Grad-PU: Arbitrary-Scale Point Cloud Upsampling via Gradient Descent with Learned Distance Functions (Supplementary Material)

Yun He[1] Danhang Tang[2] Yinda Zhang[2] Xiangyang Xue[1] Yanwei Fu[1]
[1] Fudan University [2] Google

In this supplementary material, we provide additional implementation details, ablation studies and qualitative results.

1. Implementation Details

Additional details about hyperparameter settings and detailed network architecture are elucidated in this section.

1.1. Hyperparameters

In the P2PNet, we set the feature dimension $d = 32$, the nearest neighbor number $k = 16$, and the standard deviation of Gaussian noise $\sigma = 0.02$. For training, we use random perturbation, rotation and scaling for data augmentation, the same as [10]. For testing, we choose the iteration number $T = 10$ to balance the computational cost and performance.

Our model is implemented with Pytorch [8], trained on a NVIDIA TITAN X GPU for 60 epochs with a batch size of 32. We use the Adam optimizer [5] with an initial learning rate of 1e-3 and a decay factor of 0.5 every 20 epochs.

1.2. Detailed Network Architecture

The detailed architecture of our P2PNet is shown in Fig 1. In the feature extractor, we first employ an MLP to project the initial interpolated point cloud P_I to a higher dimension space. And then stack three dense blocks with intra-block dense connection [4], where each block comprises four MLPs and three P3DConv layers. The MLP is used to reduce feature channel, and the P3DConv layer is utilized for local feature capture. Lastly, we obtain the extracted local features $\{l^0, l^1, l^2, l^3\}$ as well as global feature g. In the distance regressor, we first get the point-wise local features $\{l_p^0, l_p^1, l_p^2, l_p^3\}$ of query point p by feature interpolation [9], and then apply a four-layer MLP to regress the point-to-point distance. The formula of feature interpolation [9] at point p is listed below:

$$l_p^s = \frac{\sum_{k=1}^{3} w_{p_k} l_{p_k}^s}{\sum_{k=1}^{3} w_{p_k}}, where\ w_{p_k} = \frac{1}{||p - p_k||_2} \tag{1}$$

where l_p^s ($s \in \{0, 1, 2, 3\}$) are the interpolated multi-scale local features, p_k ($k \in \{1, 2, 3\}$) are the three nearest-neighbors of query point p in the initial interpolated point cloud, and $||p - p_k||_2$ is the distance between p and p_k.

2. Additional Ablation Studies

In this section, we conduct more ablation experiments to validate our choices of iteration number T, regressor input, training scheme with Gaussian noise and location refinement strategy. All the evaluations are done on the PU1K dataset with 4× upsampling.

Initial Point Cloud to Be Updated. We utilize the interpolated result P_I as the initial point cloud, and then move it towards ground truth point cloud by distance minimazation process. To further verify the robustness of our refinement to different initialization, we also substitute P_I with the randomly initialized point cloud P_R (consistent with the third row of Tab 7 in main paper). As Tab 1 shows, even for the random initialization, our refinement can still achieve comparable performance by more iterations.

Initial Point Cloud to Be Updated	CD↓ 10^{-3}	HD↓ 10^{-3}	P2F↓ 10^{-3}
P_R ($T = 10$)	0.648	5.474	3.523
P_R ($T = 20$)	0.521	4.653	2.637
P_R ($T = 30$)	0.485	4.346	2.326
P_R ($T = 40$)	0.461	4.231	2.157
$\boldsymbol{P_I (T = 10)}$	**0.404**	**3.732**	**1.474**

Table 1. 4× comparative results on the PU1K dataset with different initial point clouds and iteration number T. Our refinement is robust to various initilization.

Regressor Input. In our full model, we regress the point-to-point distance of each query point using the concatenation of coordinate p, interpolated local features

Yun He and Xiangyang Xue are with the School of Computer Science, Fudan University.

Yanwei Fu is with the School of Data Science, Fudan University. He is also with Shanghai Key Lab of Intelligent Information Processing, and Fudan ISTBI–ZJNU Algorithm Centre for Brain-inspired Intelligence, Zhejiang Normal University, Jinhua, China.

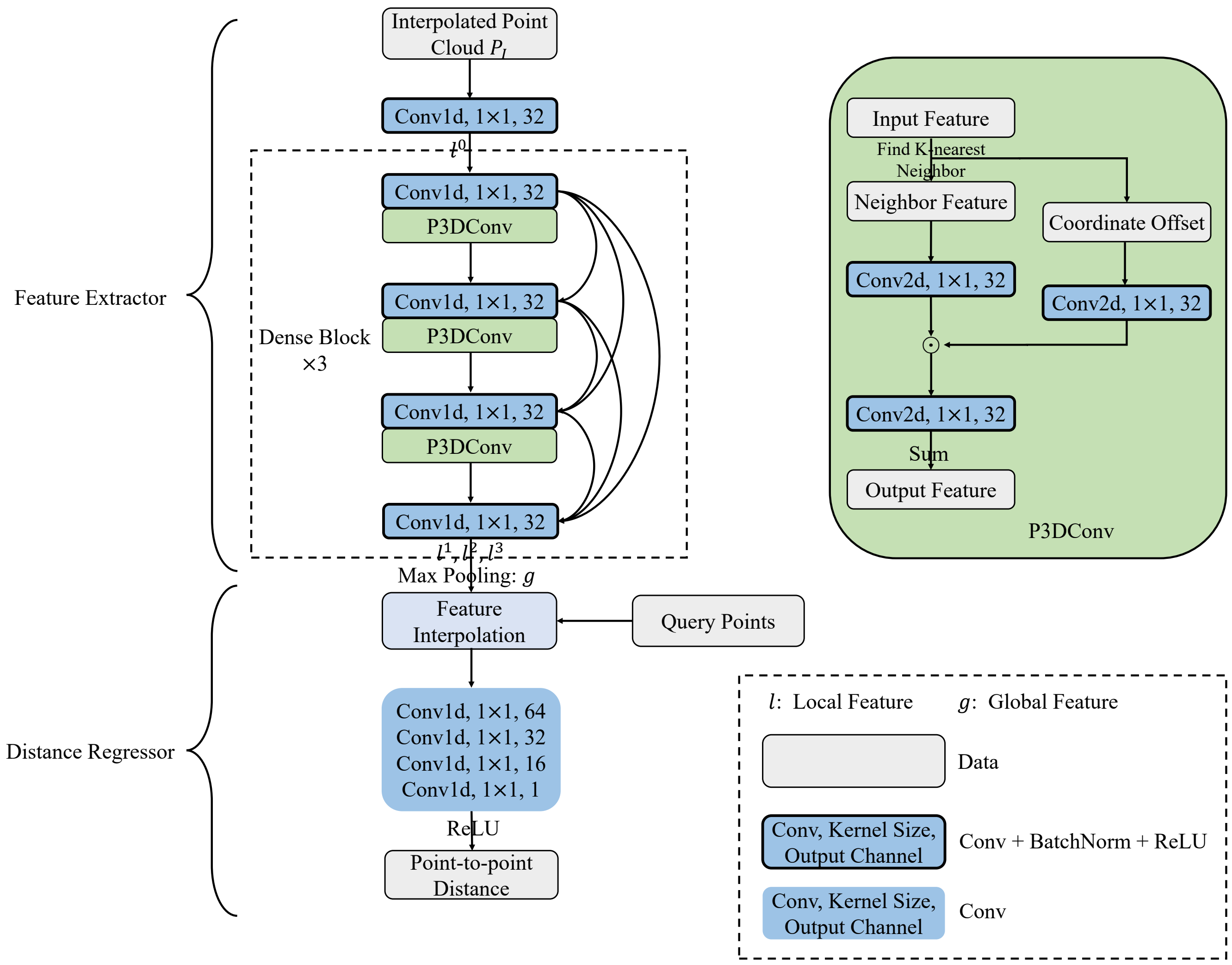

Figure 1. The detailed structure of our P2PNet.

$\{l_p^0, l_p^1, l_p^2, l_p^3\}$ and global feature g. We here analyze the impact of each type of feature on the final performance. We train models with either local feature or global feature, and show their performances in Tab 2. Both cases perform worse than our full model, which uses both features.

Regressor Inputs	CD ↓ 10^{-3}	HD ↓ 10^{-3}	P2F ↓ 10^{-3}
(p, g)	0.729	8.439	2.324
$(p, l_p^0, l_p^1, l_p^2, l_p^3)$	0.423	4.011	1.652
$\boldsymbol{(p, l_p^0, l_p^1, l_p^2, l_p^3, g)}$	**0.404**	**3.732**	**1.474**

Table 2. 4× comparative results on the PU1K dataset with different regressor inputs. Utilizing both local features and global feature is clearly more effective.

Training Scheme with Gaussian Noise. During training, we jitter the interpolated points $p \in P_I$ with Gaussian noise to obtain query points, which simulates the iterative optimization process of inference. To verify the effectiveness of this training scheme, we train our P2PNet with or without Gaussian noise, while sharing the same testing process. As shown in Tab 3, using Gaussian noise for training achieves superior performance, since it simulates the upsampling error from not just the initial interpolated point cloud but all iterations, and also increases the smoothness and continuity of learned distance functions.

Training	CD ↓ 10^{-3}	HD ↓ 10^{-3}	P2F ↓ 10^{-3}
w/o Gaussian Noise	0.520	5.368	1.729
w Gaussian Noise	**0.404**	**3.732**	**1.474**

Table 3. 4× comparative results on the PU1K dataset with or without Gaussian noise. Using Gaussian noise for training significantly improves the upsampling performance.

Location Refinement Strategy. After midpoint interpolation, we iteratively move the interpolated point p towards the ground truth position, guided by the estimated point-to-point distance $P2PNet(p)$ and predefined step size λ, as formulated below:

$$p^{t+1} = p^t - \lambda \nabla P2PNet(p^t) \tag{2}$$

where $t \in [0, T-1]$, T is the predefined iteration number.

And Chibane *et al.* [2] also propose to project point by moving it the predicted distance along the normalized negative gradient, as listed below:

$$p^{t+1} = p^t - P2PNet(p^t)\frac{\nabla P2PNet(p^t)}{||\nabla P2PNet(p^t)||_2} \tag{3}$$

We use Eq 2 and Eq 3 as the refinement strategy respectively, and we also fine-tune the iteration number T for Eq 3 to achieve the best performance, as shown in Tab 4. While Eq 3 requires that the estimated distance $P2PNet(p)$ and direction $-\nabla P2PNet(p)$ should be both accurate enough to achieve a good performance. Our Eq 2 only requires the accuracy of the direction. And once the direction is precisely predicted, the convergence of the interpolated point can be guaranteed, given sufficient iterations and a suitable step size [7].

Refinement Strategies	CD$\downarrow$ 10^{-3}	HD$\downarrow$ 10^{-3}	P2F$\downarrow$ 10^{-3}
Eq 3	0.872	8.650	2.541
Eq 2	**0.404**	**3.732**	**1.474**

Table 4. $4\times$ comparative results on the PU1K dataset with different refinement strategies. Our Eq 2 achieves significantly better performance.

3. Additional Qualitative Results

In this section, we show some more qualitative results on the PU-GAN [6], PU1K [10], ScanObjectNN [12] and KITTI [3] datasets, which further validate that our method achieves superior upsampling quality. Specifically, in Fig 2 and Fig 3, we provide more visual comparisons with previous SOTA methods on the PU-GAN dataset, under $4\times$ and $16\times$ settings respectively. In Fig 4, we visualize the upsampled results with various upsampling rates after one-time training. In Fig 5, we present qualitative comparisons on the PU1K dataset with $4\times$ setting. In Fig 6 and Fig 7, we employ added noise and different input sizes to verify the robustness of our method. In Fig 8 and Fig 9, we display more qualitative results on the real-scanned inputs. Consistent with the main paper, we use CD and HD to represent Chamfer distance and Hausdorff distance respectively, while their units are both 10^{-3}.

References

[1] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics*, 5(4):349–359, 1999. 6

[2] Julian Chibane, Gerard Pons-Moll, et al. Neural unsigned distance fields for implicit function learning. *Advances in Neural Information Processing Systems*, 33:21638–21652, 2020. 3

[3] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 3

[4] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[6] 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

[7] Shitong Luo and Wei Hu. Score-based point cloud denoising. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4583–4592, 2021. 3

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 1

[9] 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

[10] 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. 1, 3

[11] Shi Qiu, Saeed Anwar, and Nick Barnes. Pu-transformer: Point cloud upsampling transformer. *arXiv preprint arXiv:2111.12242*, 2021. 5

[12] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. 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*, pages 1588–1597, 2019. 3

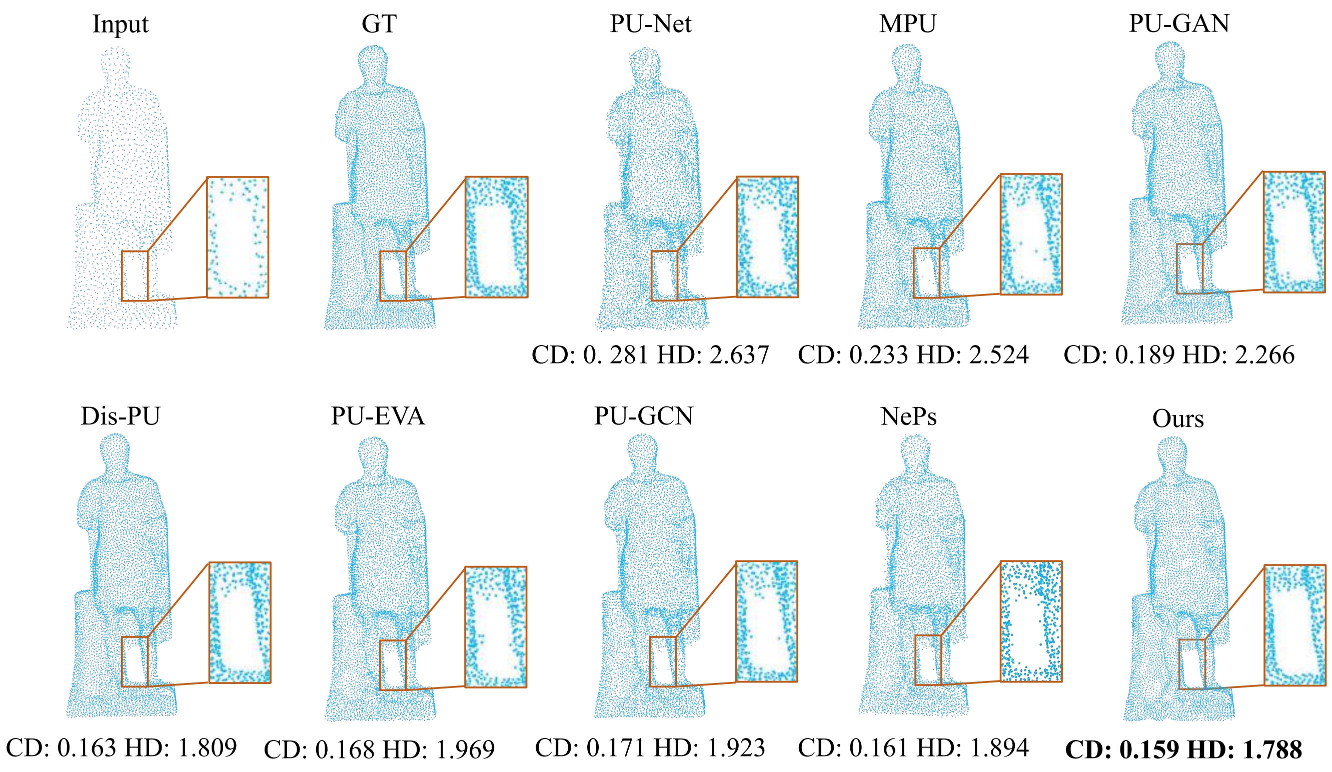

Figure 2. 4× upsampled results on the PU-GAN dataset. Our method produces much less outliers, more smooth surfaces and more fine-grained details.

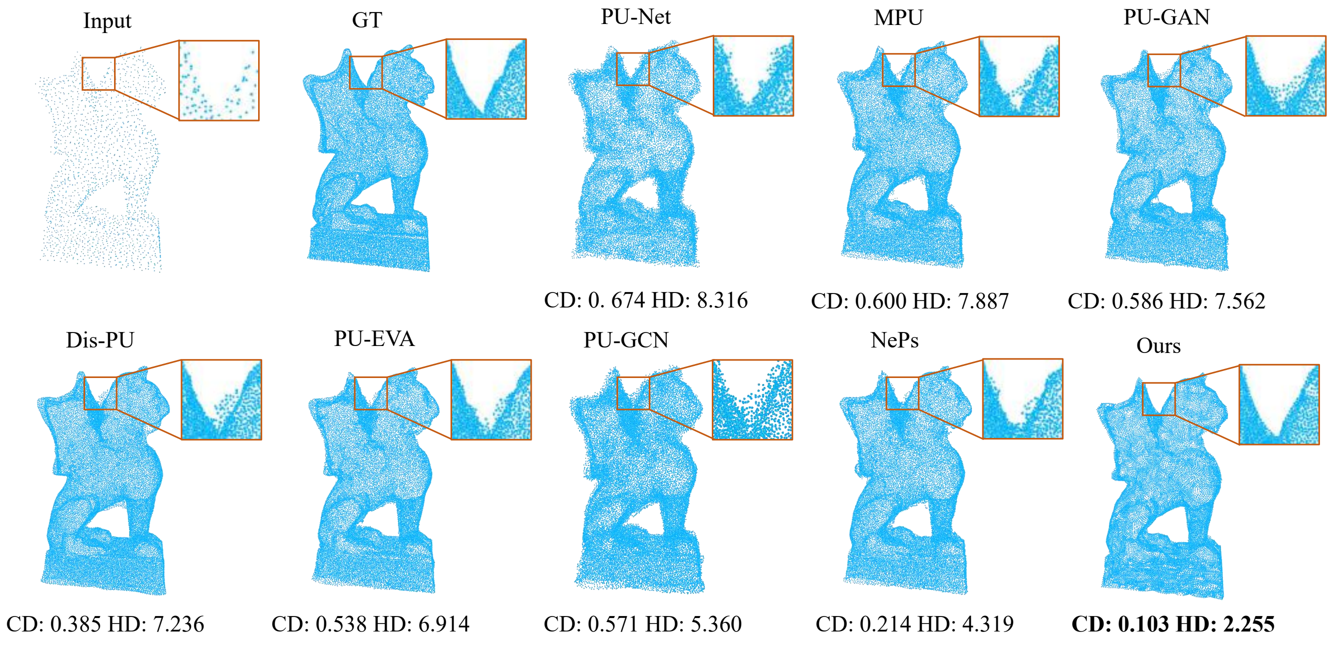

Figure 3. 16× upsampled results on the PU-GAN dataset. Our method produces much less outliers, more smooth surfaces and more fine-grained details.

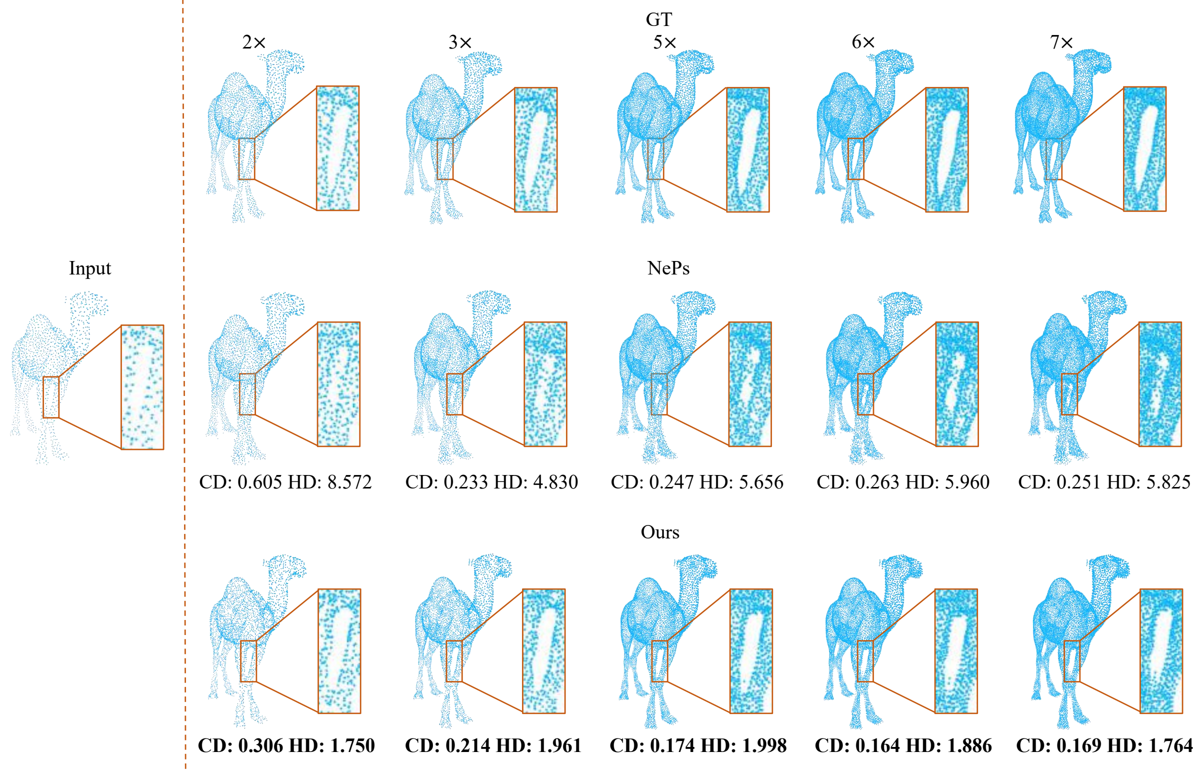

Figure 4. Qualitative comparisons on the PU-GAN dataset with upsampling rate $R \in \{2, 3, 5, 6, 7\}$. Note that we use the same trained model for different upsampling rates. Our method obviously achieves superior performance across the full range of upsampling rates.

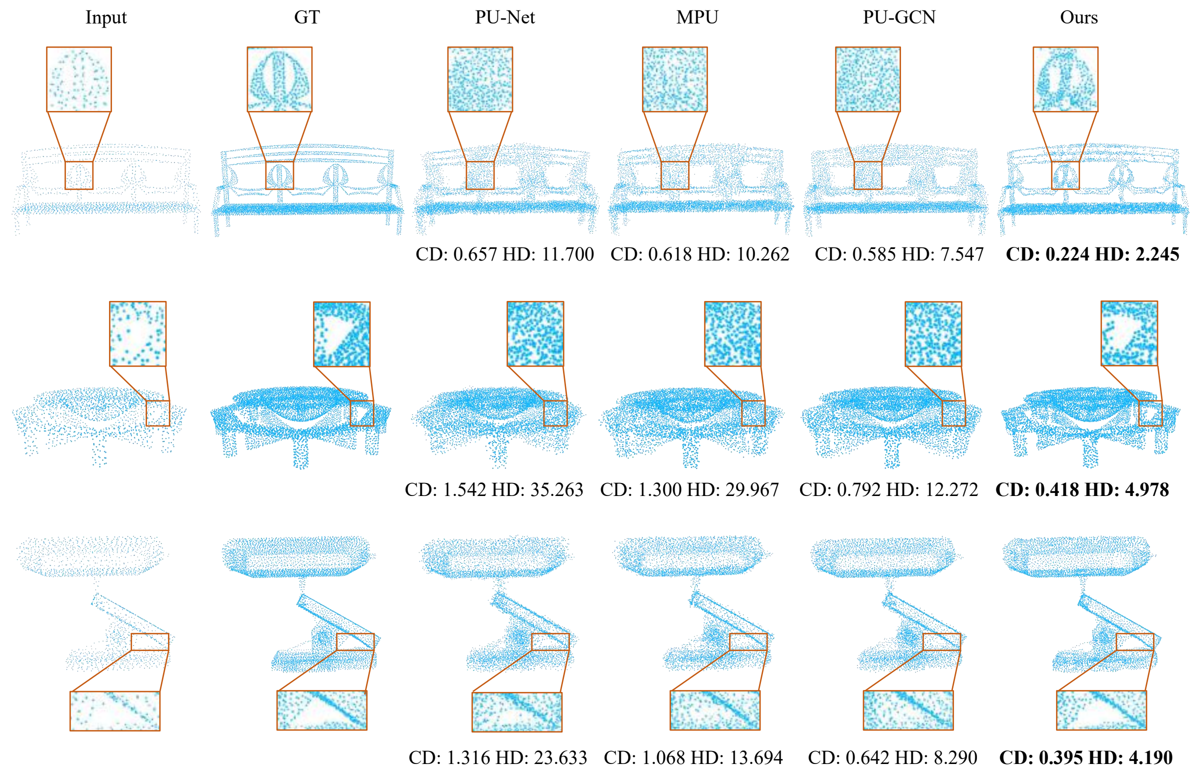

Figure 5. $4\times$ qualitative comparisons on the PU1K dataset, where the visual results of PU-Transformer [11] are not listed here since its codes have not been released so far. Our results preserve more fine-grained details and produce much less outliers.

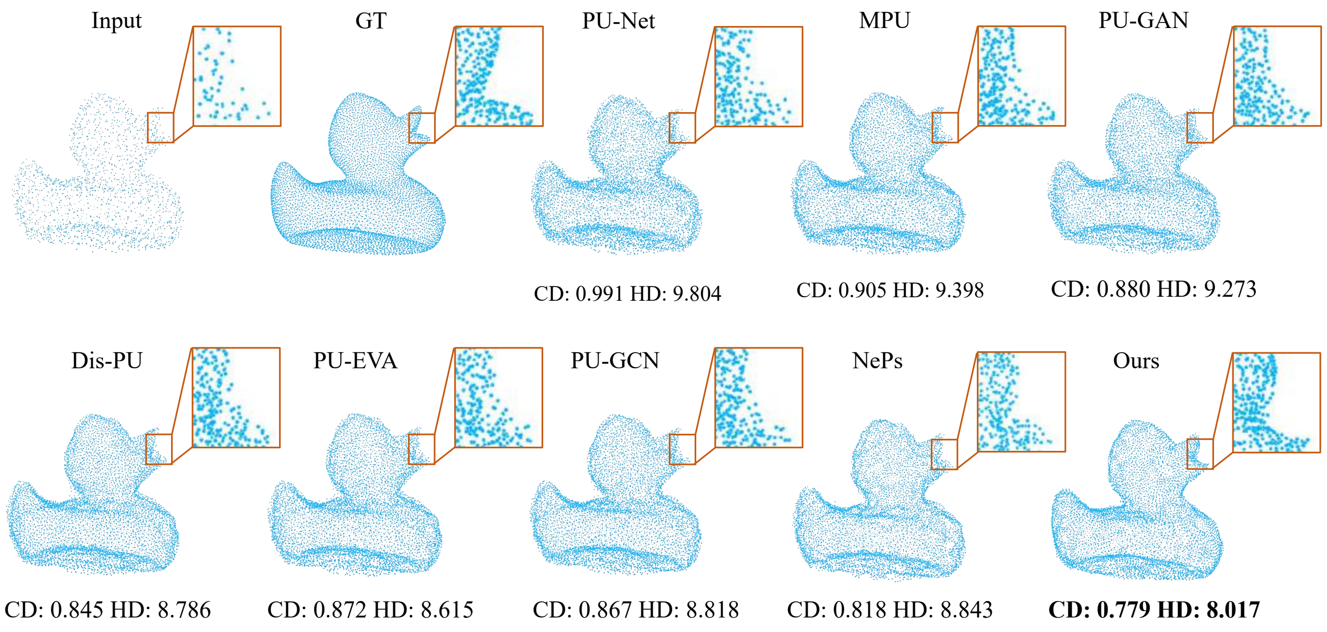

Figure 6. 4× qualitative results on the PU-GAN dataset with noise level $\tau = 0.02$. Our method generates cleaner and smoother point cloud, while preserving more details.

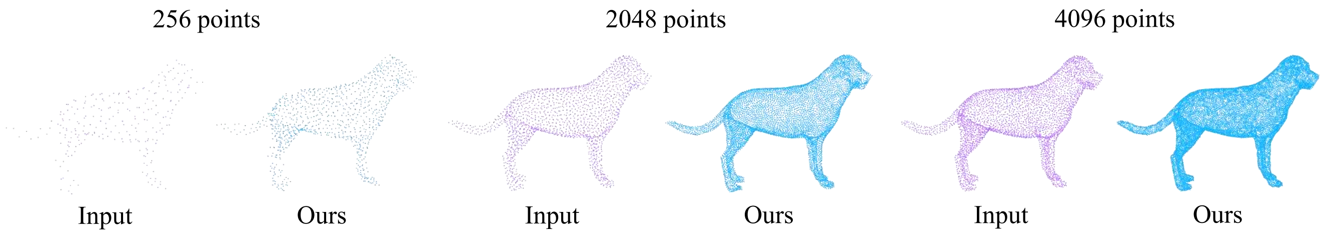

Figure 7. 4× upsampled results by our method with different input sizes. Our method consistently achieves high upsampling quality even when the input is extremely sparse.

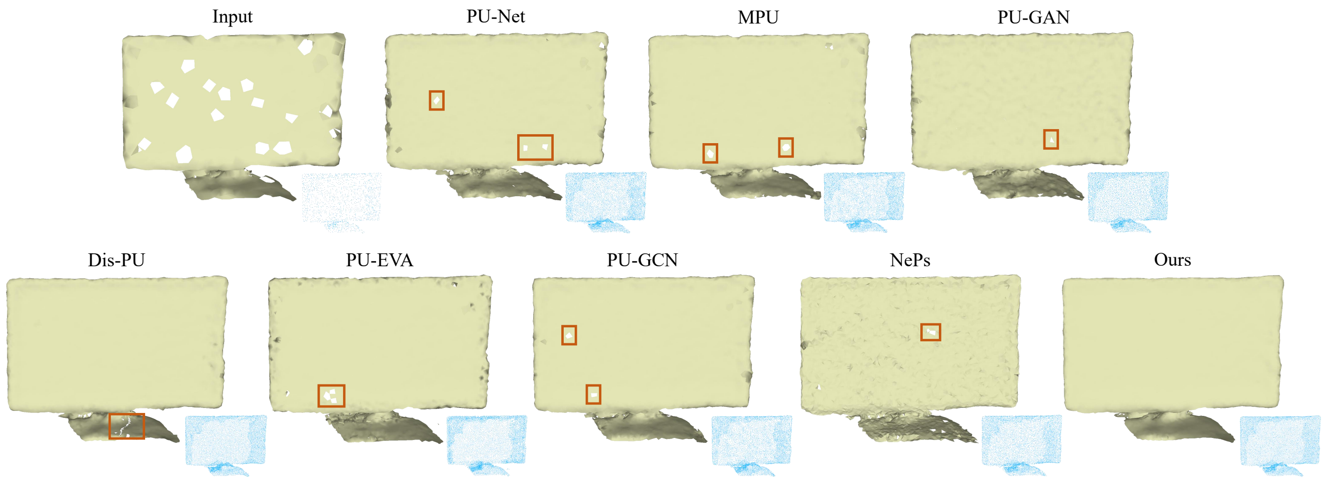

Figure 8. 4× upsampled results on the ScanObjectNN dataset, and the meshes are reconstructed by BallPivoting algorithm [1]. Our method clearly generates more complete, smooth and faithful mesh and point cloud, while other methods tend to keep the holes.

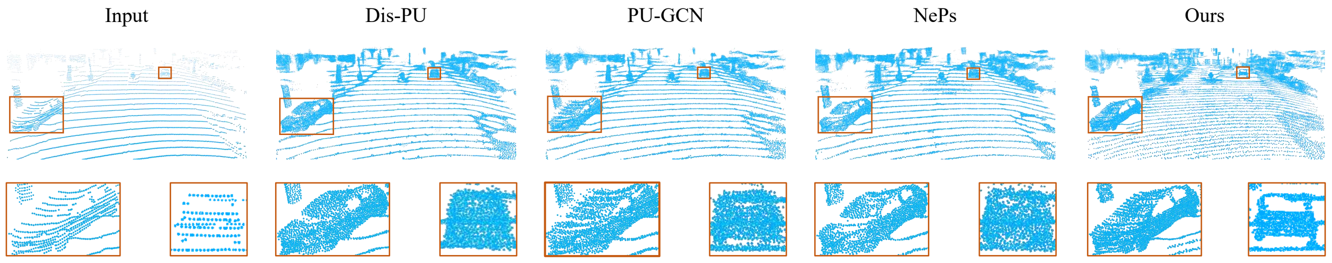

Figure 9. $4\times$ upsampled results on the KITTI dataset. Our result not only retains more fine-grained details but also fills the gaps between LiDAR fibers.