoU with a 4.7 times faster speed.

4. Time Complexity Analysis

In this section, we analyze the time complexity of neighbor search used in both voxel hashing-based methods [2] including ours and point-based methods [9–12]. We first recap the reported time complexity as shown in Table A8.

MinkowskiNet [2] and Fast Point Transformer require the same process for neighbor search since both methods benefit from voxel hashing. We analyze preparation and inference time complexity on Alg. A1 and Alg. A2, respectively. We denote ours as the representative method.

KPConv [9] constructs a k-d tree before inference. With the official code of KPConv, we analyze both preparation and inference time in Alg. A3 and Alg. A4, respectively.

PointWeb [11] uses a brute-force algorithm to search the k nearest neighbors. We analyze the time complexity of the brute-force algorithm in Alg. A5.

PAConv [10] and Point Transformer [12] do not require preparation steps for neighbor search. Thus, we set the preparation time to constant time. For analyzing inference time, we have followed the official implementation. As both methods use the same algorithm for neighbor search, we denote PAConv as the representative method in Alg. A6.

5. Qualitative Results

In this section, we show further qualitative results of consistency scores, 3D semantic segmentation results, and 3D object detection on ScanNet [3]. Figure A2 shows the point-wise consistency scores of MinkowskiNet42† and our Fast Point Transformer. In addition to this consistency, Fast Point Transformer predicts more accurate 3D semantic labels (Figure A3) and 3D bounding boxes (Figure A4) qualitatively.

Table A8. **Time complexity analysis.** We denote N as the number of dataset points, M as the number of query points (or voxel centroids), and K as the number of neighbors to search. Both M and N are much larger than K in a large-scale point cloud.

Methods	Neighbor Search	
	Preparation	Inference
PointNet [7]	✗	✗
SPGraph [5]	✗	✗
PointWeb [11]	$\mathcal{O}(1)$	$\mathcal{O}(MNK)$
KPConv *deform* [9]	$\mathcal{O}(N \log N)$	$\mathcal{O}(KM \log N)$
PAConv [10]	$\mathcal{O}(1)$	$\mathcal{O}(MN \log K)$
PointTransformer [12]	$\mathcal{O}(1)$	$\mathcal{O}(MN \log K)$
MinkowskiNet [2]	$\mathcal{O}(N)$	$\mathcal{O}(M)$
FastPointTransformer (ours)	$\mathcal{O}(N)$	$\mathcal{O}(M)$

Algorithm A1 (Ours) Hash Table Construction: $\mathcal{O}(N)$

```
Number of training points: N
An empty hash table: h
for point = 1, 2, ···, N do
    Insert(h, point)                                   // O(1)
end for
```

Algorithm A2 (Ours) Inference: $\mathcal{O}(M)$

```
Number of query points: M
A constructed hash table: h̄
for query = 1, 2, ···, M do
    Lookup(h̄, query)                                   // O(1)
end for
```

Algorithm A3 (KPConv) Tree Construction: $\mathcal{O}(N \log N)$

```
Number of training points: N
An empty tree: T
for point = 1, 2, ···, N do
    Insert(T, point)                                   // O(log N)
end for
```

Algorithm A4 (KPConv) Inference: $\mathcal{O}(KM \log N)$

```
Number of training points: N
Number of query points: M
Number of neighbors to search: K
Constructed k-d tree: T̄
k-th nearest neighbors dictionary: S = {}
for query = 1, 2, ···, M do
    arr = []
    for i = 1, 2, ···, K do
        point = SearchClosest(T̄, query)                // O(log N)
        T̄ = Pop(T̄, point)                              // O(log N)
        arr.append(point)
    end for
    S[query] = arr
end for
```

Algorithm A5 (PointWeb) Inference: $\mathcal{O}(MNK)$

```
Number of training points: N
Number of query points: M
Number of neighbors to search: K
for query = 1, 2, ···, M do
    Best score buffer: b[K]
    for point = 1, 2, ···, N do
        for k = 1, 2, ···, K do
            if d(query, point) < b[k] then
                for i = K − 1, ···, k + 1 do
                    b[i] = b[i − 1]
                end for
                b[k] = d(query, point)
            end if
        end for
    end for
end for
```

Algorithm A6 (PAConv) Inference: $\mathcal{O}(MN \log K)$

```
Number of training points: N
Number of query points: M
Number of neighbors to search: K
for query = 1, 2, ···, M do
    H = InitHeap()                                     // O(K)
    MinD = 10^10
    MinIdx = 0
    for point = 1, 2, ···, N do
        if d(point, query) < MinD then
            Reheap(H, MinD, MinIdx, K)                 // O(log K)
            MinD = d(point, query)
            MinIdx = point
        end if
    end for
    Heapsort(H, MinIdx, MinD, K)                       // O(K log K)
end for
```

Table A9. **Detailed experimental results on S3DIS [1] Area 5 test.** Note that the latency of each method denotes the per-scene wall-time latency normalized by that of Fast Point Transformer. Numbers except the latency means percentage values (%). We denote MinkowskiNet42† and Fast Point Transformer as MinkNet42† and FastPointTrans., respectively.

Method	Latency	mAcc	mIoU	ceil.	floor	wall	beam	col.	wind.	door	table	chair	sofa	book.	board	clut.
PointNet [7]	129.71	49.0	41.1	88.8	97.3	69.8	0.1	3.9	46.3	10.8	59.0	52.6	5.9	40.3	26.4	33.2
SPGraph [5]	130.57	66.5	58.0	89.4	96.9	78.1	0.0	42.8	48.9	61.6	84.7	75.4	69.8	52.6	2.1	52.2
PointWeb [11]	83.00	66.6	60.3	92.0	98.5	79.4	0.0	21.1	59.7	34.8	76.3	88.3	46.9	69.3	64.9	52.5
KPConv *deform* [9]	751.07	72.8	67.1	92.8	97.3	82.4	0.0	23.9	58.0	69.0	81.5	**91.0**	**75.4**	75.3	66.7	58.9
PAConv [10]	200.93	73.0	66.6	**94.6**	**98.6**	82.4	0.0	26.4	58.0	60.0	80.4	89.7	69.8	74.3	73.5	57.7
PointTransformer [12]	129.07	76.5	**70.4**	94.0	98.5	**86.3**	0.0	38.0	63.4	74.3	**89.1**	82.4	74.3	**80.2**	76.0	59.3
MinkNet42† (5cm)	**0.50**	73.3	66.0	93.2	97.0	84.0	0.0	25.7	63.9	66.4	76.9	88.9	58.4	70.1	78.0	54.9
+ rotation average	4.07	73.5	67.1	93.9	97.1	85.2	0.1	28.3	**64.5**	70.3	76.8	90.0	57.2	70.9	**81.1**	56.7
FastPointTrans. (5cm)	0.93	74.7	67.5	91.5	97.4	86.0	0.2	40.4	60.8	66.7	79.6	87.7	58.6	73.7	77.2	57.3
+ rotation average	7.50	75.5	68.5	90.0	96.0	86.2	0.0	47.1	61.3	69.7	81.1	88.2	60.9	74.2	78.2	57.3
MinkNet42† (4cm)	0.57	73.6	67.2	93.1	97.6	84.9	0.0	35.9	57.5	74.5	80.0	88.2	55.6	72.9	77.1	56.9
+ rotation average	4.71	74.3	68.3	93.8	97.6	85.9	0.0	38.9	58.8	75.3	81.1	88.8	53.3	74.6	80.0	**59.8**
FastPointTrans. (4cm)	1.00	77.1	68.7	93.8	97.8	85.5	**0.6**	49.9	60.5	72.9	80.2	88.7	56.0	71.4	78.0	58.1
+ rotation average	8.07	**77.9**	70.3	94.2	98.0	86.0	0.2	**53.8**	61.2	**77.3**	81.3	89.4	60.1	72.8	80.4	58.9

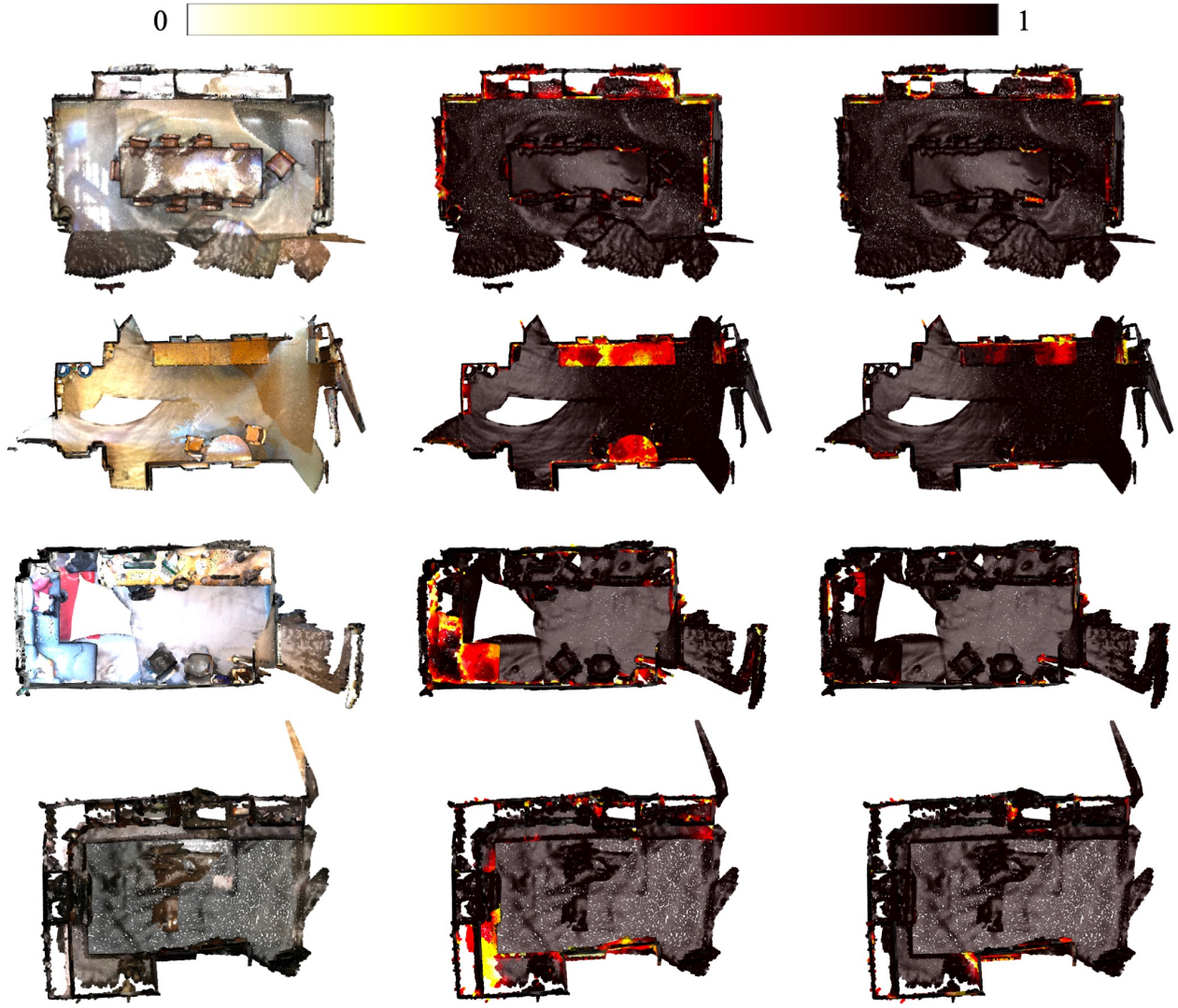

Figure A2. **Qualitative results of consistency scores** (CScore) **on ScanNet [2].** (*Left*) Input point cloud, (*Middle*) CScore of MinkowskiNet42†, and (*Right*) CScore of the proposed Fast Point Transformer. Both models are trained with voxel size as 10cm.

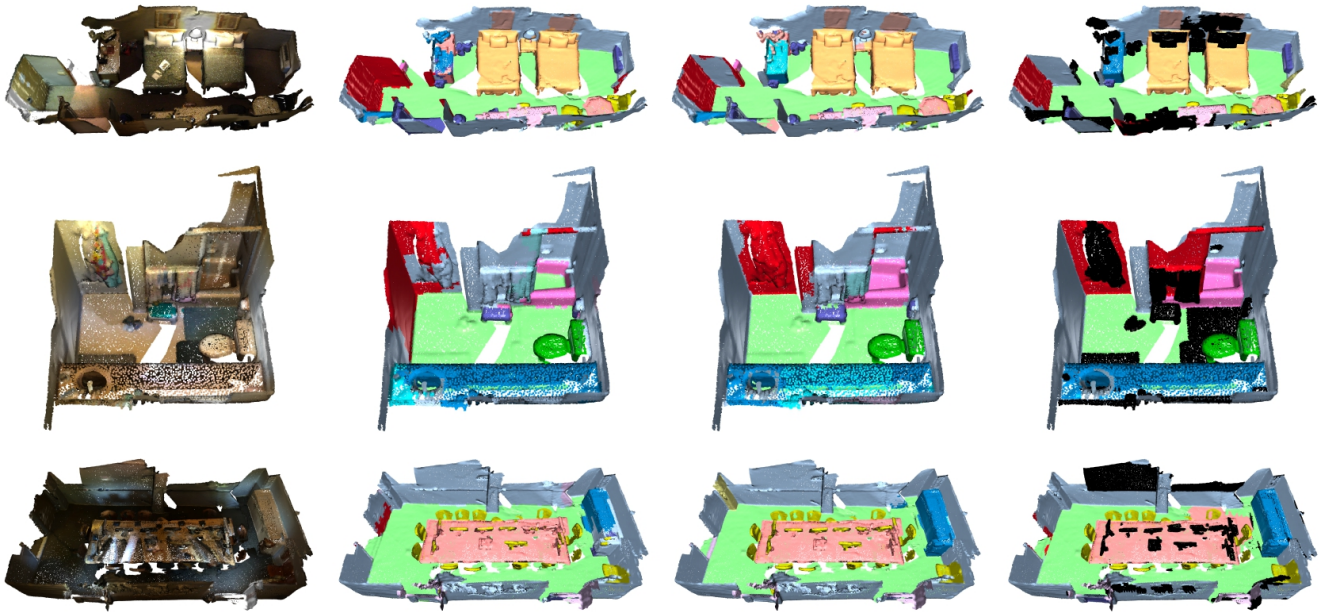

Figure A3. **Qualitative results of 3D semantic segmentation on ScanNet [2].** (*First column*) Input point cloud, (*Second column*) Predicted semantic labels by MinkowskiNet42†, (*Third column*) Predicted semantic labels by the proposed Fast Point Transformer, and (*Fourth column*) Ground truth. Both models are trained with voxel size as 10cm.

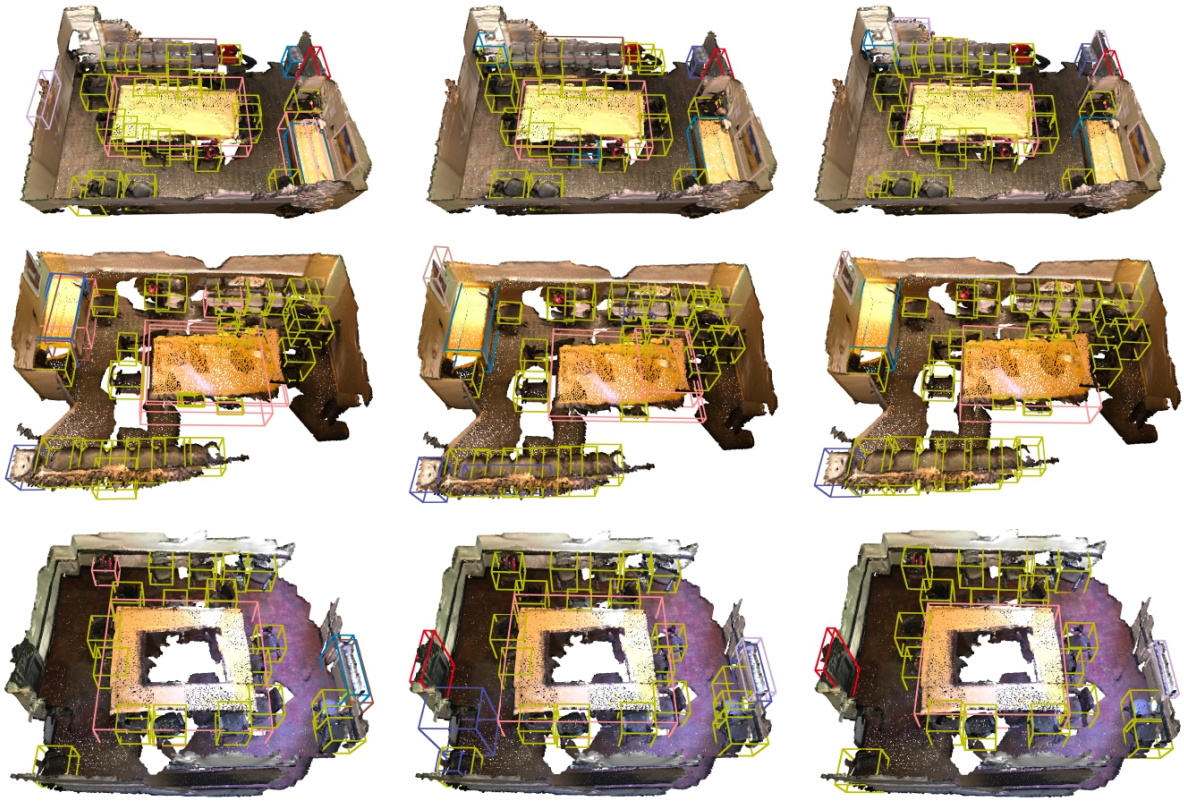

Figure A4. **Qualitative results of 3D object detection on ScanNet [2].** (*Left*) Predicted bounding boxes by VoteNet [6] with MinkowskiNet backbone, (*Middle*) Predicted bounding boxes by VoteNet [6] with the proposed Fast Point Transformer backbone, and (*Right*) Ground truth.

6. Notations

$\mathcal{P}^{\text{in}} = \{(\mathbf{p}_n, \mathbf{i}_n)\}$	Input point cloud
$\mathbf{p}_n \in \mathbb{R}^3$	The n-th point coordinate
$\mathbf{i}_n \in \mathbb{R}^{D_{\text{in}}}$	The n-th input point feature
$\mathcal{P}^{\text{out}} = \{(\mathbf{p}_n, \mathbf{o}_n)\}$	Output point cloud
$\mathbf{o}_n \in \mathbb{R}^{D_{\text{out}}}$	The n-th point feature
$\mathcal{V} = \{(\mathbf{v}_i, \mathbf{f}_i, \mathbf{c}_i)\}$	Input voxels with centroids
$\mathbf{v}_i \in \mathbb{R}^3$	The i-th voxel center coordinate
$\mathbf{f}_i \in \mathbb{R}^{D_{\text{in}}}$	The i-th input voxel feature
$\mathbf{c}_i \in \mathbb{R}^3$	The i-th voxel centroid coordinate
$\mathcal{M}(i)$	A set of point indices within the i-th voxel
Ω	A permutation-invariant operator (*e.g.*, average)
$\mathcal{V}' = \{(\mathbf{v}_i, \mathbf{f}'_i, \mathbf{c}_i)\}$	Output voxels with centroids
$\mathbf{f}'_i \in \mathbb{R}^{D_{\text{out}}}$	The i-th output voxel feature
$\mathcal{N}(i)$	A set of neighbor voxel indices the i-th voxel
$\mathbf{e}_n$	The centroid-to-point positional encoding
δ_{enc}	An encoding layer used in centroid-to-point positional encoding
$\mathbf{o}_n$	The n-th output point feature of the output point cloud $\mathcal{P}^{\text{out}}$
$\oplus$	A vector concatenation operation
$a(\cdot)$	An attention operation
ϕ	A query projection layer in attention operations
ψ	A value projection layer in attention operations
$\mathbf{g}_i$	A centroid-aware voxel feature
δ_{rel}	A discretized positional encoding layer
δ_{abs}	A continuous positional encoding layer

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*, pages 1534–1543, 2016. 1, 2, 3, 5

[2] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, pages 3075–3084, 2019. 1, 2, 3, 4, 5, 6

[3] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, pages 5828–5839, 2017. 1, 3

[4] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *ICCV*, pages 3464–3473, 2019. 1

[5] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, pages 4558–4567, 2018. 3, 4, 5

[6] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *ICCV*, pages 9277–9286, 2019. 6

[7] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017. 3, 4, 5

[8] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *ECCV*, pages 685–702. Springer, 2020. 3

[9] 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*, pages 6411–6420, 2019. 3, 4, 5

[10] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *CVPR*, pages 3173–3182, June 2021. 3, 4, 5

[11] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *CVPR*, pages 5565–5573, 2019. 3, 4, 5

[12] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H.S. Torr, and Vladlen Koltun. Point transformer. In *ICCV*, pages 16259–16268, October 2021. 3, 4, 5