Starting from Non-Parametric Networks for 3D Point Cloud Analysis
Supplementary Material

Renrui Zhang\textsuperscript{1,5}, Liuhui Wang\textsuperscript{2,6}, Yali Wang\textsuperscript{4,5}, Peng Gao\textsuperscript{5}, Hongsheng Li\textsuperscript{1}, Jianbo Shi\textsuperscript{3,3}

\textsuperscript{1}CUHK MMLab \quad \textsuperscript{2}Peking University \quad \textsuperscript{3}University of Pennsylvania
\textsuperscript{4}Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences
\textsuperscript{5}Shanghai Artificial Intelligence Intelligence Laboratory \quad \textsuperscript{6}Heisenberg Robotics

\{zhangrenrui, gaopeng\}@pjlab.org.cn, jshi@seas.upenn.edu
wangliuhui0401@pku.edu.cn, hsli@ee.cuhk.edu.hk

1. Discussion
1.1. Why Do Trigonometric Functions Work?

We leverage the trigonometric function to conduct non-parametric raw-point embedding and geometry extraction. It can reveal the 3D spatial patterns benefited from the following three properties.

Capturing High-frequency 3D Structures. As discussed in Tancik et al. [45], transforming low-dimensional input by sinusoidal mapping helps MLPs to learn the high-frequency content during training. Similarly to our non-parametric encoding, Point-NN utilizes trigonometric functions to capture the high-frequency spatial structures of 3D point clouds, and then recognize them from these distinctive characteristics by the point-memory bank. In Figure 1, we visualize the low-frequency (Top) and high-frequency (Middle) geometry of the input point cloud, and compare them with the feature responses of Point-NN (Bottom). The high-frequency geometries denotes the spatial regions of edges, corners, and other fine-grained details, where the local 3D coordinates vary dramatically, while the low-frequency structure normally includes some flat and smooth object surfaces with gentle variations. As shown, aided by trigonometric functions, our Point-NN can concentrate well on these high-frequency 3D patterns.

Encoding Absolute and Relative Positions. Benefited from the nature of sinusoid, the trigonometric functions can not only represent the absolute position in the embedding space, but also implicitly encode the relative positional information between two 3D points. For two points, \( p_i = (x_i, y_i, z_i) \) and \( p_j = (x_j, y_j, z_j) \), we first obtain their \( C \)-dimensional embeddings referring to Equation (5~7) in the main paper, formulated as

\[
\begin{align*}
\text{PosE}(p_i) &= \text{Concat}(f_x^i, f_y^i, f_z^i), \\
\text{PosE}(p_j) &= \text{Concat}(f_x^j, f_y^j, f_z^j),
\end{align*}
\]

where \( \text{PosE}(\cdot) \) denotes the positional encoding by trigonometric functions, and \( f_x^i, f_y^i, f_z^i \in \mathbb{R}^{1 \times \frac{\pi}{\ell}} \) denote the embeddings of three axes. Then, their spatial relative relation can be revealed by the dot production between the two embeddings, formulated as

\[
\sum_{m=0}^{\frac{\pi}{\ell}} \cosine\left(\alpha(x_i - x_j)/\beta^{\text{num}}\right) = f_x^i f_x^j T,
\]

which indicates the relative x-axis distance of two points, in a similar way to the other two axes. Therefore, the trigonometric function is capable of encoding both absolute and relative 3D positional information for point cloud analysis.

Local Geometry Extraction. In Equation (9) of the main paper, we weigh each neighbor feature \( f_j \) within the local region by the relative positional embedding, \( \text{PosE}(\Delta p_j) \), formulated as

\[
f_{w}^c = (f_{cj} + \text{PosE}(\Delta p_j)) \odot \text{PosE}(\Delta p_j).
\]

The weighing is conducted sequentially by element-wise addition and multiplication. Firstly, the addition is to complement \( f_{cj} \) with higher frequency information. Due to feature expansion, the output dimensions of \( \text{PosE}(\Delta p_j) \) of 4 stages are respectively \( 2C_1, 4C_1, 8C_1, \) and \( 16C_1 \). As
Figure 1. Why Do Trigonometric Functions Work? For an input point cloud, we visualize its low-frequency and high-frequency geometries referring to [57], and compare with the feature responses after the first network stage of Point-NN, where darker colors indicate higher responses. As shown, Point-NN can focus on the high-frequency 3D structures with sharp variations of the point cloud.

Table 1. Can Point-NN Improve Point-PN by Plug-and-play? We report the accuracy (%) on the PB-T50-RS split of ScanObjectNN [48], ModelNet40 [53], and ShapeNetPart [59].

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>ScanObjectNN</th>
<th>ModelNet40</th>
<th>ShapeNetPart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-PN</td>
<td>87.1</td>
<td>93.8</td>
<td>86.6</td>
</tr>
<tr>
<td>+NN</td>
<td>+0.1</td>
<td>+0.2</td>
<td>+0.0</td>
</tr>
</tbody>
</table>

Table 2. Comparison of Training-free Methods in 3D. We report their performance without training on ModelNet40 [53].

<table>
<thead>
<tr>
<th>Method</th>
<th>Pre-train in 2D</th>
<th>Pre-train in 3D</th>
<th>3D Data</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointCLIP [62]</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>20.2</td>
</tr>
<tr>
<td>CALIP [16]</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>21.5</td>
</tr>
<tr>
<td>CLIP2Point [22]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>49.4</td>
</tr>
<tr>
<td>ULIP [58]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>60.4</td>
</tr>
<tr>
<td>PointCLIP V2 [73]</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>64.2</td>
</tr>
<tr>
<td>Point-NN</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>81.8</td>
</tr>
</tbody>
</table>

The embedding frequency depends on feature dimension referring to Equation (6) of the main paper, the embeddings at higher stages obtain higher frequencies. Taking the first stage as an example, \( \text{PosE}(\Delta p_j) \) is \( 2C_I \)-dimensional, but \( f_{cj} \) is a concatenation of two \( C_I \)-dimensional vectors, which makes their embedding frequencies inconsistent. Therefore, we adopt addition to endow \( f_{cj} \) with the frequency corresponding to \( 2C_I \) dimension. Then, the second-step multiplication weighs the magnitude of each element in \( f_{cj} \) by its relative positional information. This determines the importance of different neighbor points in the subsequent pooling operations, and the final aggregated features of the local neighborhood. In this way, Point-NN can effectively embed local 3D patterns via \( \text{PosE}(\cdot) \) without any learnable operators.

1.2. Can Point-NN Improve Point-PN?

Point-NN can provide complementary geometric knowledge and serve as a plug-and-play module to boost existing learnable 3D models. Although Point-PN is also a learnable 3D network, the enhanced performance brought by Point-NN is marginal as reported in Table 1. By visualizing feature responses in Figure 2, we observe that the complementarity between Point-NN and Point-PN is much weaker than that between Point-NN and PointNet++ [36]. This is because the non-parametric framework of Point-PN is mostly inherited from Point-NN, also capturing high-frequency 3D geometries via trigonometric functions. Therefore, the learnable Point-PN extracts similar 3D patterns to Point-NN, which harms its plug-and-play capacity.

1.3. Training-free Methods in 3D

Our Point-NN conducts no training, but requires 3D training data to construct the point-memory bank. Inspired by the transfer learning in 2D and language [2, 10, 12, 27, 60, 63, 68, 72], some recent works [16, 22, 58, 62, 73] adapt the pre-trained models from other modalities, e.g., CLIP [39], to 3D domains in a zero-shot manner. Via the diverse pre-trained knowledge, they are also training-free and do not need any 3D training data. As compared in Table 2, different from other methods based on 2D or 3D pre-training, our method is a pure non-parametric network without any learnable parameters or pre-trained knowledge.
Figure 2. **Can Point-NN Improve Point-PN by Plug-and-play?** We visualize the feature responses after the first network stage for Point-NN, the trained PointNet++ [36] and Point-PN, where darker colors indicate higher responses. As shown, Point-PN captures similar 3D patterns to Point-NN, which harms their complementarity.

![PointNet++](image)

![Point-NN](image)

![Point-PN](image)

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>10</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-$k$ PoM</td>
<td>80.4</td>
<td>81.1</td>
<td>81.3</td>
<td>81.4</td>
<td>81.4</td>
<td>81.7</td>
<td><strong>81.8</strong></td>
</tr>
<tr>
<td>$k$-NN</td>
<td>80.7</td>
<td>79.5</td>
<td>67.0</td>
<td>45.7</td>
<td>36.4</td>
<td>8.5</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3. **Point-Memory Bank vs. $k$-NN.** ‘Top-$k$ PoM’ denotes the point-memory bank with top-$k$ similarities, and ‘All’ denotes 9,840 training samples. We utilize our non-parametric encoder to extract features and report the accuracy (%) on ModelNet40 [53].

1.4. **Point-Memory Bank vs. $k$-NN?**

Based on the already extracted point cloud features, our point-memory bank and $k$-NN algorithm both leverage the inter-sample feature similarity for classification without training, but are different from the following two aspects.

**Soft Integration vs. Hard Assignment.** As illustrated in Section (2.3) of the main paper, our point-memory bank regards the similarities $S_{\text{cos}}$ between the test point cloud feature and the feature memory, $F_{\text{mem}}$, as weights, which are adopted for weighted summation of the one-hot label memory, $T_{\text{mem}}$. This can be viewed as a soft label integration. Instead, $k$-NN utilizes $S_{\text{cos}}$ to search the $k$ nearest neighbors from the training set, and directly outputs the category label with the maximum number of samples within the $k$ neighbors. Hence, $k$-NN conducts a hard label assignment, which is less adaptive than the soft integration. Additionally, our point-memory bank can be accomplished simply by two matrix multiplications and requires no sorting, which is more efficient for hardware implementation.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Method</th>
<th>Gain (%)</th>
<th>Param.</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAConv</td>
<td>PnP-3D</td>
<td>+0.2 Acc.</td>
<td>+0.7 M</td>
<td>+14 h</td>
</tr>
<tr>
<td></td>
<td>Point-NN</td>
<td>+0.2 Acc.</td>
<td>+0 M</td>
<td>+48 s</td>
</tr>
<tr>
<td>VoteNet</td>
<td>PnP-3D</td>
<td>+1.4 AP$_{25}$</td>
<td>+0.3 M</td>
<td>+10 h</td>
</tr>
<tr>
<td></td>
<td>Point-NN</td>
<td>+1.2 AP$_{25}$</td>
<td>+0 M</td>
<td>+9.3 min</td>
</tr>
</tbody>
</table>

Table 4. **Point-NN vs. PnP-3D [38].** We adopt two baseline models for comparison, PAConv [56] and VoteNet [5], respectively on ModelNet40 [53] and SUN RGB-D [43] datasets.

**All Samples vs. $k$ Neighbors.** Our point-memory bank integrates the entire label memory with different weights. This can take the semantics of all training samples into account for classification. In contrast, $k$-NN only involves the nearest $k$ neighbors to the test sample, which discards the sufficient category knowledge from other training samples.

**Performance Comparison.** In Table 3, based on the point cloud features extracted by our non-parametric encoder, we implement the top-$k$ version of point-memory bank for comparison with $k$-NN, which only aggregates the label memory of the training samples with top-$k$ similarities. As the neighbor number $k$ increases, $k$-NN’s performance is severely harmed due to its hard label assignment, while our point-memory bank attains the highest accuracy by utilizing all 9,840 samples for classification, indicating their different characters.
1.5. Point-NN vs. PnP-3D?

One previous work, PnP-3D [38], proposes local-global 3D processing modules that are plugged into other 3D models for performance improvement. Different from Point-NN’s plug-and-play, PnP-3D introduces additional learnable parameters and requires to re-train the baseline networks from scratch, which is time-consuming. In contrast, our Point-NN is non-parametric and enhances the baseline directly during inference. In Table 4, we compare Point-NN with PnP-3D respectively on PAConv [56] for shape classification and VoteNet [5] for 3D object detection. As shown, our method contributes to similar performance enhancement on the benchmarks, while brings no extra parameters or re-training. In the table, we report the additional time for Point-NN to construct the point-memory bank before plug-and-play, which are 48 seconds and 9.3 minutes for the two tasks.

2. Related Work

3D Point Cloud Analysis. As the main data form in 3D, point clouds have stimulated a range of challenging tasks, including shape classification [29, 31, 35–37, 55], scene segmentation [4, 25, 71], 3D object detection [5, 18, 21, 34, 42, 64], 3D vision-language learning [16, 52, 62, 73]. Existing solutions as backbone networks can be categorized into projection-based and point-based methods. To handle the irregularity and sparsity of point clouds, projection-based methods convert them into grid-like data, such as tangent planes [46], multi-view depth maps [13, 17, 44, 62, 73], and 3D voxels [30, 32, 41, 67]. By doing this, the efficient 2D networks [19] and 3D convolutions [30] can be adopted for robust point cloud understanding. However, the projection process inevitably causes geometric information loss and quantization error. Point-based methods directly extract 3D patterns upon the unstructured input points to alleviate this loss of information. The seminal PointNet [35] utilizes shared MLP layers to independently extract point features and aggregate the global representation via a max pooling. PointNet++ [36] further constructs a multi-stage hierarchy to encode local spatial geometries progressively. Since then, the follow-up methods introduce advanced yet complicated local operators [31, 56] and global transformers [1, 8, 9, 14, 15, 61, 66] for spatial geometry learning. In this paper, we follow the paradigm of more popular point-based methods, and propose a pure non-parametric network, Point-NN, with its two promising applications. For the first time, we verify the effectiveness of non-parametric components for 3D point cloud analysis.

Local Geometry Operators. Referring to the inductive bias of locality [19, 24], most existing 3D models adopt delicate 3D operators to iteratively aggregate neighborhood features. Following PointNet++ [36], a series of methods utilize shared MLP layers with learnable relation modules for local pattern encoding, e.g., fully-linked webs [69], structural relation network [7], and geometric affine module [31]. Some methods define irregular spatial kernels and introduce point-wise convolutions by relation mapping [28], Monte Carlo estimation [20, 51], and dynamic kernel assembling [56]. Inspired by graph networks, DGCNN [50] and others [26, 47] regard points as vertices and interact local geometry through edges. Transformers [25, 70] are also introduced in 3D for attention-based feature communication. CurveNet [54] proposes generating hypothetical curves for point grouping and feature aggregation. Unlike all previous methods with learnable operators, Point-NN adopts non-parametric trigonometric functions to reveal the spatial geometry within local regions, and Point-NN further appends simple linear layers on top with high performance-parameter trade-off.

3. Implementation Details

Point-NN. The non-parametric encoder of Point-NN contains 4 stages. Each stage reduces the point number by half via FPS, and doubles the feature dimension during feature expansion. For shape classification, the initial feature dimension $C_I$ is set to 72, and the final dimension $C_G$ of global representation is 1,152. The neighbor number $k$ of $k$-NN is 90 for all stages. We set the two hyperparameters $\alpha, \beta$ in PosE(\cdot) as 1000 and 100, respectively, referring to Equation (6) and (7) in the main paper. For part segmen-
Figure 3. **Point-Memory Bank for Part Segmentation.** We first utilize the non-parametric encoder and decoder to extract the point-wise features of the point cloud in the training set. Then, we average the point features with the same part label to obtain the part-wise features, and construct them as the feature memory.

<table>
<thead>
<tr>
<th>Grouping Method</th>
<th>Feature Expand</th>
<th>Pooling</th>
<th>Magnitude $\alpha$</th>
<th>Wavelength $\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ball Query 78.5</td>
<td>70.0</td>
<td>Max 80.4</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$k$-NN 81.8</td>
<td>70.0</td>
<td>Ave 77.1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>w/o 70.0</td>
<td>w 81.8</td>
<td>Both 81.8</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>w 81.8</td>
<td></td>
<td></td>
<td>500</td>
<td>3000</td>
</tr>
</tbody>
</table>

Table 5. **Ablation Study of Non-Parametric Encoder.** We ablate the grouping method for local neighbors, feature expansion by concatenation, and pooling operation for feature aggregation. We report the classification accuracy (%) on ModelNet40 [53].

Table 6. **Magnitude $\alpha$ in Trigonometric Functions.** We report the classification accuracy of Point-NN on ModelNet40 [53].

Table 7. **Wavelength $\beta$ in Trigonometric Functions.** We report the classification accuracy of Point-NN on ModelNet40 [53].

optimizer with a weight decay 0.0002, and cosine scheduler with an initial learning rate 0.1. On ScanObjectNN [48], we adopt AdamW optimizer [23] with a weight decay 0.05, and cosine scheduler with an initial learning rate 0.002. We follow the data augmentation in PointMLP [31] and PointNeXt [37] respectively for ModeNet40 and ScanObjectNN datasets. For part segmentation, we simply utilize the same learnable decoder and training settings as CurveNet [54] for a fair comparison.

**Plug-and-play.** For part segmentation and 3D object detection, concurrently running an extra Point-NN to enhance existing models would be expensive in both time and memory. Thus, referring to SN-Adapter [65], we directly adopt the encoders of already trained models to extract point cloud features, and only apply our point-memory bank on top for plug-and-play. In this way, we can also achieve performance improvement by leveraging the complementary knowledge between similarity matching and traditional learnable classification heads.
Let's perform the best-performing value (\(k\)) in trigonometric functions of Point-NN. We fix one of the hyperparameters in Table 6 and 7, we show the influence of two hyperparameters on the performance of Point-NN. We achieve the highest accuracy, which can summarize the local patterns from two different aspects.

### 4. Additional Ablation Study

**Non-Parametric Encoder.** In Table 5, we further investigate other designs at every stage of Point-NN’s non-parametric encoder. As shown, \(k\)-NN performs better than ball query [36] for grouping the neighbors of each center point since the ball query would fail to aggregate valid geometry in some sparse regions with only a few neighboring points. Expanding the feature dimension by concatenating the center and neighboring points can improve the performance by +5%. This is because each point obtains larger receptive fields as the network stage goes deeper and requires higher-dimensional vectors to encode more spatial semantics. For the pooling operation after geometry extraction, we observe applying both max and average pooling achieves the highest accuracy, which can summarize the local patterns from two different aspects.

**Hyperparameters in Trigonometric Functions.** In Table 6 and 7, we show the influence of two hyperparameters in trigonometric functions of Point-NN. We fix one of them to be the best-performing value (\(\alpha\) as 100, \(\beta\) as 500), and vary the other one for ablation. The combination of the magnitude \(\alpha\) and wavelength \(\beta\) control the frequency of the channel-wise sinusoid, and thus determine the raw point encoding for different classification accuracy.

**Point-Memory Bank with Different Sizes.** As default, we construct the feature memory by the entire training-set point clouds. In Table 8, we report how Point-NN performs when partial training samples are utilized for the point-memory bank. As shown, Point-NN can attain 60.1% classification accuracy with only 10% of the training data, and further achieves 70.1% with 40% data, which is comparable to the performance of 100% ratio but consumes less GPU memory. This indicates Point-NN is not sensitive to the memory bank size and can perform favorably with partial training-set data.

### References


[13] Ankit Goyal, Hei Law, Bowei Liu, Alejandro Newell, and Jia Deng. Revisiting point cloud shape classification with a sim-


