A Unified Query-based Paradigm for Point Cloud Understanding

Supplementary Material

1. Contextual Relative Positional Encoding

In this section, we elaborate the computation process of relative positional encoding $B_{\mathbf{qk}} \in \mathbb{R}^{m \times n}$ in the attention weights and $B_{\mathbf{v}} \in \mathbb{R}^{m \times n \times d}$ in the value vectors.

The relative positional encoding $B_{\mathbf{qk}}^{(ij)}$ between the i-th target and the j-th source is computed as:

$$B_{\mathbf{qk}}^{(ij)} = (F_Y^{(i)} W_{\mathbf{q}})(B_{\mathbf{q}}^{(ij)})^T + (F_X^{(j)} W_{\mathbf{k}})(B_{\mathbf{k}}^{(ij)})^T. \quad (1)$$

$B_{\mathbf{q}}$, $B_{\mathbf{k}}$ and $B_{\mathbf{v}}$ are all positional embeddings in $\mathbb{R}^{m \times n \times d}$ obtained by the relative positional difference between targets Y and sources X. These three embeddings share the same computation procedure. Therefore, we use B to denote all of them to show how they are constructed.

Before introducing the computation of relative positional embedding B, we first define three learnable embedding tables, $E_x \in \mathbb{R}^{T_x \times \lfloor \frac{d}{3} \rfloor}$, $E_y \in \mathbb{R}^{T_y \times \lfloor \frac{d}{3} \rfloor}$ and $E_z \in \mathbb{R}^{T_z \times \lfloor \frac{d}{3} \rfloor}$, for transferring the positional difference in a certain axis to a high-dimensional embedding. T_x, T_y and T_z are the lengths of the tables, which relate to the maximum ranges of the 3D scene in x-, y- and z-axis, respectively. We use R_x, R_y and R_z to denote the maximum ranges in the three axes. The computation of T_k from R_k, in which $k \in \{x, y, z\}$, is formulated as:

$$T_k = \lfloor \frac{R_k \times 2}{t} \rfloor, \; k \in \{x, y, z\}, \quad (2)$$

where t is a pre-defined value for quantification. We double R_k here since the relative positional difference between any two points within the 3D scene along k-axis is in the range of $[-R_k, R_k]$.

Then, we compute the relative positional embeddings for concrete targets Y and sources X. We compute and quantify their relative Euclidean positional difference to obtain a difference matrix $D_k \in \mathbb{R}^{m \times n}$ ($k \in \{x, y, z\}$) for each axis. For example, given the i-th target $Y^{(i)}$ and the j-th source $X^{(j)}$, $D_k^{(ij)}$ is computed as:

$$D_k^{(ij)} = \lfloor \frac{X_k^{(j)} - Y_k^{(i)} + R_k}{t} \rfloor, \; k \in \{x, y, z\}, \quad (3)$$

where $X_k \in \mathbb{R}^n$ and $Y_k \in \mathbb{R}^m$ represent the k-axis coordinates of source and target positions. We add R_k to the coordinate difference to ensure that the difference values in D_k are in the range of $[0, T_k]$. We finally construct the high-dimensional relative positional encoding matrix B by looking up embeddings from the three tables based on D_k ($k \in \{x, y, z\}$) and concatenating the results:

$$B = [E_x^{(D_x)}, E_y^{(D_y)}, E_z^{(D_z)}]. \quad (4)$$

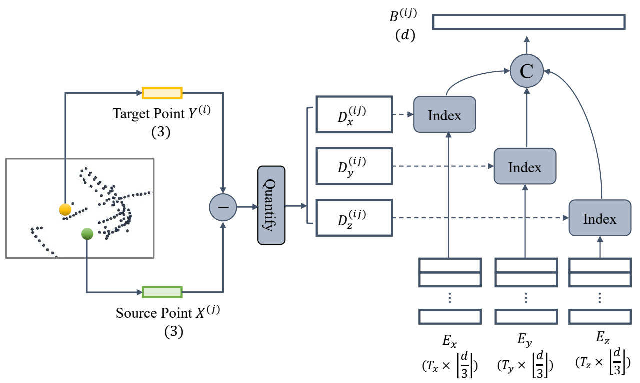

Figure 1. Illustration of the contextual relative positional encoding.

We also illustrate this computation process in Figure 1. Note that we keep different learnable tables E_k ($k \in \{x, y, z\}$) for $\mathbf{q}$, $\mathbf{k}$ and $\mathbf{v}$.

To avoid saving the large relative positional embedding matrix $B \in \mathbb{R}^{m \times n \times d}$, we re-implement the attention layer with a series of CUDA kernel functions which directly returns the attention results without storing large intermediate tensors.

2. Implementation Detail

2.1. Semantic Segmentation

For semantic segmentation, we adopt the encoder network of the voxel-based sparse U-Net [1, 4] as our embedding stage network to extract support features. The sparse U-Net is a strong baseline network in point cloud semantic segmentation, where each level of the encoder consists of a strided sparse convolution layer to downsample the input scenes and several residual blocks. A residual block is composed of two submanifold sparse convolutions and a residual addition with the input features to the block. Our

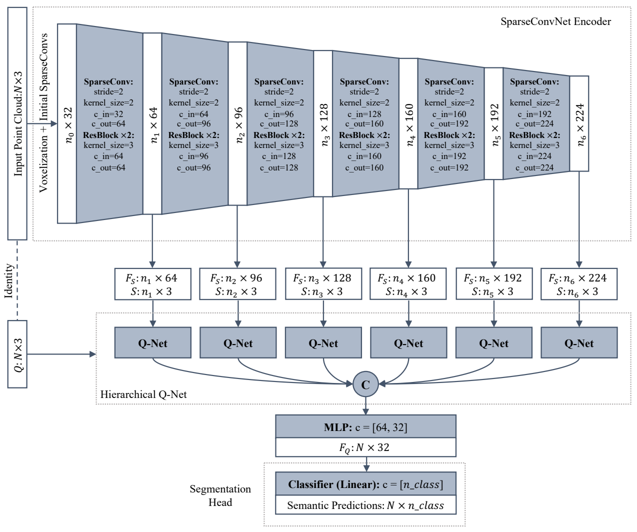
SparseConvNet Encoder
Input Point Cloud: N×3
Voxelization + Initial SparseConvs
n_0 × 32
SparseConv: stride=2 kernel_size=2 c_in=32 c_out=64 ResBlock ×2: kernel_size=3 c_in=64 c_out=64
n_1 × 64
SparseConv: stride=2 kernel_size=2 c_in=64 c_out=96 ResBlock ×2: kernel_size=3 c_in=96 c_out=96
n_2 × 96
SparseConv: stride=2 kernel_size=2 c_in=96 c_out=128 ResBlock ×2: kernel_size=3 c_in=128 c_out=128
n_3 × 128
SparseConv: stride=2 kernel_size=2 c_in=128 c_out=160 ResBlock ×2: kernel_size=3 c_in=160 c_out=160
n_4 × 160
SparseConv: stride=2 kernel_size=2 c_in=160 c_out=192 ResBlock ×2: kernel_size=3 c_in=192 c_out=192
n_5 × 192
SparseConv: stride=2 kernel_size=2 c_in=192 c_out=224 ResBlock ×2: kernel_size=3 c_in=224 c_out=224
n_6 × 224
F_S: n_1 × 64 S: n_1 × 3
F_S: n_2 × 96 S: n_2 × 3
F_S: n_3 × 128 S: n_3 × 3
F_S: n_4 × 160 S: n_4 × 3
F_S: n_5 × 192 S: n_5 × 3
F_S: n_6 × 224 S: n_6 × 3
Identity
Q: N×3
Q-Net
C
Hierarchical Q-Net
MLP: c = [64, 32]
F_Q: N × 32
Segmentation Head
Classifier (Linear): c = [n_class]
Semantic Predictions: N × n_class

Figure 2. The architecture of EQ-Net for point cloud semantic segmentation.

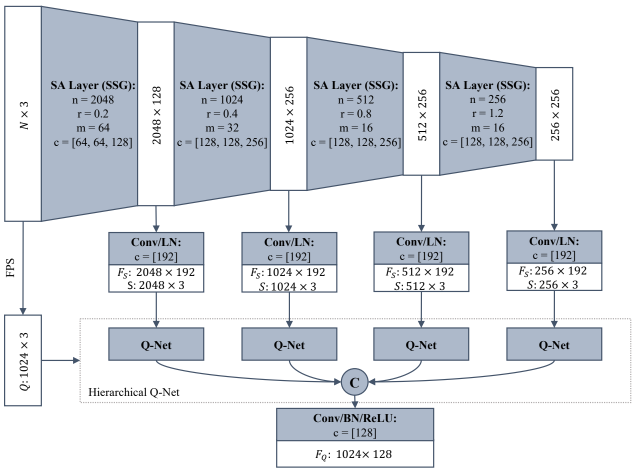
N × 3
SA Layer (SSG): n = 2048 r = 0.2 m = 64 c = [64, 64, 128]
2048 × 128
SA Layer (SSG): n = 1024 r = 0.4 m = 32 c = [128, 128, 256]
1024 × 256
SA Layer (SSG): n = 512 r = 0.8 m = 16 c = [128, 128, 256]
512 × 256
SA Layer (SSG): n = 256 r = 1.2 m = 16 c = [128, 128, 256]
256 × 256
FPS
Conv/LN: c = [192] F_S: 2048 × 192 S: 2048 × 3
Conv/LN: c = [192] F_S: 1024 × 192 S: 1024 × 3
Conv/LN: c = [192] F_S: 512 × 192 S: 512 × 3
Conv/LN: c = [192] F_S: 256 × 192 S: 256 × 3
Q: 1024 × 3
Q-Net
C
Hierarchical Q-Net
Conv/BN/ReLU: c = [128]
F_Q: 1024× 128

Figure 3. The architecture of EQ-PointNet++ used in indoor object detectors.

Q-Net then makes use of the multi-level features to produce Q-representation for query positions. To be specific, in each level, the Q-Net consists of three Q-Blocks. The K for the local attention is set to 32. The maximum range of a 3D scene is set to 20m. For the relative positional encoding, the quantification value t is set to 0.005. Figure 2 shows the detailed structure of our EQ-Net for point cloud semantic segmentation, where the sparse convolution layers and residual blocks are denoted as "SparseConv" and "ResBlock", respectively.

For both ScanNetV2 and S3DIS, we train our EQ-Net with the AdamW optimizer and adjust the learning rate following the OneCycle [8] policy, where the maximum and initial learning rates are set to 1e-3 and 1e-5, respectively. The weight decay is set to 0.1. We train for 600 epochs for ScanNetV2 and 1,500 epochs for S3DIS with a batch size of 8. The augmentations for the input point cloud include random flip, random rotation, random jittering and elastic, following [4]. For voxelizing the point cloud in the embedding stage, we set the voxel size to 2cm.

2.2. Indoor Object Detection

We base our implementation on the MMDetection3D codebase [2] with a version of 0.17.0. For ScanNetV2, we train our EQ-Paradigm VoteNet and GroupFree by AdamW optimizer for totally 80 epochs with an initial learning rate of 0.006 and 0.003, respectively. The training batch size is 16, and the learning rate is decayed at the 56th and the 68th epochs with a rate of 0.1. We set the weight decay for training VoteNet and GroupFree to 0.01 and 0.0005, respectively. For SUN RGB-D, we train our method for 48 epochs with an initial learning rate and a weight decay of 0.001 and 0.1, respectively. The learning rate is decayed at the 36th and the 42th epochs with a rate of 0.1. For all models, we apply random rotation and random flip as our data augmentation strategies.

Each Q-Net consists of three Q-Blocks for updating query and support features. For each Q-Block, we set the K in the local attention module to 64, the maximum range of the 3D scenes to 10m for all three axes, and the quantification value t for relative positional encoding to 0.01. The specific architecture of EQ-PointNet++ in VoteNet and GroupFree is illustrated in Figure 3, where the set abstraction layers with a single-scale grouping setting is described as "SA Layer (SSG)".

2.3. Outdoor Object Detection

We base our implementation on the OpenPCDet codebase [9] with a version of 0.3.0. All networks are trained with the same schedule as their baselines provided by the official OpenPCDet team. The Q-Net for outdoor detectors consists of three Q-Blocks. For each Q-Block, we set the K to 128, the maximum ranges to 70.4m, 80m and 4m for x, y and z axis, respectively, and the quantification value t to 0.1. Ground-truth sampling, random scaling and random rotation are utilized as data augmentation strategies.

EQ-SECOND SECOND is a voxel-based detector dealing with voxelized point cloud. In the encoder, It applies a SparseConvNet composed of some SparseConvBlocks to extract sparse voxel features. Each SparseConvBlock consists of a strided sparse convolution layer (SparseConv) and two submanifold sparse convolution layers (Subm.Conv) to downsample the input scene and extract high-level features. In the decoder, it utilizes a MapToBEV module to convert the sparse voxel features to the BEV image representation by merging voxel features with different heights. Then, a

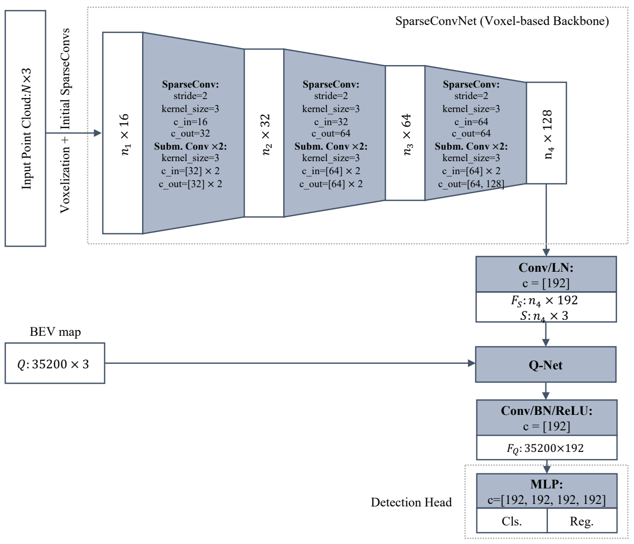
Input Point Cloud: $N \times 3$
Voxelization + Initial SparseConvs
SparseConvNet (Voxel-based Backbone)
$n_1 \times 16$
SparseConv: stride=2 kernel_size=3 c_in=16 c_out=32 Subm. Conv ×2: kernel_size=3 c_in=[32] × 2 c_out=[32] × 2
$n_2 \times 32$
SparseConv: stride=2 kernel_size=3 c_in=32 c_out=64 Subm. Conv ×2: kernel_size=3 c_in=[64] × 2 c_out=[64] × 2
$n_3 \times 64$
SparseConv: stride=2 kernel_size=3 c_in=64 c_out=64 Subm. Conv ×2: kernel_size=3 c_in=[64] × 2 c_out=[64, 128]
$n_4 \times 128$
Conv/LN: c = [192]
F_S: $n_4 \times 192$
S: $n_4 \times 3$
BEV map
Q: 35200×3
Q-Net
Conv/BN/ReLU: c = [192]
F_Q: 35200×192
Detection Head
MLP: c=[192, 192, 192, 192]
Cls.
Reg.

Figure 4. The architecture of EQ-SECOND.

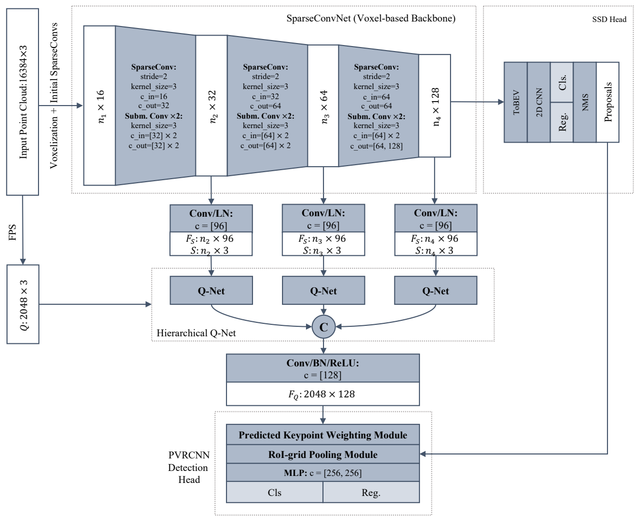
Input Point Cloud: 16384×3
Voxelization + Initial SparseConvs
SparseConvNet (Voxel-based Backbone)
$n_1 \times 16$
SparseConv: stride=2 kernel_size=3 c_in=16 c_out=32 Subm. Conv ×2: kernel_size=3 c_in=[32] × 2 c_out=[32] × 2
$n_2 \times 32$
SparseConv: stride=2 kernel_size=3 c_in=32 c_out=64 Subm. Conv ×2: kernel_size=3 c_in=[64] × 2 c_out=[64] × 2
$n_3 \times 64$
SparseConv: stride=2 kernel_size=3 c_in=64 c_out=64 Subm. Conv ×2: kernel_size=3 c_in=[64] × 2 c_out=[64, 128]
$n_4 \times 128$
SSD Head
ToBEV
2D CNN
Cls.
Reg.
NMS
Proposals
FPS
Q: 2048 × 3
Conv/LN: c = [96] F_S: $n_2 \times 96$ S: $n_2 \times 3$
Conv/LN: c = [96] F_S: $n_3 \times 96$ S: $n_3 \times 3$
Conv/LN: c = [96] F_S: $n_4 \times 96$ S: $n_4 \times 3$
Q-Net
Q-Net
Q-Net
C
Hierarchical Q-Net
Conv/BN/ReLU: c = [128]
F_Q: 2048 × 128
PVRCNN Detection Head
Predicted Keypoint Weighting Module
RoI-grid Pooling Module
MLP: c = [256, 256]
Cls
Reg.

Figure 6. The concrete architecture of EQ-PVRCNN[§].

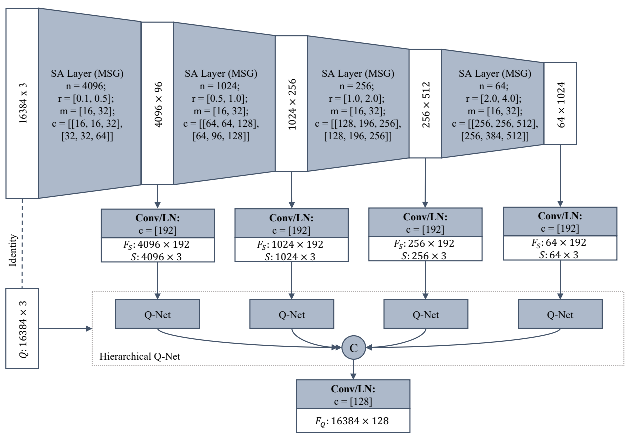
16384 × 3
SA Layer (MSG) n = 4096; r = [0.1, 0.5]; m = [16, 32]; c = [[16, 16, 32], [32, 32, 64]]
4096 × 96
SA Layer (MSG) n = 1024; r = [0.5, 1.0]; m = [16, 32]; c = [[64, 64, 128], [64, 96, 128]]
1024 × 256
SA Layer (MSG) n = 256; r = [1.0, 2.0]; m = [16, 32]; c = [[128, 196, 256], [128, 196, 256]]
256 × 512
SA Layer (MSG) n = 64; r = [2.0, 4.0]; m = [16, 32]; c = [[256, 256, 512], [256, 384, 512]]
64 × 1024
Identity
Conv/LN: c = [192] F_S: 4096 × 192 S: 4096 × 3
Conv/LN: c = [192] F_S: 1024 × 192 S: 1024 × 3
Conv/LN: c = [192] F_S: 256 × 192 S: 256 × 3
Conv/LN: c = [192] F_S: 64 × 192 S: 64 × 3
Q: 16384 × 3
Q-Net
Q-Net
Q-Net
Q-Net
C
Hierarchical Q-Net
Conv/LN: c = [128]
F_Q: 16384 × 128

Figure 5. The architecture of EQ-PointNet++ for EQ-PointRCNN.

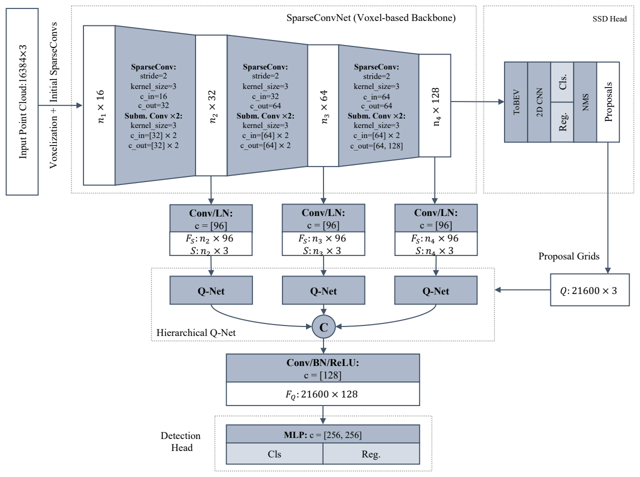
Input Point Cloud: 16384×3
Voxelization + Initial SparseConvs
SparseConvNet (Voxel-based Backbone)
$n_1 \times 16$
SparseConv: stride=2 kernel_size=3 c_in=16 c_out=32 Subm. Conv ×2: kernel_size=3 c_in=[32] × 2 c_out=[32] × 2
$n_2 \times 32$
SparseConv: stride=2 kernel_size=3 c_in=32 c_out=64 Subm. Conv ×2: kernel_size=3 c_in=[64] × 2 c_out=[64] × 2
$n_3 \times 64$
SparseConv: stride=2 kernel_size=3 c_in=64 c_out=64 Subm. Conv ×2: kernel_size=3 c_in=[64] × 2 c_out=[64, 128]
$n_4 \times 128$
SSD Head
ToBEV
2D CNN
Cls.
Reg.
NMS
Proposals
Proposal Grids
Q: 21600 × 3
Conv/LN: c = [96] F_S: $n_2 \times 96$ S: $n_2 \times 3$
Conv/LN: c = [96] F_S: $n_3 \times 96$ S: $n_3 \times 3$
Conv/LN: c = [96] F_S: $n_4 \times 96$ S: $n_4 \times 3$
Q-Net
Q-Net
Q-Net
C
Hierarchical Q-Net
Conv/BN/ReLU: c = [128]
F_Q: 21600 × 128
Detection Head
MLP: c = [256, 256]
Cls
Reg.

Figure 7. The architecture of EQ-PVRCNN[†].

2D CNN is followed to propagate non-empty pixel features to empty pixels within the BEV map. An SSD head is finally applied on the dense BEV features to perform 3D object detection.

EQ-SECOND keeps the SparseConvNet as its embedding stage network, but utilizes a Q-Net to extract dense BEV feature maps. The Q-Net treats positions and features of non-empty voxels in the last embedding layer as support points and support features, respectively, and takes coordinates of all pixels within the final BEV map as query positions. Since the resolution of the final BEV map is 200×176 in SECOND, the number of query positions is 35200. After the querying stage, the dense BEV features are sent to the SSD head for prediction. The architecture of EQ-SECOND is illustrated in Figure 4.

EQ-PointRCNN PointRCNN is a point-based detector, which utilizes PointNet++ to extract sparse point features followed by a specifically designed point-based detection head. The head deploys a bottom-up proposal generation module and a canonical bounding box refinement module to generate predictions. EQ-PointRCNN keeps the head of PointRCNN and applies the EQ-PointNet++ as its point feature extraction network. The architecture of EQ-PointNet++ in EQ-PointRCNN is illustrated in Figure 5, where the set abstraction layers with a multi-scale grouping setting is described as "SA Layer (MSG)".

EQ-PVRCNN PVRCNN is the state-of-the-art outdoor detector. It applies SECOND to generate high-quality proposals, followed by a decoder utilizing the voxel-set abstraction modules to extract keypoint features. Finally, a

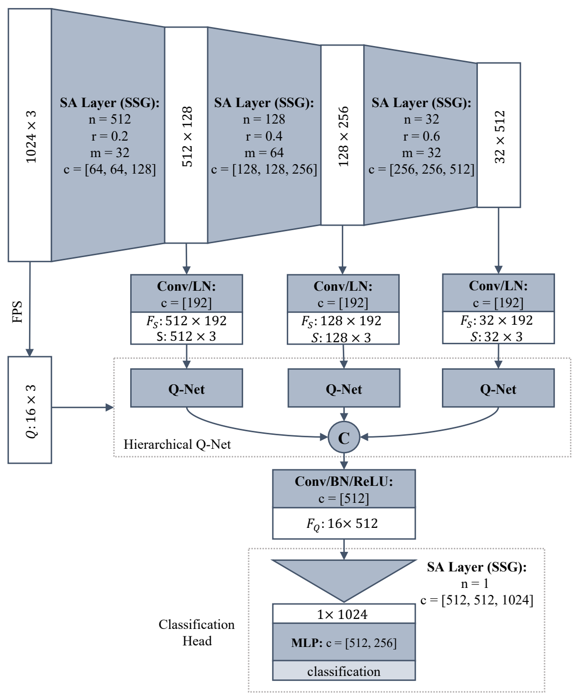

Figure 8. The architecture of EQ-PointNet++ for shape classification.

head including a predicted keypoint weighting module and a RoI-grid pooling module to convert keypoint features to proposal grid features is adopted for proposal refinement. In this paper, we propose two types of EQ-PVRCNN: EQ-PVRCNN§ and EQ-PVRCNN†.

EQ-PVRCNN§ follows the original design of PVRCNN. After obtaining proposals from SECOND, it deploys hierarchical Q-Net as the querying stage network treating keypoints as query positions to extract keypoint features, followed by the original PVRCNN head to convert keypoint features to proposal grid features for detection. The architecture of EQ-PVRCNN§ is illustrated in Figure 6.

In EQ-PVRCNN†, we directly set the proposal grid points as query positions, obtain their features, and apply a simple head composed of a classification layer and a regression layer to generate final predictions. EQ-PVRCNN† is a concise design getting rid of the complicated modules in PVRCNN head (*e.g.*, voxel-set abstraction, predicted keypoint weighting and RoI-grid Pooling) and also achieves impressive "AP" results. In EQ-PVRCNN†, 100 proposals are kept after SECOND and each proposal is equally divided into $6 \times 6 \times 6$ grid points. Therefore, the number of query positions of EQ-PVRCNN† is 21600. The architecture of EQ-PVRCNN† is illustrated in Figure 7.

No. of Q-Blocks L	Car (%)	Pedestrian (%)	Cyclist (%)
1	78.86	48.13	63.24
2	80.94	50.13	65.11
3	81.49	**53.64**	**67.13**
6	**83.10**	52.35	66.81

Table 1. AP comparison on class "Car" for applying different numbers of Q-Blocks in EQ-SECOND.

No. of Q-Blocks L	1	2	3	4
mIoU (%)	73.5	74.6	**75.3**	75.1

Table 2. Semantic segmentation results of our EQ-Net with different numbers of Q-Blocks on ScanNetV2 validation set.

Pos. Enc.	no pos.	abs. pos.	bias RPE	cont. RPE
AP (%)	0.01	80.14	81.03	**81.49**

Table 3. AP comparison on class "Car" for applying different positional encoding methods in EQ-SECOND. "no pos.": no positional encoding; "abs. pos.": absolute positional encoding following [6, 7]; "bias RPE": bias-mode relative positional encoding in [10]; "cont. RPE": contextual relative positional encoding.

2.4. Shape Classification

We base our implementation on pytorch-version PointNet++ codebase [11]. We train our EQ-PointNet++ with an initial learning rate of 0.001 by Adam optimizer for totally 200 epochs. The batch size is 24 and the learning rate is decayed at each 20 epochs with a rate of 0.7. Random scaling and random translation are applied as our data augmentation strategies. We apply hierarchical Q-Net to extract multi-level query features. Each Q-Net consists of three Q-Blocks. For each Q-Block, we set the K to 32, the maximum range to 5m for all three axes, and the quantification value t to 0.005. The architecture of EQ-PointNet++ for shape classification is illustrated in Figure 8.

3. More Ablation Studies

All ablation studies are conducted on EQ-SECOND, EQ-VoteNet and EQ-Net for outdoor object detection, indoor object detection and semantic segmentation, respectively. EQ-SECOND is evaluated on the KITTI dataset with "AP" calculated on instances labeled as "Moderate" difficulty; EQ-VoteNet is evaluated on the ScanNetV2 dataset with mAP@0.5; EQ-Net is tested on the ScanNetV2 dataset with mIoU.

Analysis on the Number of Q-Blocks In Table 1 and Table 2, we list the performance of EQ-SECOND and EQ-Net with different number L of Q-Blocks for detection and segmentation, respectively. As illustrated in Table 1, for object detection, more Q-Blocks bring a consistent improvement on the performance of large objects ("Car").

K	EQ-SECOND (%)	EQ-VoteNet (%)	EQ-Net (%)
16	78.11	41.9	74.5
32	81.09	43.5	**75.3**
64	81.45	**45.4**	75.2
128	**81.49**	44.6	-
256	81.46	-	-
GA	81.48	-	-

Table 4. Performance comparison of different methods with various K in local attention. "GA" means using global attention.

This sustained improvement comes from our design of updating support features and query features simultaneously in each Q-Block. With more Q-Blocks, the support features can embed more long-range semantic information and bring more meaningful guidance for query feature refinement. However, more global information cannot continuously benefit detecting small objects ("Pedestrian" and "Cyclist"), which may introduce excessive noise and thus is harmful to small object detection. As illustrated in Table 1, compared to utilizing three Q-Blocks, the "AP" of using six Q-Blocks improves a lot on class "Car" but is lower on class "Pedestrian" and "Cyclist". For semantic segmentation, we also observe in Table 2 that increasing the number of Q-blocks brings mIoU improvement but the performance growth stops when the number of blocks is larger than three. Based on both Table 1 and Table 2, we finally uniformly adopt three Q-Blocks in the Q-Net of all models.

Analysis on the Relative Positional Encoding Positional encoding plays a fundamental role in our Q-Net. Especially in the first block, it is the primary hint to extract features for query positions. In Table 3, we compare EQ-SECOND models with different positional encoding methods. As illustrated in the table, without positional encoding, the model cannot converge, leading to inferior AP results. Meanwhile, applying contextual relative positional encoding yields the best performance. This demonstrates that correlating the relative position with $\mathbf{q}$, $\mathbf{k}$ and $\mathbf{v}$ features in an attention layer benefits the modeling of different responses for points in objects with various scales and shapes, thus enabling more effective query feature generation.

Analysis on the K of Local Attention For each Q-Block, we apply local attention instead of global attention to reduce the GPU memory cost of attention weight $\mathcal{A}$. In Table 4, we show the performance of different methods with various K in their local attention mechanism. Meanwhile, we also show the performance of EQ-SECOND with global attention. Since the numbers of query positions and support points in EQ-SECOND (*i.e.*, 30k and 10k) are large, during the training and testing of the global attention model, we randomly choose 2k query positions in each iteration to produce losses or predictions. As illustrated, applying local attention achieves comparable performance with global attention yet with much less GPU memory cost.

4. More Quantitative Results

In this section, we provide class-wise quantitative results on semantic segmentation and indoor object detection. For point cloud semantic segmentation, the evaluation metrics include mean Intersection-over-Union (mIoU), mean Accuracy (mAcc) and Overall Accuracy (OA). In Table 5, we show the detailed segmentation results on S3DIS dataset with two different evaluation modes, Area 5 and 6-fold cross validation. In Table 6, we show the class-wise IoU for semantic segmentation on ScanNetV2 validation and test sets. In Table 7 and Table 8, we show the AP results of different methods with EQ-PointNet++ on ScanNetV2 dataset with IoU threshold of 0.25 and 0.5, respectively. In Table 9, we show the AP@0.25 and AP@0.5 of VoteNet with EQ-PointNet++ on SUN RGB-D dataset.

5. Limitation

Through EQ-Paradigm, we can easily combine different backbones and heads, which provides great flexibility in 3D model design. However, the choice of embedding stage network is still highly-dependent on the application scenarios. For example, due to the diverse scopes and point distributions on indoor and outdoor scenes, the networks designed for indoor scenario usually have smaller receptive radius to focus on object details, while the outdoor networks emphasize more on the object relations. Hence, the network specifically designed for indoor scenes usually performs bad on outdoor point cloud. To deploy a EQ-Paradigm model on a particular application scenario, we need to carefully choose a practical embedding stage network for extracting support features. To enable a universal structure for all the 3D scenes, like the ResNet [5] and ViT [3] in 2D, more exploration on the embedding stage network design is needed.

Method	Mode	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book.	sofa	board	clut.	mIoU	mAcc	OA
Sparse EQ-Net	Area 5	93.4	98.2	86.3	0.0	38.7	65.8	66.1	92.1	82.1	77.8	76.4	88.6	61.5	71.3	77.2	91.1
	6-fold	94.7	97.5	86.1	71.7	58.4	73.4	76.7	82.3	76.4	71.5	75.1	75.7	67.7	77.5	86.1	91.4

Table 5. Semantic segmentation results of our EQ-Net on S3DIS under different evaluation modes, *i.e.*, Area 5 and 6-fold cross validation. We show class-wise IoU (%) and mIoU (%), together with mAcc(%) and OA(%).

Method	Set	bathtub	bed	bookshelf	cabinet	chair	counter	curtain	desk	door	floor	other furn.	picture	refrigerator	show. curt.	sink	sofa	table	toilet	wall	window	mIoU
Sparse EQ-Net	Val.	87.7	81.2	82.6	69.2	90.3	68.5	76.9	75.3	70.2	95.2	64.0	34.0	62.7	70.9	67.0	81.8	79.7	93.5	86.4	69.4	75.3
	Test	62.0	79.9	84.9	73.0	82.2	49.3	89.7	66.4	68.1	95.5	56.2	37.8	76.0	90.3	73.8	80.1	67.3	90.7	87.7	74.5	74.3

Table 6. Semantic segmentation results of our EQ-Net on ScanNetV2 validation and test sets. We show class-wise IoU (%) and mIoU (%).

Method	cabinet	bed	chair	sofa	table	door	window	bookshelf	picture	counter	desk	curtain	refrigerator	showercurtrain	toilet	sink	bathtub	garbagebin	mAP
VoteNet	50.0	87.6	89.6	87.8	69.1	54.9	43.3	43.7	16.2	62.7	70.9	52.0	52.7	79.0	95.7	62.0	89.9	49.5	64.3
GroupFree	51.2	87.3	92.0	88.1	68.8	61.8	51.5	50.5	16.9	64.3	79.7	63.3	56.6	79.0	99.6	61.6	93.9	56.9	68.0

Table 7. Performance of different methods with EQ-PointNet++ on ScanNetV2 dataset. Evaluation metric is AP with 0.25 IoU threshold, AP@0.25 (%).

Method	cabinet	bed	chair	sofa	table	door	window	bookshelf	picture	counter	desk	curtain	refrigerator	showercurtrain	toilet	sink	bathtub	garbagebin	mAP
VoteNet	20.1	77.0	77.0	73.7	52.6	31.9	18.9	37.8	7.4	20.2	36.5	28.4	47.0	40.6	86.8	41.4	89.9	30.3	45.4
GroupFree	21.6	81.5	79.2	70.5	56.9	36.7	24.8	44.1	8.4	34.4	55.5	46.1	39.2	46.0	92.9	39.8	87.0	35.4	50.0

Table 8. Performance of different methods with EQ-PointNet++ on ScanNetV2 dataset. Evaluation metric is AP with 0.5 IoU threshold, AP@0.5 (%).

Method	Metric	bed	table	sofa	chair	toilet	desk	dresser	nightstand	bookshelf	bathtub	mAP
VoteNet	AP@0.25	86.3	49.0	68.0	76.8	88.3	28.0	35.6	64.2	29.8	78.8	60.5
VoteNet	AP@0.5	58.8	21.4	51.7	56.7	61.4	7.4	21.8	44.8	9.2	51.7	38.5

Table 9. Performance of VoteNet with EQ-PointNet++ on SUN RGB-D dataset under different evaluation metrics, AP@0.25 (%) and AP@0.5 (%).

References

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

[2] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. https://github.com/open-mmlab/mmdetection3d, 2020. 2

[3] 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. In *ICLR*, 2021. 5

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

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[6] Ze Liu, Zheng Zhang, Yue Cao, Han Hu, and Xin Tong. Group-free 3d object detection via transformers. *ICCV*, 2021. 4

[7] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3d object detection with pointformer. In *CVPR*, 2021. 4

[8] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, 2019. 2

[9] OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet, 2020. 2

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

[11] Xu Yan. Pointnet++ pytorch. https://github.com/yanx27/Pointnet_Pointnet2_pytorch, 2019. 4