Supplementary material for 'RepKPU: Point Cloud Upsampling with Kernel Point Representation and Deformation'

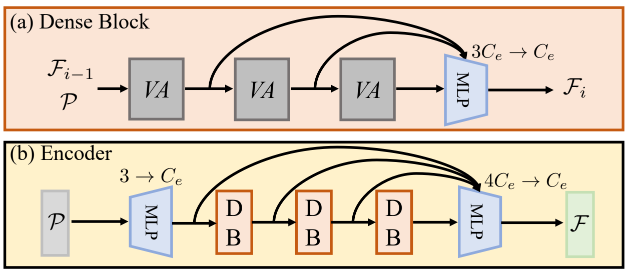

Figure 1. (a) Details of dense block (DB), where VA indicates vector attention mechanism [11]. (b) Details of our encoder.

A. Encoder

We show the complete details of our encoder in Figure 1. There are no downsampling operations in the encoder. The process of vector attention mechanisms [11] is as follows: Given coordinates $\mathcal{P} \in \mathbb{R}^{N\times 3}$ and current features $\mathcal{F}_a \in \mathbb{R}^{N\times C_e}$, we first adopt three linear layers to project $\mathcal{F}_a$ to $\mathcal{Q} \in \mathbb{R}^{N\times C_e}$, $\mathcal{K} \in \mathbb{R}^{N\times C_e}$, and $\mathcal{V} \in \mathbb{R}^{N\times C_e}$, respectively. Then we conduct feature aggregation like:

$$\mathbf{a}_{ij} = \underset{C_e \to C_e}{\mathrm{MLP}_1}(\mathbf{q}_i - \mathbf{k}_j) + \underset{3 \to C_e}{\mathrm{MLP}_2}(\mathbf{p}_i - \mathbf{p}_j), j \in \mathrm{knn}(\mathbf{p}_i), \tag{1}$$

$$\mathbf{f}_i^{'} = \underset{C_e \to C_e}{\mathrm{Conv}}\Big(\sum_{j \in \mathrm{knn}(\mathbf{p}_i)} \frac{\exp(\mathbf{a}_{ij})}{\sum_{j \in \mathrm{knn}(\mathbf{p}_i)} \exp(\mathbf{a}_{ij})} \odot \mathbf{v}_j\Big) + \mathbf{f}_i, \tag{2}$$

where, $\mathbf{q}_i \in \mathcal{Q}$, $\mathbf{k}_j \in \mathcal{K}$, $\mathbf{v}_j \in \mathcal{V}$, $\mathbf{p}_i \in \mathcal{P}$, and $\mathbf{f}_i \in \mathcal{F}_a$. Conv indicates a linear projection. $\mathcal{F}_b = \{\mathbf{f}_i^{'}\}$ is the output of VA operation. All the addition, division, and multiplication operations are element-wise.

We replace vector attention mechanisms with graph convolutions [8] to build another encoder. Instead of using k-nn searching in the feature space, we use it in the geometric space:

$$\mathbf{f}_i^{'} = \underset{C_e \to C_e}{\mathrm{Conv}}\big(\underset{j \in \mathrm{knn}(\mathbf{p}_i)}{\mathrm{MaxPooling}} \underset{2C_e \to C_e}{\mathrm{MLP}}(\mathrm{cat}(\mathbf{f}_i, \mathbf{f}_j - \mathbf{f}_i))\big) + \mathbf{f}_i. \tag{3}$$

We also construct a variant of the encoder based on Set Abstraction [4]:

$$\mathbf{f}_i^{'} = \underset{C_e \to C_e}{\mathrm{Conv}}\big(\underset{j \in \mathrm{knn}(\mathbf{p}_i)}{\mathrm{MaxPooling}} \underset{C_e+3 \to C_e}{\mathrm{MLP}}(\mathrm{cat}(\mathbf{f}_j, \mathbf{p}_j - \mathbf{p}_i))\big) + \mathbf{f}_i. \tag{4}$$

B. Cross-attention Transformer

Before feeding $\mathcal{F}_q$ and $\mathcal{F}_k$ into transformer, we first project them to $\mathcal{F}_{q'} \in \mathbb{R}^{N_q \times C_d}$ and $\mathcal{F}_{k'} \in \mathbb{R}^{N_k \times C_d}$ with two linear layers. We define the input query vectors of the i-th cross-attention layer as $\mathcal{F}_q^i$. Note that $\mathcal{F}_q^0 = \mathcal{F}_{q'}$. Following [7], we then obtain $\mathcal{Q}_i$, $\mathcal{K}_i$, and $\mathcal{V}_i$ via linear projection layers.

$$\mathcal{Q}_i, \mathcal{K}_i, \mathcal{V}_i = \underset{C_d \to C_d}{\mathrm{Conv}_q^i}(\mathcal{F}_q^i), \underset{C_d \to C_d}{\mathrm{Conv}_k^i}(\mathcal{F}_{k'}), \underset{C_d \to C_d}{\mathrm{Conv}_v^i}(\mathcal{F}_{k'}). \tag{5}$$

Next, features are aggregated in a multi-head manner:

$$\mathcal{A}^i = \mathrm{cat}(\mathcal{A}_0^i, \mathcal{A}_1^i, ..., \mathcal{A}_{h-1}^i), \mathcal{A}_j^i = \mathrm{Attention}(\mathcal{Q}_{ij}, \mathcal{K}_{ij}, \mathcal{V}_{ij}), \tag{6}$$

here, h indicates the number of head, and we set $h = 4$. Finally, with residual connections and FFN (*i.e.*, MLP), the i-th cross-attention layer outputs $\mathcal{F}_q^{i+1}$:

$$\mathcal{B}^i = \mathcal{A}^i + \mathcal{F}_q^i, \mathcal{F}_q^{i+1} = \underset{C_d \to C_d}{\mathrm{FFN}}(\mathcal{B}^i) + \mathcal{B}^i. \tag{7}$$

After passing through transformer, KP-Queries are converted into displacement features $\mathcal{F}_d$, where $\mathcal{F}_d = \mathcal{F}_q^3$.

C. Loss Function

Given ground-truth point cloud $\mathcal{P}_{gt} \in \mathbb{R}^{N_{gt} \times 3}$ and upsampled point cloud $\mathcal{P}_u \in \mathbb{R}^{N_u \times 3}$, the CD loss ($\mathcal{L}_{cd}$) is formulated as:

$$\mathcal{L}_{cd} = \frac{1}{N_u} \sum_{a \in \mathcal{P}_u} \min_{b \in \mathcal{P}_{gt}} \|a - b\| + \frac{1}{N_{gt}} \sum_{b \in \mathcal{P}_{gt}} \min_{a \in \mathcal{P}_u} \|b - a\|. \tag{8}$$

For each center point $\mathbf{p} \in \mathcal{P}$, we will search for its local positions $\mathcal{P}_r$ and subsequently get the positions of RepKPoints $\mathcal{P}_k$. If unrestricted, the positions of RepKPoints will be pulled away from the input points, and the model subsequently cannot perceive any geometry. The same phenomenon can also be seen in KPConv [6]. To alleviate this issue, we use the fitting loss ($\mathcal{L}_{fit}$) to enforce kernel points to fit the local region:

$$\mathcal{L}_{fit} = \alpha \times \frac{1}{N} \sum_{\mathbf{p} \in \mathcal{P}} \sum_{\mathbf{p}_k \in \mathcal{P}_k} \min_{\mathbf{p}_r \in \mathcal{P}_r} \left(\frac{\|\mathbf{p}_k - (\mathbf{p}_r - \mathbf{p})\|}{\sigma} \right)^2. \tag{9}$$

Additionally, we use repulsive loss ($\mathcal{L}_{rep}$) to avoid kernel points' receptive areas overlapping:

$$\mathcal{L}_{rep}=\frac{\beta}{N \times N_k \times (N_k-1)} \sum_{\mathbf{p} \in \mathcal{P}} \sum_{\mathbf{p}_k^i \in \mathcal{P}_k} \sum_{i \neq j} \max\left(0, 1-\frac{\|\mathbf{p}_k^i-\mathbf{p}_k^j\|}{\sigma}\right)^2. \tag{10}$$

$\mathcal{L}_{rep}$ and $\mathcal{L}_{fit}$ facilitate model optimization, but they also harm the flexibility of RepKPoints. To strike a balance, we set coefficients α and β to 0.1.

D. Robusteness Test

We report detailed quantitative results of the robustness test in Table 1, which corresponds to Table 3 in the main paper.

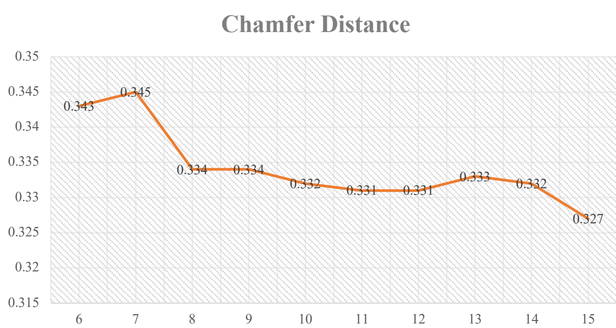

Figure 2. Impacts of the kernel point number in terms of Chamfer Distance ($\times 10^3$) on PU1K dataset.

E. Supplemental Ablation study

We show the impacts of kernel point number in RepKPoints (*i.e.*, N_k) in Figure 2. According to the results, we set N_k to 15 to achieve the best performance.

F. Parameters and Latency

We report the numbers of parameters, training-time memory usage, and inference speeds (in terms of latency) in Table 2. It is well known that latency and training-time memory are not proportionate to the number of parameters. All models in this table was trained on a single Nvidia 1080Ti with a batch size of 32, our model takes significantly less memory. The inference speed was evaluated on a single Nvidia 1080Ti with a batch size of 1.

G. More Visual Results

As discussed in the experimental section of the main paper, visual results have the same importance as quantitative results. Therefore, we provide more visual results in the supplementary material. Figure 3 shows the upsampling results at three different input resolutions and patterns on the robustness test dataset. Figure 4 shows the visual results of arbitrary-scale upsampling compared with Grad-PU. We also select several shapes of PU1K dataset and show them in Figure 5.

References

[1] Yun He, Danhang Tang, Yinda Zhang, Xiangyang Xue, and Yanwei Fu. Grad-pu: Arbitrary-scale point cloud upsampling via gradient descent with learned distance functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5354–5363, 2023. 3

[2] Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-gan: a point cloud upsampling adversarial network. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7203–7212, 2019. 3

[3] Ruihui Li, Xianzhi Li, Pheng-Ann Heng, and Chi-Wing Fu. Point cloud upsampling via disentangled refinement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 344–353, 2021. 3

[4] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 1

[5] Guocheng Qian, Abdulellah Abualshour, Guohao Li, Ali Thabet, and Bernard Ghanem. Pu-gcn: Point cloud upsampling using graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11683–11692, 2021. 3

[6] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019. 1

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1

[8] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019. 1

[9] Wang Yifan, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. Patch-based progressive 3d point set upsampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5958–5967, 2019. 3

[10] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2790–2799, 2018. 3

[11] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of*

Table 1. Detailed quantitative results of the robustness test. The best and second-best results are highlighted in bold and underline, respectively.

Input resolution	Methods	Uniform inputs			Noisy inputs			Random inputs			Noisy + Random		
		CD $\times 10^3$	HD $\times 10^3$	P2F $\times 10^3$	CD $\times 10^3$	HD $\times 10^3$	P2F $\times 10^3$	CD $\times 10^3$	HD $\times 10^3$	P2F $\times 10^3$	CD $\times 10^3$	HD $\times 10^3$	P2F $\times 10^3$
2048 points	PU-Net [10]	0.444	3.931	4.578	0.579	5.980	9.769	0.439	4.957	4.117	0.632	7.046	9.636
	MPU [9]	0.280	3.910	2.842	0.445	6.750	7.166	0.324	4.993	3.021	0.497	6.725	7.529
	PU-GAN [2]	$\underline{0.260}$	4.707	1.991	0.514	7.621	9.063	**0.256**	4.528	2.314	0.541	7.541	9.433
	Dis-PU [3]	0.264	4.411	2.020	$\underline{0.418}$	6.964	6.729	0.278	**3.773**	2.172	$\underline{0.464}$	**5.796**	8.081
	PU-GCN [5]	0.278	3.579	2.549	0.435	5.121	7.076	0.316	4.201	2.820	0.471	6.000	7.407
	Grad-PU [1]	0.264	**2.623**	$\underline{1.982}$	0.440	**4.393**	**6.439**	0.480	6.285	**2.119**	0.601	7.223	**6.445**
	RepKPU	**0.248**	$\underline{2.880}$	**1.906**	**0.404**	$\underline{4.817}$	$\underline{6.721}$	$\underline{0.268}$	$\underline{3.850}$	$\underline{2.147}$	**0.449**	$\underline{5.892}$	$\underline{7.020}$
1024 points	PU-Net [10]	0.933	7.648	7.252	1.007	9.042	11.401	0.785	8.572	6.461	1.013	10.807	10.890
	MPU [9]	0.597	5.705	4.502	0.742	7.870	7.912	0.666	9.922	4.770	0.900	11.032	8.380
	PU-GAN [2]	0.569	6.259	3.448	0.834	8.858	9.778	**0.580**	8.994	3.973	0.883	11.926	10.480
	Dis-PU [3]	$\underline{0.548}$	5.642	3.355	**0.679**	7.508	6.999	0.603	8.778	3.699	$\underline{0.783}$	$\underline{10.237}$	7.373
	PU-GCN [5]	0.595	5.277	4.133	0.731	7.272	7.719	0.680	$\underline{8.198}$	4.555	0.885	10.275	8.111
	Grad-PU [1]	0.563	**4.989**	$\underline{3.349}$	0.705	**6.496**	6.887	0.947	11.583	$\underline{3.597}$	1.051	12.657	**6.887**
	RepKPU	**0.539**	$\underline{5.086}$	**3.178**	**0.679**	$\underline{7.156}$	**6.857**	$\underline{0.594}$	**7.706**	**3.520**	**0.770**	**9.786**	$\underline{7.095}$
512 points	PU-Net [10]	1.792	13.327	11.881	1.769	12.904	14.660	1.722	15.610	10.122	1.973	18.786	13.512
	MPU [9]	1.221	11.512	7.197	1.295	13.140	9.721	1.475	15.278	7.446	1.622	18.720	10.286
	PU-GAN [2]	1.176	10.839	6.114	1.388	13.296	11.405	**1.202**	14.816	6.708	1.547	18.487	12.318
	Dis-PU [3]	1.073	10.671	5.707	$\underline{1.190}$	12.627	8.464	1.223	$\underline{13.272}$	6.056	$\underline{1.419}$	17.017	8.963
	PU-GCN [5]	1.197	$\underline{9.179}$	6.703	1.297	$\underline{11.142}$	9.450	1.416	13.566	7.179	1.620	$\underline{16.774}$	10.032
	Grad-PU [1]	$\underline{1.094}$	**8.707**	**5.464**	1.256	**9.945**	**8.117**	1.946	19.889	**5.889**	2.048	20.559	**8.310**
	RepKPU	**1.080**	10.022	$\underline{5.476}$	**1.189**	11.312	$\underline{8.207}$	$\underline{1.215}$	**13.224**	$\underline{5.966}$	**1.345**	**16.720**	$\underline{8.585}$
Average	PU-Net [10]	1.056	8.302	7.904	1.118	9.309	11.943	0.982	9.713	6.900	1.206	12.213	11.346
	MPU [9]	0.699	7.042	4.847	0.827	9.253	8.266	0.822	10.064	5.079	1.006	12.159	8.732
	PU-GAN [2]	0.668	7.268	3.851	0.912	9.925	10.082	**0.679**	9.446	4.332	0.990	12.651	10.744
	Dis-PU [3]	$\underline{0.628}$	6.908	3.694	$\underline{0.762}$	9.033	7.397	0.701	$\underline{8.608}$	3.976	$\underline{0.889}$	$\underline{11.017}$	8.139
	PU-GCN [5]	0.690	6.012	4.462	0.821	7.845	8.082	0.804	8.655	4.851	0.992	11.016	8.517
	Grad-PU [1]	0.640	**5.440**	$\underline{3.598}$	0.800	**6.945**	**7.148**	1.124	12.586	**3.868**	1.233	13.480	**7.214**
	RepKPU	**0.622**	$\underline{5.996}$	**3.520**	**0.757**	$\underline{7.762}$	$\underline{7.262}$	$\underline{0.692}$	**8.260**	$\underline{3.878}$	**0.855**	**10.799**	$\underline{7.567}$

Table 2. The number of parameters, training memory usage, and latency.

Methods	Params *kb*	Training Memory *G*	Latency *s*
PU-Net [10]	814.3	7.596	0.314
MPU [9]	76.2	7.050	0.303
PU-GAN [2]	684.2	7.058	0.403
Dis-PU [3]	1047.0	7.060	0.579
PU-GCN [5]	76.0	7.562	0.317
Grad-PU [1]	**67.1**	7.144	0.306
RepKPU	1458.6	**5.860**	**0.215**

the IEEE/CVF international conference on computer vision, pages 16259–16268, 2021. 1

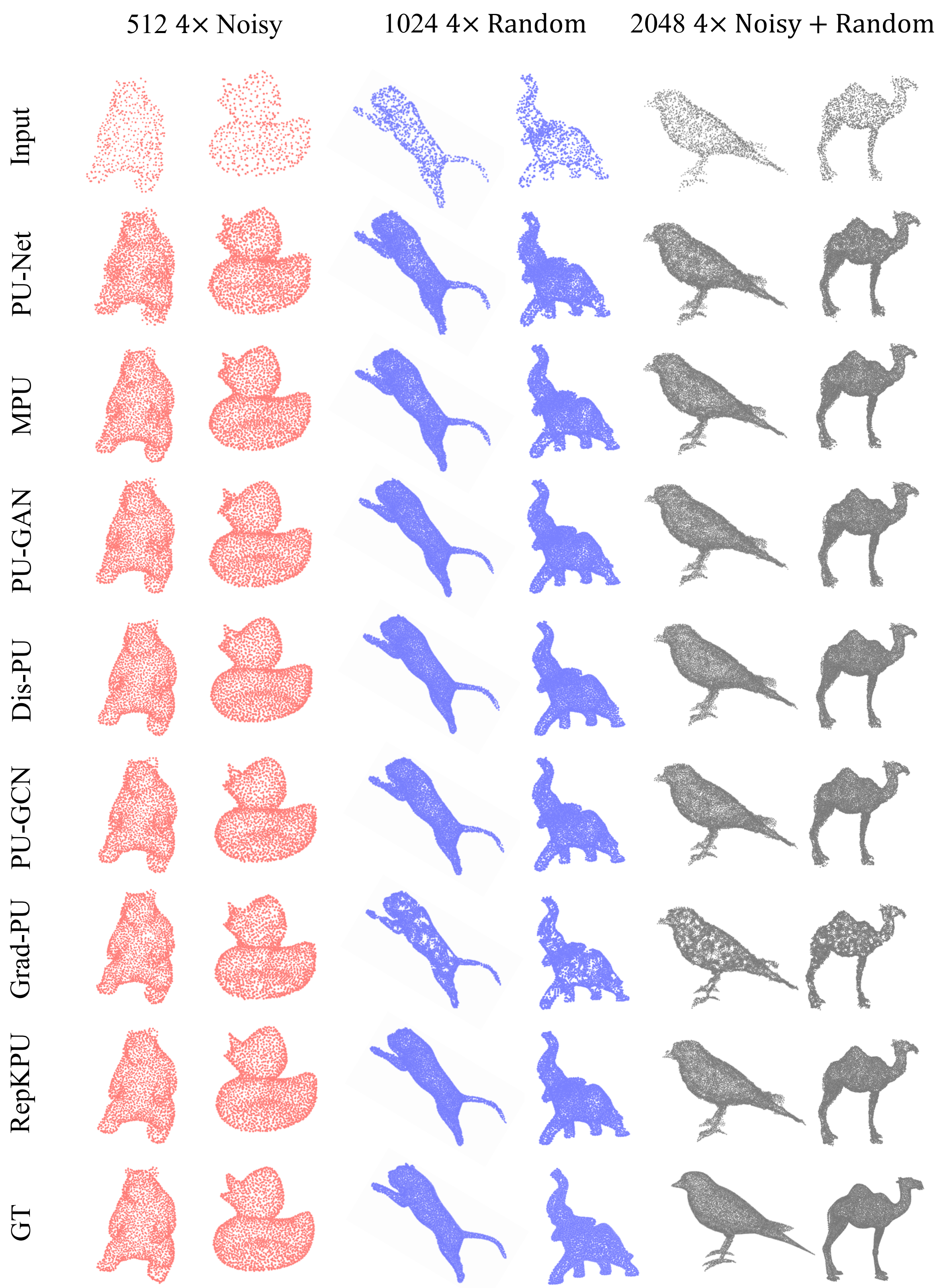

Figure 3. Visual results of robustness test. We show upsampling results at three different input resolutions (*i.e.*, 512, 1,024, 2,048). For three different resolutions, we select noisy, random, and noisy + random patterns, respectively.

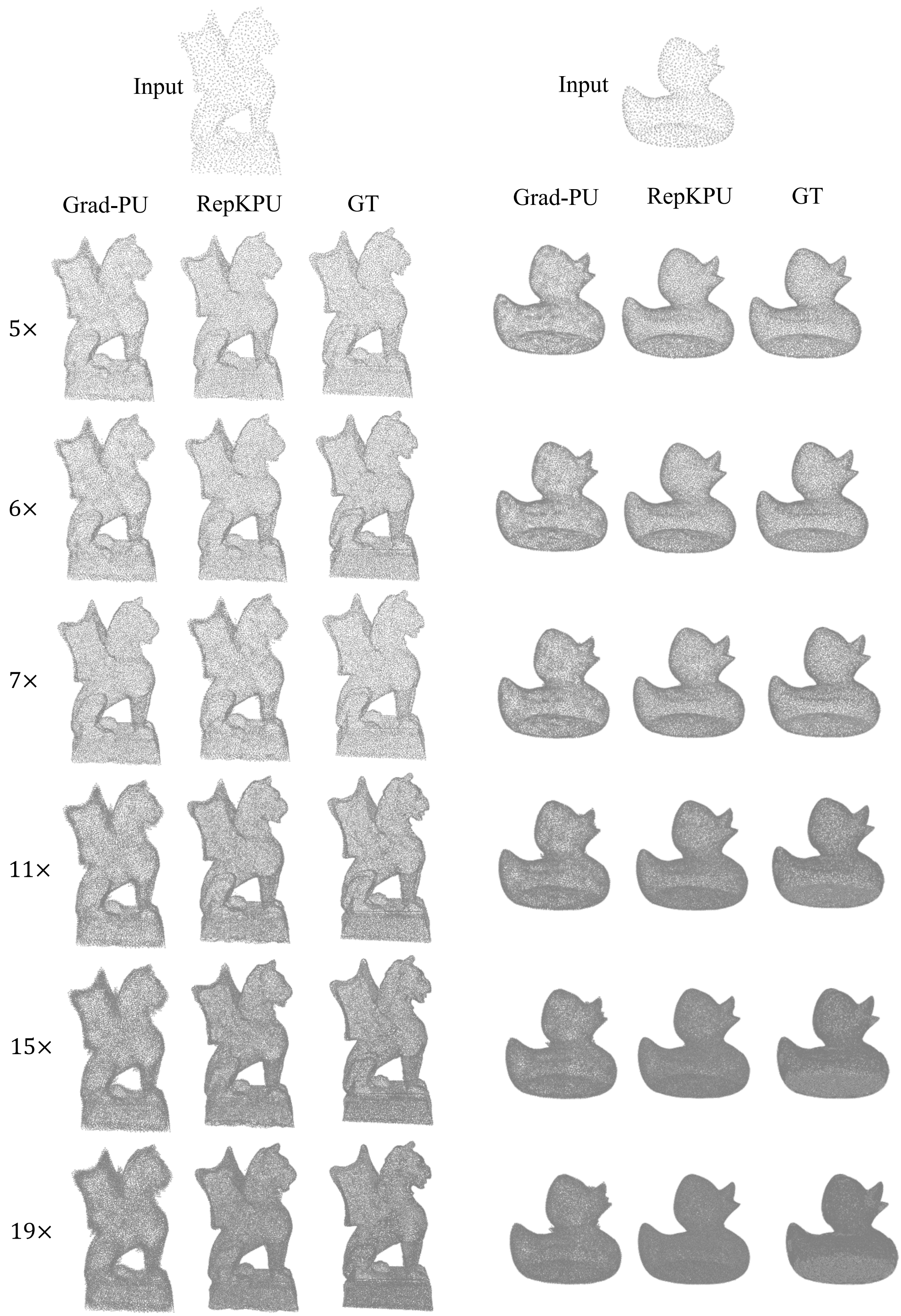

Figure 4. Visual results of arbitrary-scale upsampling compared with Grad-PU. The upsampling rates are 5, 6, 7, 11, 15, and 19, respectively.

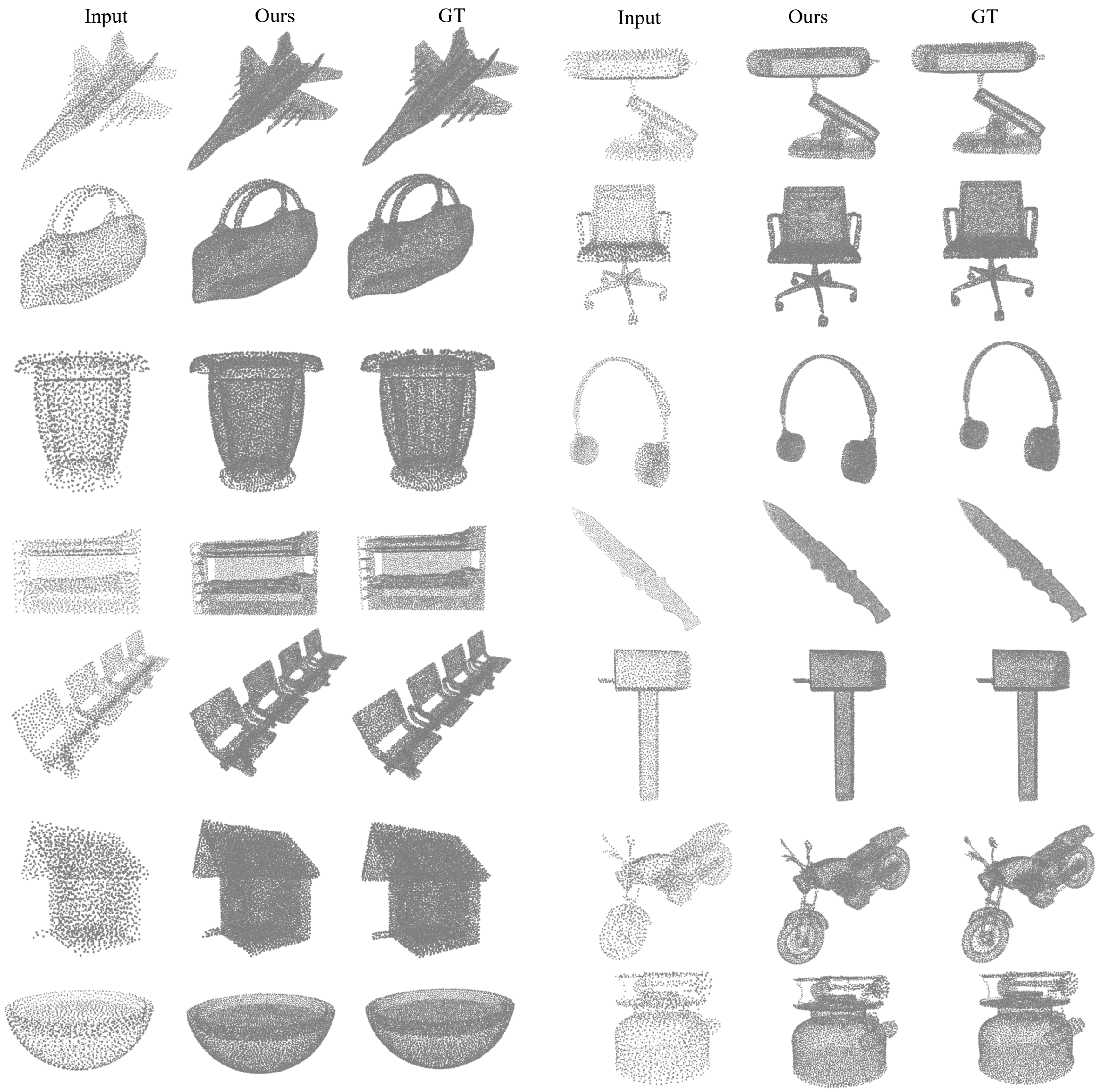

Figure 5. Visual results of RepKPU on PU1K dataset.