Supplementary materials for DiffPMAE

Yanlong Li[1], Chamara Madarasingha[2], and Kanchana Thilakarathna[1]

[1] The University of Sydney
yali8838@uni.sydney.edu.au kanchana.thilakarathna@sydney.edu.au
[2] University of New South Wales
c.kattadige@unsw.edu.au

1 Further Evaluations: Real-world scanned dataset

To evaluate the generalization ability of our work, we perform extra experiments on the ScanObjectNN dataset [4], which contains 2902 unique 3D point cloud objects in 15 categories scanned from real-world objects. We evaluate our model separately on the main split of the ScanObjectNN dataset with background and without background. We take three experiments with different settings of *DiffPMAE* on both with background split and without background split: *i*) **pre-trained model**, which is the pre-trained on ShapeNet dataset with default configurations (refer to Section 4.1 in our main paper); *ii*) **train from sketch model**, the *DiffPMAE* with default configurations trained on ScanObjectNN training set from sketch; *iii*) **fine-tuned on ScanObjectNN**, the pre-trained *DiffPMAE* with extra training on ScanObjectNN training set with default configurations. We show the evaluation results with a background in Table 1; it reports that the pre-trained *DiffPMAE* model can achieve competitive generation results even without any fine-tuning on ScanObjectNN dataset. Fig. 1 visualizes generated results of three *DiffPMAE* models. Table 2 reports the evaluation results without background. Due to objects in this split without background, some objects are incomplete, which is more complex than the main split with the background. Hence, *DiffPMAE* models perform a bit worse than the split with the background. We visualized the generated results of different *DiffPMAE* models in Fig. 2. Overall, *DiffPMAE* can achieve competitive performance on the more complex real-world dataset even without fine-tuning. Those experiments demonstrate the generalization ability of *DiffPMAE*.

Models	MMD CD ($\times 10^{-3}$)	JSD ($\times 10^{-3}$)	HD ($\times 10^{-2}$)
pre-trained model	1.120	5.814	3.417
train from sketch model	1.685	119.881	4.080
fine-tuned with pre-trained model	1.244	9.505	3.535

Table 1: Reconstruction results for *DiffPMAE* on ScanObjectNN dataset main split with background.

Models	1-NN CD ($\times 10^{-3}$)	JSD ($\times 10^{-3}$)	HD ($\times 10^{-2}$)
pre-trained model	1.131	5.934	3.355
train from sketch model	1.300	197.363	3.486
fine-tuned with pre-trained model	1.314	71.022	3.693

Table 2: Reconstruction results for *DiffPMAE* on ScanObjectNN dataset main split without background.

2 Further Evaluation: Validation of Completion on PCN dataset

Metrics	PCN	SnowFlake	P2C [1]	Ours
MMD CD	5.22	2.32	1.22	1.49

Table 3: Comparison on PCN dataset. MMD CD results are $\times 10^{-3}$.

In Table. 3, we also show the comparison with other models on PCN dataset [5]. Performance of DiffPMAE and P2C are close but P2C is slightly better than ours. We emphasize that completion is only one of our downstream tasks, therefore, with further fine-tuning, DiffPMAE could achieve further improvements for completion, which we keep in future work.

3 Further Evaluation: Inference speed

Our model can produce around 4 items/sec on a one RTX3080 with the default setting. We tested the inference speed with different t settings in Table 4. To allocate the computational resource effectively, we also test the parallel sampling with default setting ($t = 200$) by setting the batch size to 32. With that setting, our model can produce 31 items/sec.

	$t = 100$, batch size $= 1$	$t = 200$, batch size $= 1$	$t = 300$, batch size $= 1$	$t = 200$, batch size $= 32$
items/sec	8.22	4.08	1.89	31.03

Table 4: Inference speed for different t settings.

4 Further Ablation: Diffusion Timestep

To measure the impact of timestemp t on diffusion process, we set timestep t=50, 100, 200, 300. We kept other hyperparameters as default as in Section 4.1

in our main paper. According to Table 6, *DiffPMAE* performs best at $t = 200$. However, further increase in t reduces the quality of the process in all metrics considered. The main reason is that with the increase of t, the required number of epochs should also be increased which is not scalable. Hence, we select $t = 200$ in *DiffPMAE* which significantly reduces time consumption compared to other point cloud generation methods [2, 3, 6] that requires t around 1000.

t	MMD CD ($\times 10^{-3}$)	1-NN CD ($\times 10^{-3}$)	JSD ($\times 10^{-3}$)	HD ($\times 10^{-2}$)
$t = 50$	1.474	11.718	4.527	3.891
$t = 100$	1.449	12.939	117.491	3.713
$t = 200$	**1.125**	**1.464**	**2.890**	**3.445**
$t = 300$	1.506	8.544	71.055	3.842

Table 5: Impact of diffusion timestep.

5 Further Ablation: Validation of Diffusion Model Parameters

We perform further ablation study with 300 diffusion steps with different parameters alongside the diffusion timestep experiments in our main paper. We show results with different setups in Table 6. We first extend the training process by increasing 300 epochs to 600 epochs. Then, we test a more complex decoder configuration with 8 depths and 6 headers. We also test the different ΔT values that control the diffusion process. However, none of those configurations with $t = 300$ can beat the default configuration of our model with $t = 200$. Those experiments further prove that our work can achieve better results with a more simple structure with $t = 200$, and those experiments also prove our method is efficient. Compared with prior diffusion-based works like PVD [7], our work only requires 200 steps to generate high-fidelity results instead of 1000 steps.

Model configurations	MMD CD ($\times 10^{-3}$)	1-NN CD ($\times 10^{-3}$)	JSD ($\times 10^{-3}$)	HD ($\times 10^{-2}$)
$t = 200$	**1.125**	**1.464**	**2.890**	**3.445**
$t = 300$	1.506	8.544	71.055	3.842
$t = 300(e = 600)$	2.128	46.386	138.907	4.891
$t = 300(d = 8, h = 6, e = 600)$	1.792	26.367	64.276	4.427
$t = 300(d = 8, h = 6, e = 600, \Delta T = 0.02)$	1.653	15.869	22.556	4.251

Table 6: Impact of different model configurations for *DiffPMAE*. t is number of diffusion steps, e is number of training epochs, d is number of depth in decoder, h is number of header in decoder.

6 Qualitative Results: Visualizations

In this section, we present further evidence for the quality of results for reconstruction and upsampling tasks that were presented in the main paper. For

reconstruction tasks, we use the trained *DiffPMAE* with the default configuration and parameters in our main paper. For upsampling tasks, we use the trained *DiffPMAE* for upsampling correspondingly (See Section 4.3 Upsampling part in our main paper). Fig. 3 shows the reconstruction results of *DiffPMAE*, with both input ground truth and output predicted results of 2048 points. Fig. 4 shows the upsampling results of *DiffPMAE*; the input size is 2048, and the output size is 8192.

7 Diffusion code

In this section, we provide a simple diffusion code of *DiffPMAE*. The full code repo can be find in our main paper.

```
class Diff_Point_MAE(nn.Module):
    def sampling_t(self, noisy_t, t, mask, center, x_vis):
        """
        Reverse sampling at timestep t.
        Input noisy level at timestep t,
        return noisy level at timestep t-1.
        """
        B, _, C = x_vis.shape  # B VIS C
        ts = self.time_emb(t.to(x_vis.device)).unsqueeze(1).expand(-1, self.num_group, -1)
        betas_t = self.get_index_from_list(self.betas, t, noisy_t.shape).to(x_vis.device)

        pos_emd_vis = self.decoder_pos_embed(center[~mask]).reshape(B, -1, C)
        pos_emd_msk = self.decoder_pos_embed(center[mask]).reshape(B, -1, C)
        pos_full = torch.cat([pos_emd_vis, pos_emd_msk], dim=1)
        _, N, _ = pos_emd_msk.shape
        mask_token = self.mask_token(noisy_t.reshape(B, N, -1).transpose(1, 2)).transpose(1,
      2).to(x_vis.device)
        x_full = torch.cat([x_vis, mask_token], dim=1)
        x_rec = self.MAE_decoder(x_full, pos_full, N, ts)
        x_rec = self.increase_dim(x_rec.transpose(1, 2)).transpose(1, 2).reshape(B, -1, 3)

        alpha_bar_t = self.get_index_from_list(self.alpha_bar, t, noisy_t.shape).to(x_vis.device)
        alpha_bar_t_minus_one = self.get_index_from_list(self.alpha_bar_t_minus_one, t,
      noisy_t.shape).to(x_vis.device)
        sqrt_alpha_t = self.get_index_from_list(self.sqrt_alphas, t, noisy_t.shape).to(x_vis.device)
        sqrt_alphas_bar_t_minus_one = self.get_index_from_list(self.sqrt_alpha_bar_minus_one, t,
      noisy_t.shape).to(
            x_vis.device)

        model_mean = (sqrt_alpha_t * (1 - alpha_bar_t_minus_one)) / (1 - alpha_bar_t) * noisy_t + (
                    sqrt_alphas_bar_t_minus_one * betas_t) / (1 - alpha_bar_t) * x_rec

        sigma_t = self.get_index_from_list(self.sigma, t, noisy_t.shape).to(x_vis.device)

        if t == 0:
            return model_mean
        else:
            return model_mean + torch.sqrt(sigma_t) * x_rec

    def sampling(self, x_vis, mask, center, trace=False, noise_patch=None):
        """
        Sampling the masked patches from Gaussian noise.
        """
        B, M, C = x_vis.shape
        if noise_patch is None:
            noise_patch = torch.randn((B, (self.num_group - M) * self.group_size, 3)).to(x_vis.device)
        diffusion_sequence = []

        for i in range(0, self.timestep)[::-1]:
            t = torch.full((1,), i, device=x_vis.device)
            noise_patch = self.sampling_t(noise_patch, t, mask, center, x_vis)
            if trace:
                diffusion_sequence.append(noise_patch.reshape(B, -1, 3))

        if trace:
            return diffusion_sequence
        else:
            return noise_patch.reshape(B, -1, 3)
```

References

1. Cui, R., Qiu, S., Anwar, S., Liu, J., Xing, C., Zhang, J., Barnes, N.: P2c: Self-supervised point cloud completion from single partial clouds (2023)
2. Kalischek, N., Peters, T., Wegner, J.D., Schindler, K.: Tetrahedral diffusion models for 3d shape generation (2022)
3. Nichol, A., Jun, H., Dhariwal, P., Mishkin, P., Chen, M.: Point-e: A system for generating 3d point clouds from complex prompts (Dec 2022)
4. Uy, M.A., Pham, Q.H., Hua, B.S., Nguyen, D.T., Yeung, S.K.: Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In: International Conference on Computer Vision (ICCV) (2019)
5. Yuan, W., Khot, T., Held, D., Mertz, C., Hebert, M.: Pcn: Point completion network. In: 2018 International Conference on 3D Vision (3DV). pp. 728–737 (2018)
6. Zeng, X., Vahdat, A., Williams, F., Gojcic, Z., Litany, O., Fidler, S., Kreis, K.: Lion: Latent point diffusion models for 3d shape generation. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 10021–10039. Curran Associates, Inc. (2022), https://proceedings.neurips.cc/paper_files/paper/2022/file/40e56dabe12095a5fc44a6e4c3835948-Paper-Conference.pdf
7. Zhou, L., Du, Y., Wu, J.: 3d shape generation and completion through point-voxel diffusion. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 5826–5835 (2021)

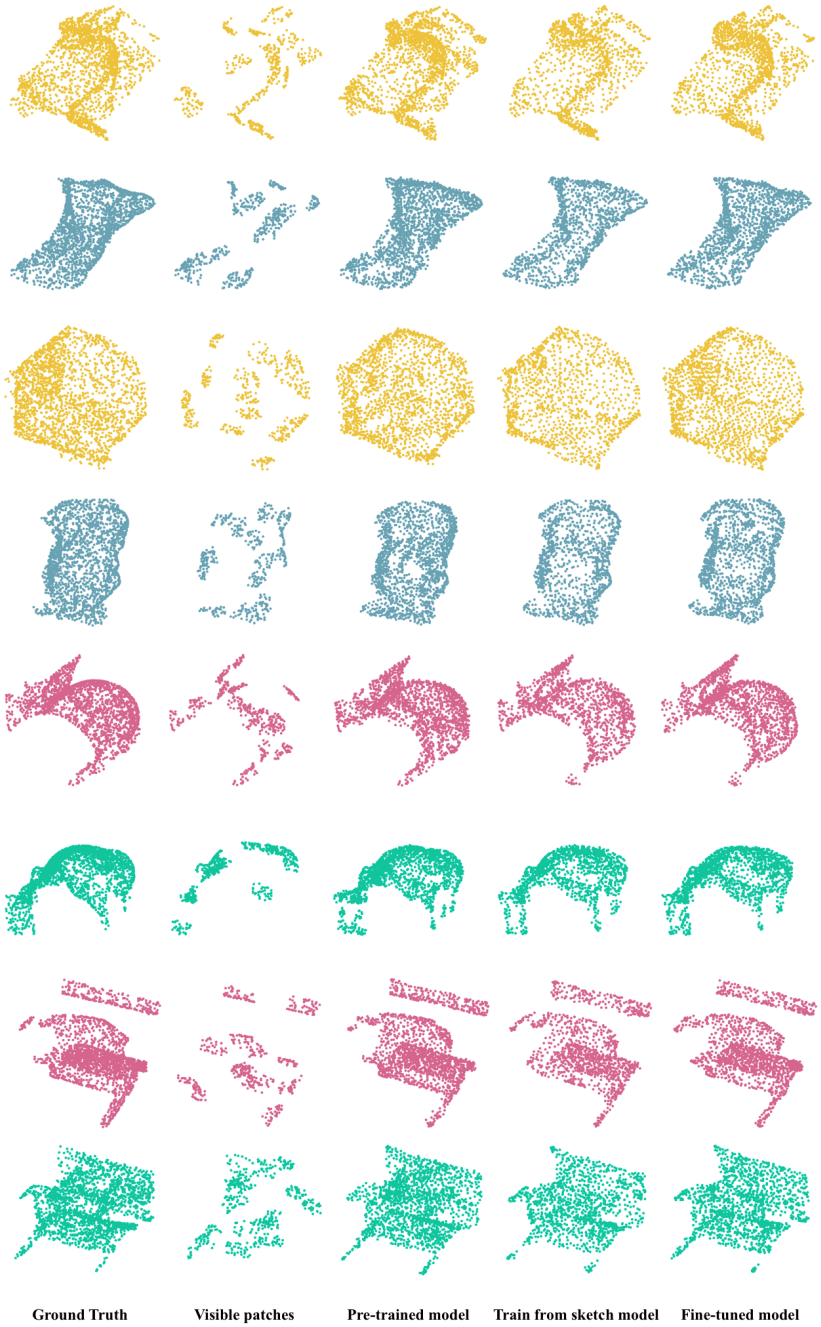

Fig. 1: Reconstruction results of *DiffPMAE* on ScanObjectNN dataset, main split with background. The predicted results are generated by *DiffPMAE* with mask ratio 0.75.

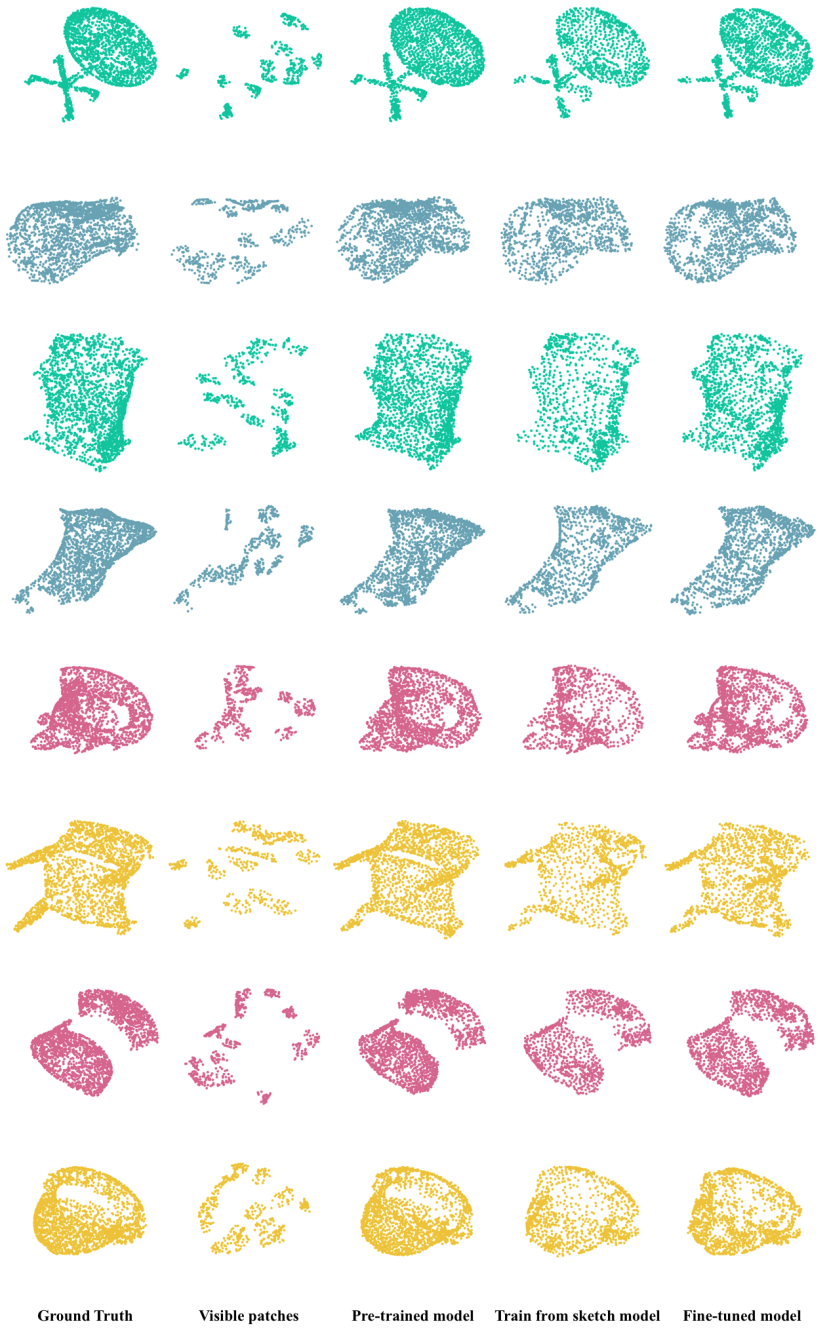

Fig. 2: Reconstruction results of *DiffPMAE* on ScanObjectNN dataset, main split without background. The predicted results are generated by *DiffPMAE* with mask ratio 0.75.

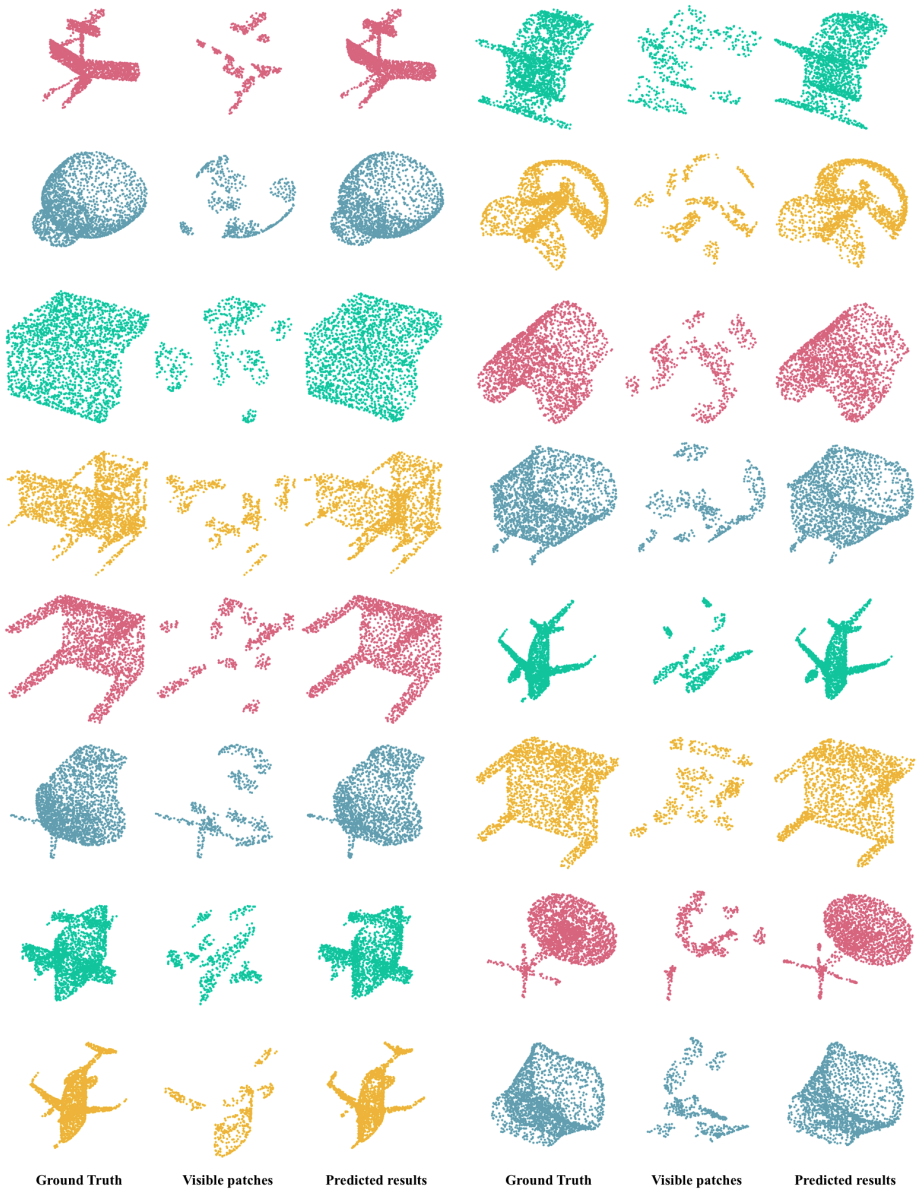

Fig. 3: Reconstruction results of *DiffPMAE* with multiple categories. Input point cloud and predicted results are 2048 points. The predicted results are generated by *DiffPMAE* with mask ratio 0.75.

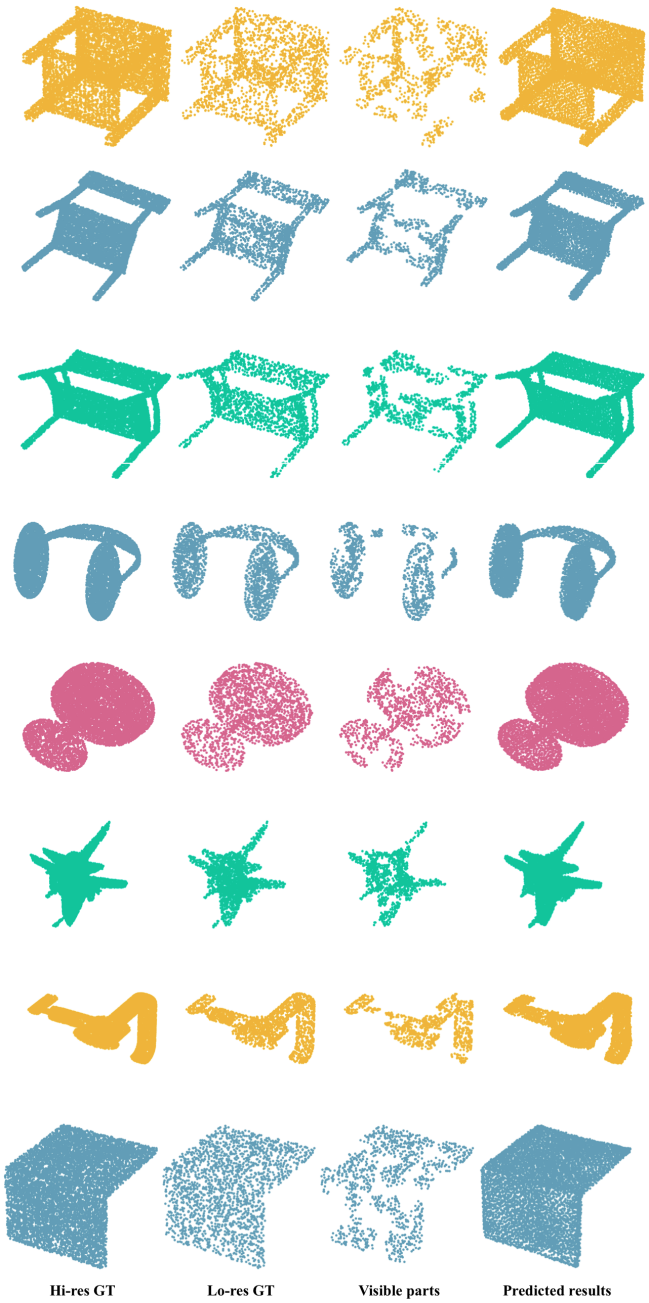

Fig. 4: Upsampling results of *DiffPMAE* with multiple categories. Input low resolution point clouds are contains 2048 points, high resolution and generated results are contains 8192 points. Our model is trained based on pairs of Hi-res GT and Lo-res GT and use Visible parts to generate the Hi-res predicted results. The predicted results are generated by *DiffPMAE* with mask ratio 0.4.