APP-Net: Auxiliary-point-based Push and Pull Operations for Efficient Point Cloud Classification

Tao Lu, Chunxu Liu, Youxin Chen, Gangshan Wu, and Limin Wang

1 State Key Laboratory for Novel Software Technology, Nanjing University, China
{taolu,181240035}@smail.nju.edu.cn {gswu,lmwang}@nju.edu.cn
2 Samsung Electronics (China) R&D Centre, Nanjing, China
yx113.chen@samsung.com

Abstract. Point-cloud-based 3D classification task involves aggregating features from neighbor points. In previous works, each source point is often selected as a neighbor by multiple center points. Thus each source point has to participate in calculation multiple times with high memory consumption. Meanwhile, to pursue higher accuracy, these methods rely on a complex local aggregator to extract fine geometric representation, which slows down the network. To address these issues, we propose a new local aggregator of linear complexity, coined as APP. Specifically, we introduce an auxiliary container as an anchor to exchange features between the source point and the aggregating center. Each source point pushes its feature to only one auxiliary container, and each center point pulls features from only one auxiliary container. This avoids the re-computation of each source point. To facilitate the learning of the local structure, we use an online normal estimation module to provide the explainable geometric information to enhance our APP modeling capability. The constructed network is more efficient than all the previous baselines with a clear margin while only occupying a low memory. Experiments on both synthetic and real datasets verify that APP-Net reaches comparable accuracies with other networks. We will release the complete code to help others reproduce the APP-Net.

Keywords: Point Cloud Classification, Local Aggregator, Efficient

1 Introduction

Classification is the most fundamental task in the computer vision community. With the development of deep learning and the growing demand for 3D applications, how to classify 3D objects with neural networks have become a popular topic in recent years.

In the early stage, some works project the point cloud into 2D image planes from multiple views and then apply the well-designed 2D convolutional neural network to learn representations. Although achieving promising results, they fail to use the geometric information fully. The point cloud is a memory-efficient data form that uses many discrete and disordered points to
record the 3D structure of objects and the environment. Pioneered by PointNet [25], a large number of point-based networks [26,33,30,31,11,15,16,14] are proposed to directly extract the 3D representation to support the classification task. Most of the works are devoted to hitting a higher accuracy metric. So far more than 10 works [11,29,22,23,4,38,7,17,28,39] have claimed that they achieved an accuracy better than 93.5% on the popular benchmark ModelNet40 [37]. And the SOTA under single-pass test is just 94.1% by PointMLP [22]. This indicates that the mainly used benchmark ModelNet40 has been saturated for a long time. The detailed analysis in CloserLook3D [20] also makes clear that, under fair comparison, the performance gap among different local aggregators can be covered by well-designed network architecture. All the evidence shows that pursuing higher accuracy is not that necessary. Thus, we consider how to reduce the computation and memory resources while maintaining comparable accuracy.

Other than PointNet [25], previous point-based methods highly rely on an excellent local extractor to aggregate neighbor context. Point cloud lacks the

![Speed Comparison](image1)

![Memory Comparison](image2)

![Batch Size-Speed](image3)

![Train Speed-Test Speed](image4)

Fig. 1. Quantitative Comparisons. (a) shows the accuracy-speed tradeoff on ModelNet40. We achieve the highest efficiency while maintaining comparable accuracy. (b) and (c) is the GPU memory consumption and inference speed under different inference batch sizes, respectively. In both of them, we outperform other methods with a large margin. (d) provides comprehensive comparisons on the training process and test process. All the experiments in (b) and (c) use 1024 points, (a) and (d) use the default number of points in corresponding papers with a batch size=16.
mutual spatial relation among each point pair. Thus, the local extractor must query neighbors by traversing the whole point cloud several times. As long as a source point is selected as a neighbor, a copy will be generated to facilitate parallelization. This causes multiple times of memory consumption. Then, each copy of the neighbor is processed independently by the learnable parameters, resulting in re-computations for each point. Although such delicate operations have shown great ability to extract local geometry, the redundant use of memories and calculators limits their applications under resource constraints.

Considering these issues, we propose a novel network with linear memory consumption and linear computation complexity. To avoid redundant access, we introduce an auxiliary container as an anchor to exchange features between the source and center points. Each point is processed by two operations: first, push its feature to only one auxiliary container; second, pull the feature from only one auxiliary container. Point close to each other in Euclidean space will fall into the same container automatically, and the container only costs a tiny maintenance overhead. The linear operation pair, the so-called APP operator, achieves a huge reduction in the memory consumption and computation complexity while maintaining comparable accuracy with previous methods. To facilitate the learning of the local structure, we use an online normal estimation module to provide the explainable geometric information to enhance our APP modeling capability. The comparisons among different styles of network structure, including PointNet, previous local-aggregator-based networks, and the proposed APP-based network, are depicted in Fig 2. It’s straightforward that the overhead of the APP is linear to the number of input points. We conducted a detailed quantitative analysis of the running speed and memory. As depicted in Fig 1, we achieve an inference speed of more than 2000 samples/second with batch size=16. Among those networks outperforming 92%, we are 2× faster than the second efficient network, ShellNet [45]. Moreover, our network consumes an amazing low GPU memory. According to Fig 1(b), APP only costs memories less than 10GB with a batch
size=1024. Correspondingly, the maximum supported batch size in our machine for Point Transformer [46] is 64, which costs more than 30GB of memory. And the lightweight version of PointMLP [22] consumes more than 25GB with a maximum batch size of 256. Furthermore, according to Fig 1(c), we can even achieve a speed more than 10,000 samples/s with a batch size of 256, which is 5× faster than the peak of other baselines. More details and analysis are presented in the following sections.

2 Related Work

In this section, we review some remarkable works in 3D object classification. According to the data type, we divide those methods into multi-view-based and point-cloud-based methods.

2.1 Multi-view Based Methods

Multi-view-based methods consider learning point features with the mature CNN by projecting the point cloud into a series of 2D images from multiple views. Some works [32,43] are devoted to investigating how to fuse the features with pooling policy. Yang et al. [42] propose to combine with the guidance of the inter-relationship among views. Wei et al. [35] treat the different views as nodes in a graph, which facilitates to use of GCN to exploit the relation among views. In general, the projection process is view-dependent. It loses some geometrical information inevitably, making it challenging to fuse the view-wise features from different views to obtain a consistent and discriminative representation for each 3D object.

2.2 Point Based Methods

This is a widely used method in point cloud learning. Motivated by the 2D CNN [13,21], lots of works are designed to aggregate local point features. The local area is determined by the distance in Euclidean space or the topology. DGCNN [30] constructs a graph to learn with the connections between nodes. PointNet++ [26] provides a standard paradigm for fully point-based method. It points out how to locate the neighbor area and aggregate local features point-wisely. Subsequent works [45,41] improve the designs of local area structure, downsampling methods, and local aggregator. Different from PointNet++ [26], some works design convolutional operators for point cloud. SpiderNet [5] uses Taylor expansion to approximate the spatial distribution of filters. KPConv [33] specifies a series of kernel points to implement convolution. PointConv [36] directly learns the values of the filter through coordinates. The recent hot topic, transformer, has started to show its effectiveness in the point cloud. PCT [6] is the first totally transformer-based network. Point Transformer [46] enhances the local aggregator with a Transformer-like operator. These methods have achieved reasonable accuracy. However, efficiency and low memory consumption are not
their main targets. So they leave us with a huge room for eliminating the overhead. We achieve the most efficient network for point cloud classification with the proposed APP operator.

3 The Proposed Method

3.1 Motivation and Discussion

**Preliminaries** The source point cloud with $N$ points is denoted as $P = \{p_1, p_2, ..., p_N\}$. The target is to aggregate current source features into a set of center points $Q = \{q_1, q_2, ..., q_M\}$. The general process of a local point feature aggregation is as follows:

$$g_i = R(G(r(q_i, p_j, [f_i, f_j]), f_j)|p_j \in N(q_i)))$$

(1)

where $q_i$ is the center point, $g_i$ is the output feature for the center point. $N(q_i)$ queries neighbors for $q_i$. $f_*$ and $r(q_i, p_j, [f_i, f_j])$ denotes the input feature of $p_*$ or $q_*$. $r(q_i, p_j, [f_i, f_j])$ measures the relation between the neighbor point and the center point. Most methods mainly consider the position relation and some others would combine the feature relation. $G$ and $R$ refers to the features transformation and reduction. The reduction function is MAX Pooling in most cases.

**Analysis and Insights** For the common point-based system [26], each center point selects neighbors from the source points independently. If a source point is selected as multiple center points’ neighbor, it has to participate in the aggregation process repeatedly and occupies several duplicates of memories to facilitate parallelization. The advantage of this manner is obvious: it enables the neighbor point to adaptively contribute to different center points according to the current spatial and feature relation. However, it inevitably induces extra computations and memory consumption for neighbors. In the first layer of PointNet++ [26], $N = 1024$, $M = 512$, each center point queries 32 neighbors, which indicates that each source point would be replicated and re-computed for $\frac{512 \times 32}{1024} = 16$ times.

Fig. 3. The feature aggregation process is driven by the APP operator. The inputs are coordinates with features and the output are downsampled coordinates and updated features.
To avoid using redundant resources, we notice that all the previous methods conduct the aggregation procedure from the view of the center point \( Q \), and each center point doesn’t care about how many times other center points have accessed the neighbor. To limit the access times of each source point, we turn to guide the aggregation from the view of the source point. In this paper, we proposed a so-called APP (auxiliary-based push and pull) operator which introduces the auxiliary point set \( A \) as a bridge between the source points and the center points. Each source point \( p \) only pushes its information to one auxiliary point, and each center point \( q \) only pulls information from one auxiliary point, i.e., during the whole feature aggregation process, each source point only participates in emitting feature for one time. And the center point only participates in gathering features for one time. Such paired operations linearize the complexity of computation. Although reducing some overhead, auxiliary points may also introduce incorrect dependencies on the artifacts. To eliminate the influence of auxiliary points, we implement the operation pair in a novel manner, with which the influence of auxiliary points is reducible. The core idea is to find a group of functions \( \{\alpha, \beta, \gamma\} \) which satisfy the following relation:

\[
\gamma(x, y) = \alpha(x,*)\beta(*, y)
\]  

(2)

where \(?\) denotes some kind of operation. We will introduce an instantiation of the proposed auxiliary-point-aided aggregation in the following sections.

### 3.2 Auxiliary-based Push and Pull Operations

**Block Partition** For simplicity, the auxiliary points \( A = \{a_0, a_1, ..., a_{Na} \} \) is a subset of the source point cloud \( P \) with a downsample rate of \( r_a \). Following RandLA-Net [10], we adopt random sampling instead of farthest point sampling to accelerate the generation of \( A \). As mentioned in Section 3.1 that each source point only emits its information to one auxiliary point, we conduct a 1-NN query for every source point from the auxiliary point set \( A \). For a point \( p_i \), we denote its auxiliary point as \( A(p_i) \). Points close to each other in Euclidean space naturally choose the same auxiliary point. As a result, the whole point cloud is partitioned into several non-overlapped blocks \( \{B(a_0), B(a_1), ..., B(a_{Na})\} \), where \( B(a_i) \) denotes the source point set whose 1-NN auxiliary point is \( a_i \).

**Feature Aggregation** According to the previous analysis, a reducible operation group satisfying Equation 2 eliminates the influence of auxiliary point to prevent artifacts. Considering the basic equation that

\[
e^{x-y} = e^{x-\ast}e^{\ast-y} = e^{x-\ast} : e^{\ast-y},
\]  

(3)

we construct the function group \( \{\alpha, \beta, \gamma\} \) with exponential function. The whole feature aggregation process consists of two steps: **Push-step** and **Pull-step**.
Push-step

In this step, the information of the source point is delivered to the auxiliary point as followings:

$$g_{p_i\rightarrow A(p_i)} = f_i \cdot e^{W \times [\phi(p_i) - \phi(A(p_i))]}$$

(4)

where $\phi(*)$ is a global sharing position encoding function implemented by [FC Layer + BatchNorm + Leaky ReLU], it lifts the coordinate from a 3-channel vector to an embedding of $C_{in}$ channels. $W \in \mathbb{R}^{C_{in} \times C_{in}}$ is a learnable weight matrix. $\cdot$ denotes element-wise multiplication and $\times$ denotes matrix multiplication.

This process pushes the source point’s feature to the auxiliary point according to the spatial relation. Mind that all the source points belonging to the same auxiliary block will push their feature to the same auxiliary point; thus the final feature in the auxiliary point is as follows:

$$g_{A(p_i)} = \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))} f_j \cdot e^{W \times [\phi(p_j) - \phi(A(p_i))]}.$$  

(5)

Pull-step

Each auxiliary point gathers the block context through Push-step. In this step, we apply an inverse operation for each source point to pull features from the corresponding auxiliary point. The inverse operation is formulated as follows:

$$g_i = g_{A(p_i)\rightarrow p_i} = g_{A(p_i)} \cdot e^{W \times [\phi(A(p_i)) - \phi(p_i)]},$$

$$= \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))} f_j \cdot e^{W \times [\phi(p_j) - \phi(A(p_i))]} \cdot e^{W \times [\phi(A(p_i)) - \phi(p_i)]},$$

$$= \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))} f_j \cdot e^{W \times [\phi(p_j) - \phi(p_i)].$$

(6)

Essentially speaking, Formula (6) is an instantiation of Formula (4) with a reduction function of MEAN pooling. It considers each source point the center point and gathers neighbors’ features according to the pair-wise spatial relation. The Push-step and Pull-step share the same weight matrix $W$ and the position encoding function $\phi(*)$, the influence of the auxiliary point is mathematically reducible. Thus there is no need to conduct the computation of $\phi(A(p_i))$ actually. The auxiliary point only affects the receptive field of $g_i$. Moreover, since $e^{W \times [\phi(A(p_i)) - \phi(p_i)]} = \frac{1}{e^{W \times [\phi(A(p_i)) - \phi(p_i)]},}$ the spatial relation computed in Formula (4) can be directly reused. This greatly reduces the overhead both on computation complexity and memory consumption.

Then the final features are updated by

$$g_{i,\text{final}} = \delta([g_i, f_i]).$$

(7)

where $\delta(*)$ is a non-linear function constituted by \{FC+BatchNorm+LeakyReLU\}, $g_{i,\text{final}} \in \mathbb{R}^{C_{out}}$. 

APP-Net for Efficient Point Cloud Classification 7
Block-based Down Sample After processing by Push-step and Pull-step, the point cloud still has N points, which is similar to the PointNet [25]. We design a block-based downsample strategy to further reduce the intermediate overhead. Like the operation in Section 3.2, we downsample the point cloud with a rate of $r_d$ and re-split the whole cloud into several non-overlap blocks $\{D_0, D_1, \ldots, D_{\frac{N}{r_d}}\}$ based on 1-NN algorithm. Then, for all the points belonging to the same block $D_i$, their features are aggregated by a MAX pooling function as follows:

$$g_{d_i} = \text{MAX}\{f_j | p_j \in D_i\}$$ (8)

3.3 Local Geometry by Online Normal and Curvature Estimation

The normal and curvature reflect local surface geometry. We use them to reduce the dependence on the delicate local extractor. In point cloud, normal estimation is approximated by the problem of estimating the normal of a plane tangent to the surface. In this paper, we adopt the simplest PCA-based method. A brief revision is present here to make the text more readable. For the centroid point $p$, assembling its local covariance matrix by

$$Cov = \frac{1}{|N(p)|} \sum_{p_i \in N(p)} (p_i - p) \cdot (p_i - p), \quad Cov \cdot \vec{v}_j = \lambda_j \cdot \vec{v}_j, j \in \{0, 1, 2\}$$ (9)

$\vec{v}_n$ and $\lambda_n$ represent the eigenvectors and eigenvalues of the covariance matrix respectively. The eigenvector $\vec{v}$ corresponding to the minimum eigenvalue is the estimated normal. Supposed that $\lambda_0$ is the minimum eigenvector, then the curvature $\sigma$ of the local surface is determined by

$$\sigma = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$$ (10)
A direction consistency check will flip those normal who do not orient towards a pre-specified viewpoint to alleviate the ambiguity in the normal direction. To further reduce the influence of noise, we apply a brute $\text{abs}$ operation to make every component of the normal vector non-negative. The resulting vector is no longer normal in the physical meaning. The comparison among different normals is depicted in Fig. [4]

### 3.4 The Full Structure of APP-Net

This section introduces how to build an end-to-end network with the APP operator. As shown in Figure 5, the input is the information of the point, including the position $xyz$, the normal and local curvature (optional). An embedding layer lifts the input normal (and curvature) to a feature space with a dimension of 32. Three layers consisting of APP operator and blocks down sample are cascaded to aggregate neighbor features for every point. A pooling layer outputs the global feature after the last APP layer. According to the global feature, a classifier consisting of MLPs would predict the probability for each category.

### 4 Experiments

To explore the characteristics of our method, we conduct extensive experiments on a synthetic dataset and a real scene dataset. Our method has achieved comparable accuracy while maintaining very low overhead and high efficiency compared with existing methods. We further conduct some ablation studies on Section 4.4 to test how every module works.

#### 4.1 Settings

Our experiments are evaluated on a server with one Tesla V100 GPU. We use PyTorch [24] to implement most of our projects. For all the tasks, we use the default Adam [12] to update the network parameters. And we use the cosine learning rate scheduler to update the learning rate, with an initial learning rate of $2e-3$, and the minimum learning rate threshold is set to $2e-4$. The cycle for the cosine scheduler is $T_{\text{max}} = 200$. For all the experiments, we train the network for 300 epochs with training batch size=32 and test batch size=16; the first epoch is used to warm up. All the results of the comparison methods are obtained in
Table 1. Classification on ModelNet40. We report the overall accuracy, train speed, test speed, and the number of parameters of some baselines. 5k denotes 4096 points and the 7k for KPConv means using around 7,000 points. ‘P’ and ‘N’ means using point and ground truth normal, respectively. (Coarse normal) means using the online estimated normal. **Bold** number denotes the best one and blue number denotes the second-best one.

| Model            | Inputs   | Train Speed (samples/s) | Test Speed (samples/s) | Param | OA(%) |
|------------------|----------|-------------------------|------------------------|-------|-------|
| PointNet [25]    | 1k P     | 960.9                   | 1422.4                 | 3.5M  | 89.2  |
| Pointnet++ [26]  | 1k       | 352.2                   | 730.2                  | 1.41M | 90.7  |
| PointNet++ [26]  | 5k P+N   |                         |                        | 1.41M | 91.9  |
| PointCNN [16]    | 1k P     |                         |                        | 92.5  |
| PointConv [35]   | 1k P+N   | 104.5                   | 76.4                   | 18.6M | 92.5  |
| KPConv [33]      | 7k P     | 211.7                   | 717.7                  | 15.2M | 92.9  |
| DGCNN [30]       | 1k P     |                         |                        | 92.9  |
| RS-CNN [19]      | 1k P     |                         |                        | 92.9  |
| DensePoint [18]  | 1k P     |                         |                        | 92.8  |
| ShellNet [45]    | 1k P     | 551.3                   | 1077.5                 | 0.48M | 93.1  |
| PointASNL [41]   | 1k P+N   | 285.9                   | 316.4                  | 3.2M  | 93.2  |
| PosPool [20]     | 5k P     | 51.0                    | 59.5                   | 18.5M | 93.2  |
| Point Trans. [2] | 1k P     |                         |                        | 92.8  |
| GBNet [28]       | 1k P     | 17.7                    | 175.9                  | 8.39M | 93.8  |
| GDANet [39]      | 1k P     | 29.8                    | 273.3                  | 0.93M | 93.4  |
| PA-DGC [38]      | 1k P     |                         |                        | 93.6  |
| MLMSPT [8]       | 1k P     |                         |                        | 92.9  |
| PCT [6]          | 1k P     | 115.7                   | 1044.9                 | 2.88M | 93.2  |
| Point Trans. [46]| 1k P     | 67.1                    | 82.3                   | 12.9M | 93.7  |
| CurveNet [23]    | 1k P     | 89.9                    | 112.9                  | 2.04M | 93.8  |
| PointMLP [22]    | 1k P     | 60.4                    | 169.0                  | 12.6M | **94.1** |
| PointMLP-elite [22]| 1k P   | 240.1                   | 632.8                  | 0.68M | 93.6  |
| APP-Net 5k P+N   | 1097.0   | **2164.9**              | 0.77M                  | 93.6  |
| APP-Net(Coarse normal) | 5k P | 1060.7 | 1451.8 | 0.77M | 93.0  |

three ways: 1. careful reproduction in our environment to prevent the higher performance caused by machine development (To confirm fairness, if we fail to reproduce the open results, we will adopt the following two ways). 2. the reported results in the original papers; 3. the updating results on the public websites or other published papers.

4.2 Shape Classification on ModelNet40

ModelNet40 [27] is the most influential benchmark for 3D classification task, including 40 common categories. The point-cloud-type data is synthesized by randomly sampling the surface of 3D CAD models, with a training set of 9,843 samples and a testing set of 2,468 samples. We report the most widely used metric-Overall Accuracy, on the testing set. For fairness, we do not adopt the
Table 2. Classification on ScanObjectNN. The input for $A^3$-Net contains 1024 points. We run the experiment five times and report the mean±std. **Bold** number denotes the best one and **blue** number denotes the second-best one.

| Methods              | Inputs       | OA(%)        | Train Speed  | Test Speed   |
|----------------------|--------------|--------------|--------------|--------------|
| PointNet [25]        | Point        | 68.2         | 960.9        | 1422.4       |
| SpiderCNN [40]       | Point        | 73.7         |              |              |
| PointNet++ [26]      | Point        | 77.9         | 352.2        | 730.2        |
| DGCNN [30]           | Point        | 78.1         |              |              |
| PointCNN [16]        | Point        | 78.5         |              |              |
| BGA-DGCNN [34]       | Point        | 79.7         |              |              |
| BGA-PN++ [34]        | Point        | 80.2         |              |              |
| DRNet [27]           | Point        | 80.3         |              |              |
| GBNet [28]           | Point        | 80.5         |              |              |
| SimpleView [4]       | Multi-view   | 80.5 ± 0.3   |              |              |
| PRANet [1]           | Point        | 82.1         |              |              |
| MVTN [7]             | Multi-view   | 82.8         |              |              |
| PointMLP [22]        | Point        | **85.4 ± 0.3** | 60.4        | 169.0        |
| PointMLP-elite [22]  | Point        | **83.8 ± 0.6** | 240.1       | 632.8        |
| **APP-Net**          | **Point**    | **84.1 ± 0.1** | **1898.7**  | **2442.0**   |

voting strategy for all the methods (which often improves the accuracy by 0.4% for some methods). Besides, we also report the computation overhead. As shown in Table 1, we achieve comparable accuracy to these SOTAs as well as maintain a very efficient running speed. The speed is measured by \(\frac{\text{Total Samples}}{\text{Total Process Time}}\). Most speeds of the baseline methods are faster than the one reported in the previously published papers. We believe it is mainly because we reproduce them with the recent optimized CUDA operations. Although APP-Net uses more points in each sample (4096 vs. 1024), it still runs faster than all the baselines. And it’s 3× faster than the PointNet++ [26], 19× faster than CurveNet [23] during test. APP-Net has a clear advantage over the other methods even with the coarse normal (online estimated).

4.3 Shape Classification on ScanObjectNN

Considering the saturation of ModelNet40 [37] and challenging real-world cases, Uy et al. propose the ScanObjectNN [34], collecting by scanning the indoor objects. The real-world data often face some annoying issues, like the cluttered or occluded by fore/background. So ScanObjectNN reveals the great potential to promote the classification application in the real world.

According to the common practice in other works, we use the hardest variant PB_T50_RS to conduct our experiments. The whole dataset contains a training set with 11416 samples and a testing set with 2882 belonging to 15 categories. We choose the most representative point-based and multi-view methods as the baselines. The normal we put into the network is computed online, and the duration of normal estimation is considered in the speed test. The overall accuracy is
Table 3. Time cost analysis for each module of the APP-Net. The time cost is counted after the system runs stably.

|                        | Normal Estimation | Feature Embedding | Layer 1 | Layer 2 | Layer 3 | Classifier |
|------------------------|-------------------|-------------------|---------|---------|---------|------------|
| APP-Net                | 2ms               | 0.13ms            | 1.2ms   | 1.4ms   | 1.1ms   | 0.07ms     |
| PointNet++ [20]        | -                 | -                 | 20.01ms | 12.8ms  | 0.88ms  | 0.27ms     |

Table 4. The reducible operation. We explore whether the reducible attribute is necessary and a new instantiation of the reducible philosophy. The best one is colored with Bold, and the second best one is blue.

|                                  | OA(%)            |
|----------------------------------|------------------|
| Aux. Point Reducible             | **84.1**         |
| Aux. Point Not Reducible by new Position Encoding Function | 82.8             |
| Aux. Point Not Reducible by Introducing Non-linear | 83.2             |
| Concat-driven APP                | **83.5**         |

shown in Table 2. Following SimpleView [1], we report the mean±std. We almost outperform all the previous methods, except for the PointMLP [22]. Considering that APP-Net is 14× faster than PointMLP [22], the gap in accuracy is acceptable. In Table 3, we report the time cost of each module under test mode, with batch size=16 and N =1024. Each layer only costs 1.1 ∼ 1.4ms. Although the normal estimation occupies a relatively long time(2ms), it’s worthwhile because the estimated normal promotes our simple architecture to achieve better accuracy.

4.4 Ablation Studies

There are some fine-designed structures in APP. To test their functionality and substitutability, we conducted some ablation studies and analyses based on the ScanObjectNN. In some experiments, we propose new architectures for comparison; the details of these new designs will be presented in the supplementary material.

The Advantages of Reducible Operation One of the core designs of APP is the reducible operation-pair: **Push-Step** and **Pull-step**. They make each point’s representation independent of the auxiliary point. Moreover, they save the computation resources by re-using the spatial relation. To verify the effectiveness, we design two types of non-reducible operations: 1. use two different functions to compute the spatial relation for the Push-step and Pull-step respectively; 2. add non-linear operation, i.e., BatchNorm and Leaky ReLU, to the spatial relation. The results in Table 4 show that the reducible operation is superior to the others in accuracy, run speed, and memory consumption. As to the concat-driven APP, we design a concatenation-based reducible APP opera-
tor. It also achieves promising results, which further proves the importance of the philosophy of reducible operation.

**The Effect of Normal** The normal act an essential role in the network. It provides compact and well explainable geometric cues, significantly reducing the need for a delicate local aggregator. To verify the effectiveness, we explore different inputs in the initial layer. As reported in Table 5, no matter using the real normal or the coarse estimated normal, the accuracy is improved by a large margin in all the datasets. Although the absolute normal degrades its physical meaning, it still contributes to the accuracy positively.

| Table 5. Different inputs to the network. The metric is OA and Abs(*) means an absolution operation. |
|---------------------------------------------------------------------------------------------------|
| ModelNet40(%) | ScanObjectNN(%) |
|----------------|-----------------|
| xyz            | 90.3            | 70.2            |
| GT normal      | **93.6**        | -               |
| Estimated normal+curvature | 92.4 | **84.1** |
| Abs(Estimated normal)+curvature | 93.0 | 83.1 |

**The Effect of Modeling Relation** We compare different ways of modeling spatial relations in this part. The variants include using the global position encoding or local position encoding or directly computing the spatial relation without using position encoding. The local position is computed by subtracting the corresponding center point. Considering that the transformer measures the relation between features, we also try to combine the feature relation in the APP. Results in Table 6 show that the global position encoding obtains better results than the local one. Maybe the local position is not stable due to the random block partition, which hinders the network convergence during training. Besides, the feature relation does not provide a positive effect. We guess it may wrongly build the couple relation between the position and feature. This also explains why the "Feat+Global Position Encoding" performs better than "Concat[Feat, Global Position] Encoding" since the former decouples the process of learning feature relation and spatial relation.

| Table 6. Different ways to model the relations among neighbors. |
|-------------------------------------------------------------|
| OA(%)                                                      |
| Global Position Encoding                                   | **84.1** |
| Local Position Encoding                                    | 82.8     |
| No Position Encoding                                       | 82.5     |
| Feat+Global Position Encoding                              | 83.6     |
| Concat[Feat, Global Position] Encoding                     | 82.1     |
Table 7. Different ways to update the feature after the Pull-step.

| Feature Updating Style | OA(%) |
|------------------------|-------|
| Concat                 | 84.1  |
| Not Concat             | 77.8  |
| Res Feature            | 83.7  |
| no Sigma               | 79.3  |
| no APP                 | 75.2  |

**Feature Updating Style** At the end of the Pull-step, every point’s feature is updated by concatenating the output with the original feature and sending it to a $\sigma(\ast)$ function. Among the comparisons, we try to remove some parts of it or to leverage a residual structure. As shown in Table 7, the concatenation manner is superior to the other configurations. Moreover, in the last two rows, we remove the $\sigma(\ast)$ function or the whole APP block; the results clearly state their indispensability.

**Network Depth** It’s well-known that network depth has an impact on performance. In the 2-layer and 3-layer version, we adopt the same $r_d$ and $r_a$. In the 4-layer version, due to the original number of the point being 1024, after two downsample operations, the remaining points are not sufficient to support a large $r_a$, so we adopt a smaller $r_a = 16$ in the last two layers. Table 8 shows that the 3-layer version achieves the best performance.

Table 8. The influence of the number of layer. With different layers, the rate will be adaptively adjusted.

| Configs                          | OA(%) |
|----------------------------------|-------|
| 2 layers, $r_d=[8,8,8]$, $r_a=[64,64,64]$ | 83.0  |
| 3 layers, $r_d=[8,8,8]$, $r_a=[64,64,64]$ | 84.1  |
| 4 layers, $r_d=[4,4,8,8]$, $r_a=[64,64,16,16]$ | 83.9  |

**Down Sample Rate** The rate $r_a$ and $r_d$ serve as the network’s kernel size. They control the receptive field for each center point. For simplicity, we adopt the same $r_d$ for all layers in each variant. Results in Table 8 imply that a large $r_a$ for the auxiliary point generation is critical to a better representation, especially at the high level (according to the last two rows). And the result is relatively not sensitive to the rate $r_d$ for aggregation.

**Pooling Policies in the Aggregation** We have tested three common pooling policies to explore a proper aggregation operation for the second block partition. The position-adaptive pooling aggregates local points weighted by the reciprocal distance in Euclidean space. The results in Table 8 show that for the classification task, selecting the local most representative feature is more important than keeping all the details.
Table 9. Different rate configurations for the APP-Net.

| Rate          | OA(%) |
|---------------|-------|
| $r_d=[8,8,8], r_a=[64,64,64]$ | 84.1  |
| $r_d=[4,4,4], r_a=[64,64,64]$ | 84.1  |
| $r_d=[16,16,16], r_a=[64,64,64]$ | 83.5  |
| $r_d=[8,8,8], r_a=[32,32,32]$ | 82.6  |
| $r_d=[8,8,8], r_a=[32,64,32]$ | 83.7  |
| $r_d=[8,8,8], r_a=[64,64,32]$ | 83.1  |

Table 10. Different pooling policies.

| Pooling Policy            | OA(%) |
|---------------------------|-------|
| MAX Pooling               | 84.1  |
| AVG Pooling               | 79.7  |
| Position Adaptive Pooling | 79.9  |

5 Conclusion

This paper proposes a novel APP operator that aggregates local features through auxiliary points. It avoids redundant memory consumption and re-computation of the source point. Furthermore, the influence of the auxiliary points is reducible, which makes the method able to preserve more geometric details. Experiments on the synthetic and real datasets show a good balance between performance, speed, and memory consumption in the classification task. Especially in speed, it outperforms all the previous methods significantly. Our future goal is to extend it to more tasks, like semantic segmentation and object detection.

References

1. Cheng, S., Chen, X., He, X., Liu, Z., Bai, X.: Pra-net: Point relation-aware network for 3d point cloud analysis. IEEE Transactions on Image Processing 30, 4436–4448 (2021)
2. Engel, N., Belagiannis, V., Dietmayer, K.: Point transformer. IEEE Access 9, 134826–134840 (2021)
3. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)
4. Goyal, A., Law, H., Liu, B., Newell, A., Deng, J.: Revisiting point cloud shape classification with a simple and effective baseline. In: International Conference on Machine Learning. pp. 3809–3820. PMLR (2021)
5. Gu, X., Nahrstedt, K., Yu, B.: Spidernet: An integrated peer-to-peer service composition framework. In: Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 2004. pp. 110–119. IEEE (2004)
6. Guo, M.H., Cai, J.X., Liu, Z.N., Mu, T.J., Martin, R.R., Hu, S.M.: Pct: Point cloud transformer. Computational Visual Media 7(2), 187–199 (2021)
7. Hamdi, A., Giancola, S., Ghanem, B.: Mvtn: Multi-view transformation network for 3d shape recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1–11 (2021)
8. Han, X.F., Kuang, Y.J., Xiao, G.Q.: Point cloud learning with transformer. arXiv preprint arXiv:2104.13636 (2021)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
10. Hu, Q., Yang, B., Xie, L., Rosa, S., Guo, Y., Wang, Z., Trigoni, N., Markham, A.: Randla-net: Efficient semantic segmentation of large-scale point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11108–11117 (2020)

11. Hua, B.S., Tran, M.K., Yeung, S.K.: Pointwise convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 984–993 (2018)

12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25, 1097–1105 (2012)

14. Lang, I., Manor, A., Avidan, S.: Samplenet: differentiable point cloud sampling. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7578–7588 (2020)

15. Li, J., Chen, B.M., Lee, G.H.: So-net: Self-organizing network for point cloud analysis. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 9397–9406 (2018)

16. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: Pointcnn: Convolution on x-transformed points. Advances in neural information processing systems 31, 820–830 (2018)

17. Liu, K., Gao, Z., Lin, F., Chen, B.M.: Fg-net: Fast large-scale lidar point clouds understanding network leveraging correlated feature mining and geometric-aware modelling. arXiv preprint arXiv:2012.09439 (2020)

18. Liu, Y., Fan, B., Meng, G., Lu, J., Xiang, S., Pan, C.: Densepoint: Learning densely contextual representation for efficient point cloud processing. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5239–5248 (2019)

19. Liu, Y., Fan, B., Xiang, S., Pan, C.: Relation-shape convolutional neural network for point cloud analysis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8895–8904 (2019)

20. Liu, Z., Hu, H., Cao, Y., Zhang, Z., Tong, X.: A closer look at local aggregation operators in point cloud analysis. In: European Conference on Computer Vision. pp. 326–342. Springer (2020)

21. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)

22. Ma, X., Qin, C., You, H., Ran, H., Fu, Y.: Rethinking network design and local geometry in point cloud: A simple residual MLP framework. In: International Conference on Learning Representations (2022)

23. Muzahid, A., Wan, W., Sohel, F., Wu, L., Hou, L.: Curvenet: Curvature-based multitask learning deep networks for 3d object recognition. IEEE/CAA Journal of Automatica Sinica 8(6), 1177–1187 (2020)

24. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32 (2019)

25. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)

26. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. arXiv preprint arXiv:1706.02413 (2017)
27. Qiu, S., Anwar, S., Barnes, N.: Dense-resolution network for point cloud classification and segmentation. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 3813–3822 (2021)
28. Qiu, S., Anwar, S., Barnes, N.: Geometric back-projection network for point cloud classification. IEEE Transactions on Multimedia (2021)
29. Ran, H., Zhuo, W., Liu, J., Lu, L.: Learning inner-group relations on point clouds (2021)
30. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3693–3702 (2017)
31. Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.H., Kautz, J.: Splatnet: Sparse lattice networks for point cloud processing. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2530–2539 (2018)
32. Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E.: Multi-view convolutional neural networks for 3d shape recognition. In: Proceedings of the IEEE international conference on computer vision. pp. 945–953 (2015)
33. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: Flexible and deformable convolution for point clouds. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6411–6420 (2019)
34. Uy, M.A., Pham, Q.H., Hua, B.S., Nguyen, T., Yeung, S.K.: Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1588–1597 (2019)
35. Wei, X., Yu, R., Sun, J.: View-gen: View-based graph convolutional network for 3d shape analysis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1850–1859 (2020)
36. Wu, W., Qi, Z., Fuxin, L.: Pointconv: Deep convolutional networks on 3d point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9621–9630 (2019)
37. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. 2015 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR) pp. 1912–1920 (2015)
38. Xu, M., Ding, R., Zhao, H., Qi, X.: Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 3173–3182 (2021)
39. Xu, M., Zhang, J., Zhou, Z., Xu, M., Qi, X., Qiao, Y.: Learning geometry-disentangled representation for complementary understanding of 3d object point cloud. arXiv preprint arXiv:2012.10921 2 (2021)
40. Xu, Y., Fan, T., Xu, M., Zeng, L., Qiao, Y.: Spidercnn: Deep learning on point sets with parameterized convolutional filters. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 87–102 (2018)
41. Yan, X., Zheng, C., Li, Z., Wang, S., Cui, S.: Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5589–5598 (2020)
42. Yang, Z., Wang, L.: Learning relationships for multi-view 3d object recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7505–7514 (2019)
43. Yu, T., Meng, J., Yuan, J.: Multi-view harmonized bilinear network for 3d object recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 186–194 (2018)
44. Zhang, C., Wan, H., Liu, S., Shen, X., Wu, Z.: Pvt: Point-voxel transformer for 3d deep learning. arXiv preprint arXiv:2108.06076 (2021)
45. Zhang, Z., Hua, B.S., Yeung, S.K.: Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1607–1616 (2019)
46. Zhao, H., Jiang, L., Jia, J., Torr, P.H., Koltun, V.: Point transformer. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 16259–16268 (2021)
Appendix

In the following parts, we introduce more details to make the ablation study experiments more clear.

A The Advantages of the Reducible Operations

In this section, we use four configurations to explore the advantages of the reducible auxiliary points. Below are the details of each setting.

A.1 Aux. Point Reducible

This is the default setting in the main body of the paper, i.e., using the exponential-function-based operation pair:

**Push Step**

\[ g_{p_i \rightarrow A(p_i)} = f_i \cdot e^{W \times [\phi(p_i) - \phi(A(p_i))]}, \]  

**Pull Step**

\[
\begin{align*}
&g_i = g_{A(p_i) \rightarrow p_i} \\
&= g_{A(p_i)} \cdot e^{W \times [\phi(A(p_i)) - \phi(p_i)]} \\
&= \left\{ \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))} f_j \cdot e^{W \times [\phi(p_j) - \phi(A(p_i))]} \right\} \cdot e^{W \times [\phi(A(p_i)) - \phi(p_i)]} \\
&= \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))} f_j \cdot e^{W \times [\phi(p_j) - \phi(A(p_i))]} \cdot e^{W \times [\phi(A(p_i)) - \phi(p_i)]}. 
\end{align*}
\]

A.2 Aux. Point Not Reducible by new position Encoding Function

In this setting, we use a different position encoding function \( \phi'(\cdot) \) for the Pull-Step as follows:

**Pull Step**

\[
\begin{align*}
&g_i = g_{A(p_i) \rightarrow p_i} \\
&= g_{A(p_i)} \cdot e^{W \times [\phi'(A(p_i)) - \phi'(p_i)]} \\
&= \left\{ \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))} f_j \cdot e^{W \times [\phi'(p_j) - \phi'(A(p_i))]} \right\} \cdot e^{W \times [\phi'(A(p_i)) - \phi'(p_i)]} \quad (13)
\end{align*}
\]

The influence of the auxiliary point is not reducible with the different position encoding function \( \phi'(\cdot) \). Although it introduces more learnable parameters, results in Table 4 of the original paper show a performance drop.
A.3 Aux. Point Not Reducible by Introducing Non-linearity

The original spatial relation is calculated by a linear mapping matrix $W$. To introduce non-linearity, we append a BatchNorm and LeakyReLU after the linear mapping as follows:

**Push Step**

$$g_{p_i \rightarrow A(p_i)} = f_i \cdot e^{LeakyReLU(BN(W \times [\phi(p_i) - \phi(A(p_i))])} \quad (14)$$

**Pull Step**

$$g_i = g_{A(p_i) \rightarrow p_i} = g_{A(p_i)} \cdot e^{LeakyReLU(BN(W \times [\phi(A(p_i)) - \phi(p_i)])}$$

$$= \left\{ \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))}^{} W \times [\phi(p_j) - \phi(A(p_i))) \right\} f_j \cdot e^{LeakyReLU(BN(W \times [\phi(p_i) - \phi(A(p_i)))} \right)$$

$$+ e^{LeakyReLU(BN(W \times [\phi(A(p_i)) - \phi(p_i))}$$

$$= \left\{ \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))}^{} W \times [\phi(p_j) - \phi(p_i)), f_j - f_i] \right\} \quad (15)$$

A.4 Concat-driven APP

In this part, we propose a concat-based APP operation pair, which also eliminates the influence of the auxiliary point. Notice that

$$(x - *) + (* - y) = x - y \quad (16)$$

we design the new **Push-step** and **Pull-step** as follows:

**Push Step**

$$g_{p_i \rightarrow A(p_i)} = W \times [\phi(p_i) - \phi(A(p_i)), f_i]$$

$$= \left\{ \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))}^{} W \times [\phi(p_j) - \phi(A(p_i))) \right\} f_j$$

$$= \left\{ \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))}^{} W \times [\phi(p_j) - \phi(p_i)), f_i - f_j] \right\} \quad (17)$$

**Pull Step**

$$g_i = g_{A(p_i) \rightarrow p_i} = g_{A(p_i)} + W \times [\phi(A(p_i)) - \phi(p_i), -f_i]$$

$$= \left\{ \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))}^{} W \times [\phi(p_j) - \phi(A(p_i))) \right\} f_j$$

$$+ W \times [\phi(A(p_i)) - \phi(p_i), -f_i]$$

$$= \left\{ \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))}^{} W \times [\phi(p_j) - \phi(p_i)), f_i - f_j] \right\} \quad (18)$$

The result is superior to the non-reducible version, which further confirms the importance of the ”Reducible Philosophy”.

B  The effect of modeling relation

We adopt different manners to model the relationships among neighbors, including the spatial relation and feature relation. Below are the details of the configuration.

B.1 Global Position Encoding

This is the default setting in the main paper.

B.2 Local position Encoding

In this part, we calculate the centroid $V = \{v_0, v_1, ..., v_k\}$ for each block, and the position encoding is modified from $\phi(p_i)$ to $\phi(p_i - v)$, where $v$ is the corresponding centroid.

B.3 No position Encoding

This is the simplest spatial relation. We directly use the original coordinates, which means the weight matrix has a shape of $3 \times C_{in}$, rather than $C_{in} \times C_{in}$.

B.4 Feat+Global Position Encoding

To explore the feature relation, we design two manners to combine the feature into the process of weighting the original feature. The first one is introduced in this part. The Push-step and Pull-step are modified as follows:

**Push Step**

$$g_{p_i \rightarrow A(p_i)} = f_i \cdot e^{W_s \times [\phi(p_i) - \phi(A(p_i))]} \cdot e^{W_f \times f_i}.$$  \hspace{1cm} (19)

**Pull Step**

$$g_i = g_{A(p_i) \rightarrow p_i} = g_{A(p_i)} \cdot e^{W_s \times [\phi(A(p_i)) - \phi(p_i)]} \cdot e^{W_f \times (-f_i)} = \frac{1}{|B(A(p_i))|} \sum_{p_j \in B(A(p_i))} f_j \cdot e^{W_s \times [\phi(p_j) - \phi(p_i)]} \cdot e^{W_f \times (f_j - f_i)}.$$  \hspace{1cm} (20)

According to the red part, the feature relation is modeled by linearly mapping the feature difference. Moreover, the spatial relation and feature relation are decoupled.

B.5 Concat [Feat, Global Position] Encoding

This is a coupled version to combine the feature relation. We use a weight matrix of $W_{sf} \in \mathbb{R}^{2C_{in} \times C_{in}}$ to jointly mapping the $[f_i, \phi(p_i)]$. According to the results in Table 6, the decoupled version is superior to the coupled one.
C Feature Updating Style

In this section, we introduce different feature updating styles at the end of the Pull-step. The default setting is "Concat":

\[ g_{i}^{\text{final}} = \delta([g_{i}, f_{i}]). \]  

(21)

The other settings are listed as follows:

- "Not Concat"

\[ g_{i}^{\text{final}} = \delta([g_{i}]). \]  

(22)

- "Res Feature"

\[ g_{i}^{\text{final}} = \delta([g_{i}]) + \eta(f_{i}). \]  

(23)

where \( \eta(*) \) is a linear mapping function.

- "No Sigma"

\[ g_{i}^{\text{final}} = g_{i}. \]  

(24)

- "No APP"

This means that we do not use the APP operations and directly maps the input feature as follows:

\[ g_{i}^{\text{final}} = \delta(f_{i}). \]  

(25)

D Different Pooling Policies

In this section, we introduce different policies for block pooling. The MAX and AVG mean max pooling and average pooling, respectively. The position adaptive pooling is implemented as follows:

\[ g_{i} = \frac{1}{\sum_{p_{j} \in B(A(p_{i}))} e^{-d(p_{i}, p_{j})}} \sum_{p_{j} \in B(A(p_{i}))} e^{-d(p_{i}, p_{j})} f_{j}. \]  

(26)

where \( d(p_{i}, p_{j}) \) measures the Euclidean distance between \( p_{i} \) and \( p_{j} \).