A. Implementation details

Completion pre-training Previous point completion models [10, 60, 48, 51] all use an "encoder-decoder" architecture. The encoder maps a partial point cloud to a vector of a fixed dimension, and the decoder reconstructs the full shape.

In the OcCo experiments, we exclude the last few MLPs of PointNet and DGCNN, and use the remaining architecture as the encoder to map a partial point cloud into a 1024-dimensional vector. We adapt the folding-based decoder design from PCN, which is a two-stage point cloud generator that generates a coarse and a fine-grained output point cloud (Y_{coarse}, Y_{fine}) for each input feature. We sketch the network structures of PCN encoder and output layers for downstream tasks in Figure 7. We removed all the batch normalisation in the folding-based decoder since we find they bring negative effects in the completion process in terms of loss and convergence rate, this has been reported in image generations [34]. Also, we find L2 normalisation in the Adam optimiser is undesirable for completion training but brings improvements for the downstream fine-tuning tasks.

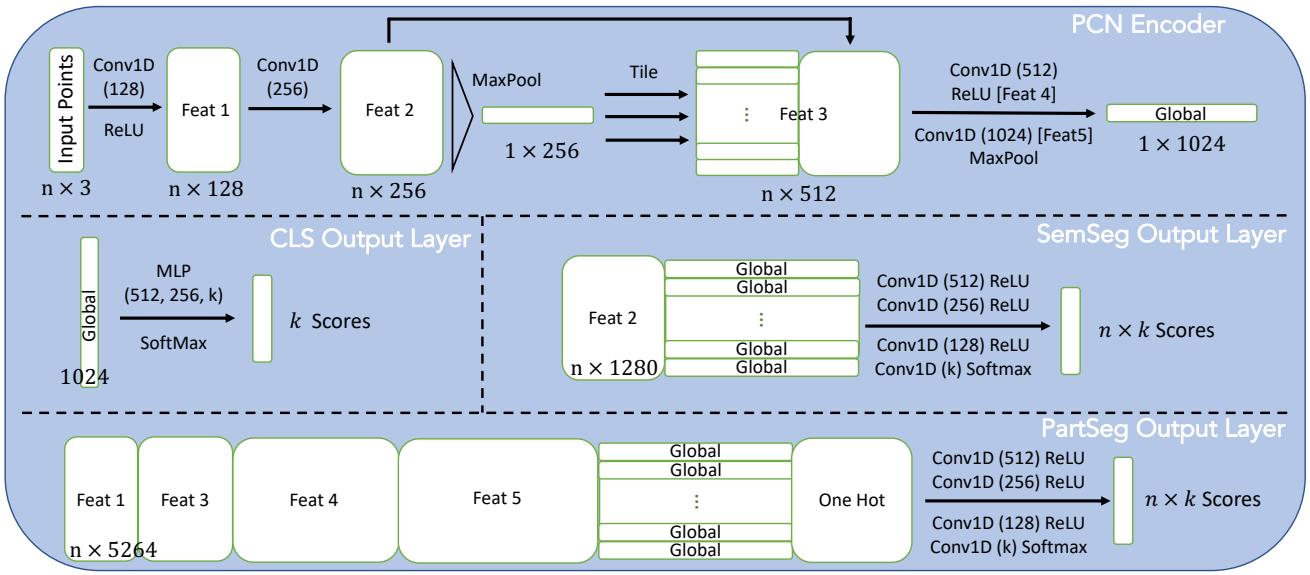

Figure 7: Encoder and Output Layers of PCN

We compare the occluded datasets based on ModelNet40 and ShapeNet8 for the OcCo pre-training. We construct the ModelNet Occluded using the methods described in Section 3 and for ShapeNet Occluded we directly use the data provided in the PCN, whose generation method are similar but not exactly the same with ours. Basic statistics of these two datasets are reported in Table 8.

Table 8: Statistics of occluded datasets for OcCo pre-training

Name	# of Class	# of Object	# of Views	# of Points/Object
ShapeNet Occluded (PCN and follow-ups)	8	30974	8	1045
ModelNet Occluded (OcCo)	40	12304	10	20085

By visualising the objects from the both datasets in Figure 8 and Figure 9, we show that our generated occluded shapes are more naturalistic and closer to real collected data. We believe this realism will be beneficial for the pre-training. We then test our hypothesis by pre-training models on one of the dataset, and fine tune them on the other. We report these results in Table 9. Clearly we see that the OcCo models pre-trained on ShapeNet Occluded do not perform as well as the ones pre-trained on ModelNet Occluded in most cases. Therefore we choose our generated ModelNet Occluded rather than ShapeNet Occlude [60, 48, 51] used in for the pre-training.

Figure 8: Examples from ShapeNet Occluded which fail to depict the underlying object shapes

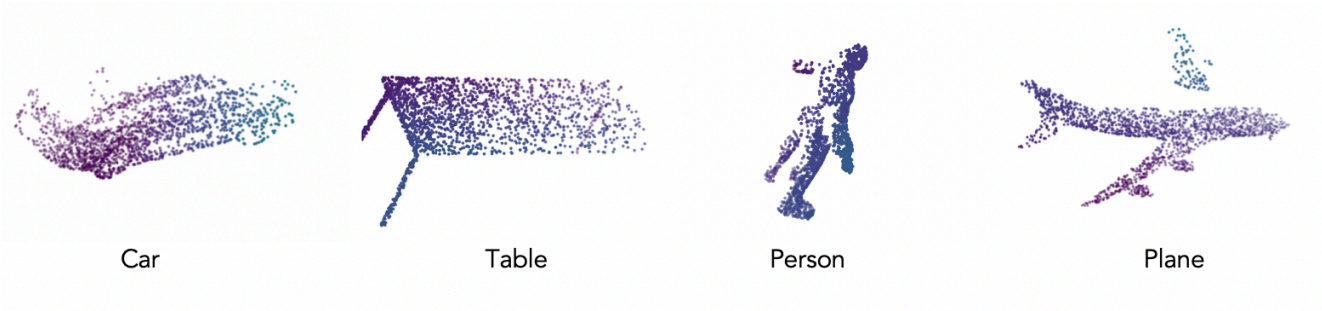

Figure 9: Examples of our generated self-occluded objects from ModelNet.

Table 9: Performance of OcCo pre-trained models with different pre-trained datasets

OcCo Settings		Classification Accuracy	
Encoder	Pre-Trained Dataset	ModelNet Occ.	ShapeNet Occ.
PointNet	ShapeNet Occ.	81.0	94.1
	ModelNet Occ.	**85.6**	**95.0**
PCN	ShapeNet Occ.	81.6	94.4
	ModelNet Occ.	**85.1**	**95.1**
DGCNN	ShapeNet Occ.	86.7	94.5
	ModelNet Occ.	**89.1**	**95.1**

Re-Implementation details of "Jigsaw" pre-training methods We describe how we reproduce the 'Jigsaw' pre-training methods from [42]. Following their description, we first separate the objects/chopped indoor scenes into $3^3 = 27$ small cubes and assign each point a label indicting which small cube it belongs to. We then shuffle all the small cubes, and train a model to make a prediction for each point. We reformulate this task as a 27-class semantic segmentation, for the details on the data generation and model training, please refer to our released code.

B. Ablations

As pointed out by the reviewers, we agree adding more runs will help. To help judge significance, we have ran 10 runs for three settings[2] and computed p-values via t-tests (unpaired, unequal variances) between OcCo and baselines (i.e., Jigsaw or random). We observe that all p-values are below the conventional significance threshold $\alpha = 0.05$ (the family-wise error rate is also, using Holm-Bonferroni).

As suggested by the reviewers, we ran ablations varying the number of object views and categories in Tables 11 and 12. We use Setting (1) from Table 10 as it is the fastest to run ($*$ indicates few-shot result in main paper).

[2]We chose settings that have low FLOPs across tasks and encoders.

Table 10: P-values for unpaired (unequal variance) t-tests between OcCo and baselines (across 10 runs). Setting: (1) Few-Shot (10-way 10-shot), ScanObjectNN, DGCNN; (2) Classification, ScanNet, PCN; (3) Segmentation, SensatUrban, PointNet.

Setting	OcCo vs. Rand	OcCo vs. Jigsaw
(1)	10^{-7}	10^{-7}
(2)	0.02	0.05
(3)	0.006	0.02

Table 11: Ablation: number of views (5 runs), $*$=main paper result.

# of Views	1	5	10*	20
PointNet	44.7±1.8	53.6±1.2	54.9±1.2	54.8±1.0
DGCNN	42.7±2.1	56.9±1.4	56.8±1.5	57.0±1.6

Table 12: Ablation: number of object categories (5 runs)

# of Categories	1	10	40*
PointNet	41.1±1.2	52.2±1.5	54.9±1.2
DGCNN	37.9±3.8	44.8±2.9	56.8±1.5

C. More results

3D object classification with Linear SVMs We follow the similar procedures from [1, 14, 42, 54, 59], to train a linear Support Vector Machine (SVM) to examine the generalisation of OcCo encoders that are pre-trained on the occluded objects from ModelNet40. For all six classification datasets, we fit a linear SVM on the output 1024-dimensional embeddings of the train split and evaluate it on the test split. Since [42] have already proven their methods are better than the prior, here we only systematically compare with theirs. We report the results[3] in Table 13, we can see that all OcCo models achieve superior results compared to the randomly-initialised counterparts, demonstrating that OcCo pre-training helps the generalisation both in-domain and cross-domain.

Table 13: linear SVM on the output embeddings from random, Jigsaw and OcCo initialised encoders

Dataset	PointNet			PCN			DGCNN		
	Rand	Jigsaw	OcCo	Rand	Jigsaw	OcCo	Rand	Jigsaw	OcCo
ShapeNet10	91.3	91.1	93.9	88.5	91.8	**94.6**	90.6	91.5	94.5
ModelNet40	70.6	87.5	88.7	60.9	73.1	88.0	66.0	84.9	**89.2**
ShapeNet Oc	79.1	86.1	91.1	72.0	87.9	90.5	78.3	87.8	**91.6**
ModelNet Oc	65.2	70.3	80.2	55.3	65.6	**83.3**	60.3	72.8	82.2
ScanNet10	64.8	64.1	67.7	62.3	66.3	**75.5**	61.2	69.4	71.2
ScanObjectNN	45.9	55.2	69.5	39.9	49.7	72.3	43.2	59.5	**78.3**

Few-shot learning We use the same setting and train/test split as cTree [44], and report the mean and standard deviation across on 10 runs. The top half of the table reports results for eight randomly initialised point cloud models, while the bottom-half reports results on two models across three pre-training methods. We bold the best results (and those whose standard deviation overlaps the mean of the best result). It is worth mentioning cTree [44] pre-trained the encoders on both datasets before fine tuning, while we only pre-trained once on ModelNet40. The results show that models pre-trained with OcCo either outperform or have standard deviations that overlap with the best method in 7 out of 8 settings.

[3] In our implementation, we also provide an alternative to use grid search to find the optimal set of parameters for SVM with a Radial Basis Function (RBF) kernel. In this setting, all the OcCo pre-trained models have outperformed the random and Jigsaw initialised ones by a large margin as well.

Table 14: More results on few-shot learning.

Baseline	ModelNet40				Sydney10			
	5-way		10-way		5-way		10-way	
	10-shot	20-shot	10-shot	20-shot	10-shot	20-shot	10-shot	20-shot
3D-GAN, Rand	55.8±10.7	65.8±9.9	40.3±6.5	48.4±5.6	54.2±4.6	58.8±5.8	36.0±6.2	45.3±7.9
FoldingNet, Rand	33.4 ±13.1	35.8±18.2	18.6±6.5	15.4±6.8	58.9±5.6	71.2±6.0	42.6±3.4	63.5±3.9
Latent-GAN, Rand	41.6±16.9	46.2±19.7	32.9±9.2	25.5±9.9	64.5±6.6	79.8±3.4	50.5±3.0	62.5±5.1
PointCapsNet, Rand	42.3±17.4	53.0±18.7	38.0±14.3	27.2±14.9	59.4±6.3	70.5±4.8	44.1±2.0	60.3±4.9
PointNet++, Rand	38.5±16.0	42.4±14.2	23.1±7.0	18.8±5.4	**79.9±6.8**	85.0±5.3	55.4±2.2	63.4±2.8
PointCNN, Rand	65.4±8.9	68.6±7.0	46.6±4.8	50.0±7.2	75.8±7.7	83.4±4.4	56.3±2.4	73.1±4.1
PointNet, Rand	52.0±12.2	57.8±15.5	46.6±13.5	35.2±15.3	74.2±7.3	82.2±5.1	51.4±1.3	58.3±2.6
PointNet, cTree	63.2±10.7	68.9±9.4	49.2±6.1	50.1±5.0	76.5±6.3	83.7±4.0	55.5±2.3	64.0±2.4
PointNet, OcCo	**89.7±6.1**	**92.4±4.9**	**83.9±5.6**	**89.7±4.6**	77.7±8.0	84.9±4.9	60.9±3.7	65.5±5.5
DGCNN, Rand	31.6 ±9.0	40.8±14.6	19.9±6.5	16.9±4.8	58.3±6.6	76.7±7.5	48.1±8.2	76.1±3.6
DGCNN, cTree	60.0±8.9	65.7±8.4	48.5±5.6	53.0±4.1	**86.2±4.4**	**90.9±2.5**	**66.2±2.8**	**81.5±2.3**
DGCNN, OcCo	**90.6±2.8**	**92.5±6.0**	**82.9±4.1**	**86.5±7.1**	**79.9±6.7**	**86.4±4.7**	63.3±2.7	**77.6±3.9**

Detailed results of the part segmentation Here in Table 15 we report the detailed scores on each individual shape category from ShapeNetPart, we bold the best scores for each class respectively. We show that for all three encoders, OcCo-initialisation has achieved better results over two thirds of these 15 object classes.

Table 15: Detailed Results on Part Segmentation Task on ShapeNetPart

Shapes	PointNet			PCN			DGCNN		
	Rand*	Jigsaw	OcCo	Rand	Jigsaw	OcCo	Rand*	Jigsaw*	OcCo
mean (point)	83.7	83.8	84.4	82.8	82.8	83.7	85.1	85.3	**85.5**
Aero	83.4	83.0	82.9	81.5	82.1	82.4	84.2	84.1	**84.4**
Bag	78.7	79.5	77.2	72.3	74.2	79.4	83.7	**84.0**	77.5
Cap	82.5	82.4	81.7	85.5	67.8	**86.3**	84.4	85.8	83.4
Car	74.9	76.2	75.6	71.8	71.3	73.9	77.1	77.0	**77.9**
Chair	89.6	90.0	90.0	88.6	88.6	90.0	90.9	90.9	**91.0**
Earphone	73.0	69.7	74.8	69.2	69.1	68.8	78.5	**80.0**	75.2
Guitar	91.5	91.1	90.7	90.0	89.9	90.7	91.5	91.5	**91.6**
Knife	85.9	86.3	88.0	84.0	83.8	85.9	87.3	87.0	**88.2**
Lamp	80.8	80.7	81.3	78.5	78.8	80.4	82.9	83.2	**83.5**
Laptop	95.3	95.3	95.4	95.3	95.1	95.6	96.0	95.8	**96.1**
Motor	65.2	63.7	65.7	64.1	64.7	64.2	67.8	**71.6**	65.5
Mug	93.0	92.3	91.6	90.3	90.8	92.6	93.3	94.0	**94.4**
Pistol	81.2	80.8	81.0	81.0	81.5	81.5	**82.6**	**82.6**	79.6
Rocket	57.9	56.9	58.2	51.8	51.4	53.8	59.7	**60.0**	58.0
Skateboard	72.8	75.9	74.2	72.5	71.0	73.2	75.5	**77.9**	76.2
Table	80.6	80.8	81.8	81.4	81.2	81.2	82.0	81.8	**82.8**

D. Algorithmic Description of OcCo

Algorithm 1 Occlusion Completion (OcCo)

```
# P: an initial point cloud
# K: camera intrinsic matrix
# V: number of total view points
# loss: a loss function between point clouds
# c: encoder-decoder completion model
# p: downstream prediction model

while i < V:
   # sample a random view-point
   R_t = [random.rotation(), random.translation()]

   # map point cloud to camera reference frame
   P_cam = dot(K, dot(R_t, P))

   # create occluded point cloud
   P_cam_oc = occlude(P_cam, alg='z-buffering')

   # point cloud back to world frame
   K_inv = [inv(K), zeros(3,1); zeros(1,3), 1]
   R_t_inv = transpose([R_t; zeros(3,1), 1])
   P_oc = dot(R_t_inv, dot(K_inv, P_cam_oc))

   # complete point cloud
   P_c = c.decoder(c.encoder(P_oc))

   # compute loss, update via gradient descent
   l = loss(P_c, P)
   l.backward()
   update(c.params)
   i += 1

# downstream tasks, use pre-trained encoders
p.initialise(c.encoder.params)
p.train()
```

E. Visualisation from Completion Pre-Training

In this section, we show some qualitative results of OcCo pre-training by visualising the input, coarse output, fine output and ground truth at different training epochs and encoders. In Figure. 10, Figure. 11 and Figure. 12, we notice that the trained completion models are able to complete even difficult occluded shapes such as plants and planes. In Figure. 13 we plot some failure examples of completed shapes, possibly due to their complicated fine structures, while it is worth mentioning that the completed model can still completed these objects under the same category.

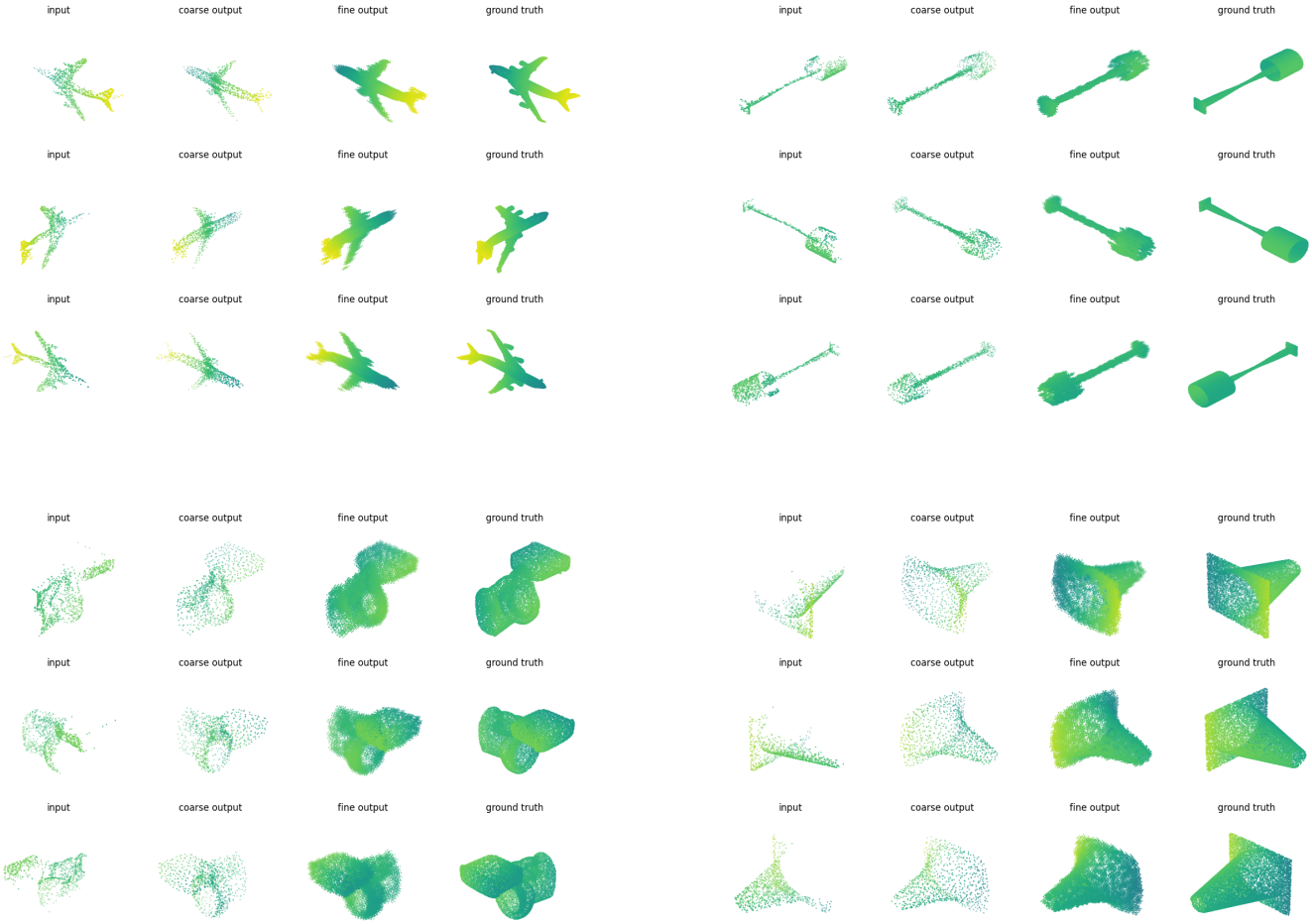

Figure 10: OcCo pre-training with PCN encoder on occluded ModelNet40.

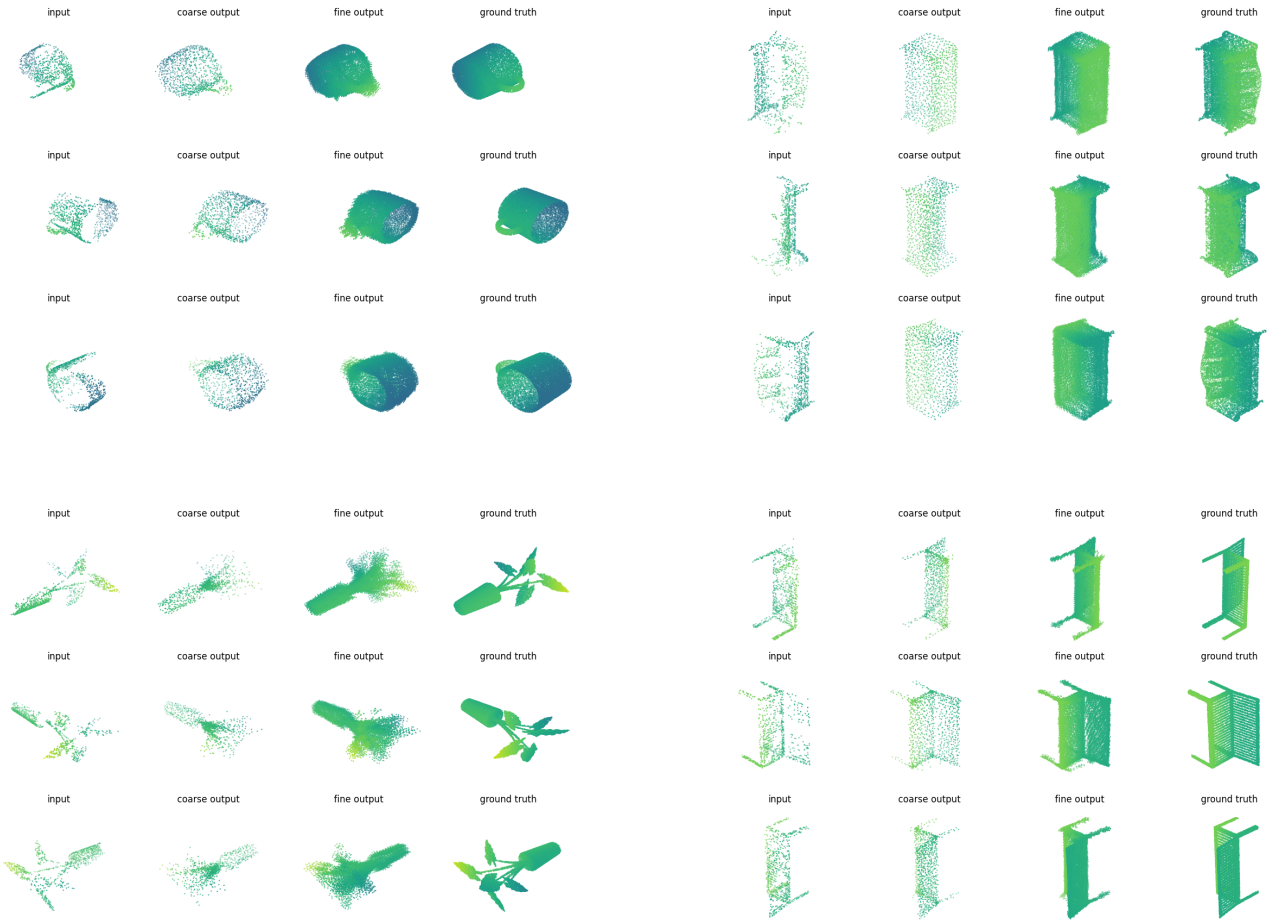

Figure 11: OcCo pre-training with PointNet encoder on occluded ModelNet40.

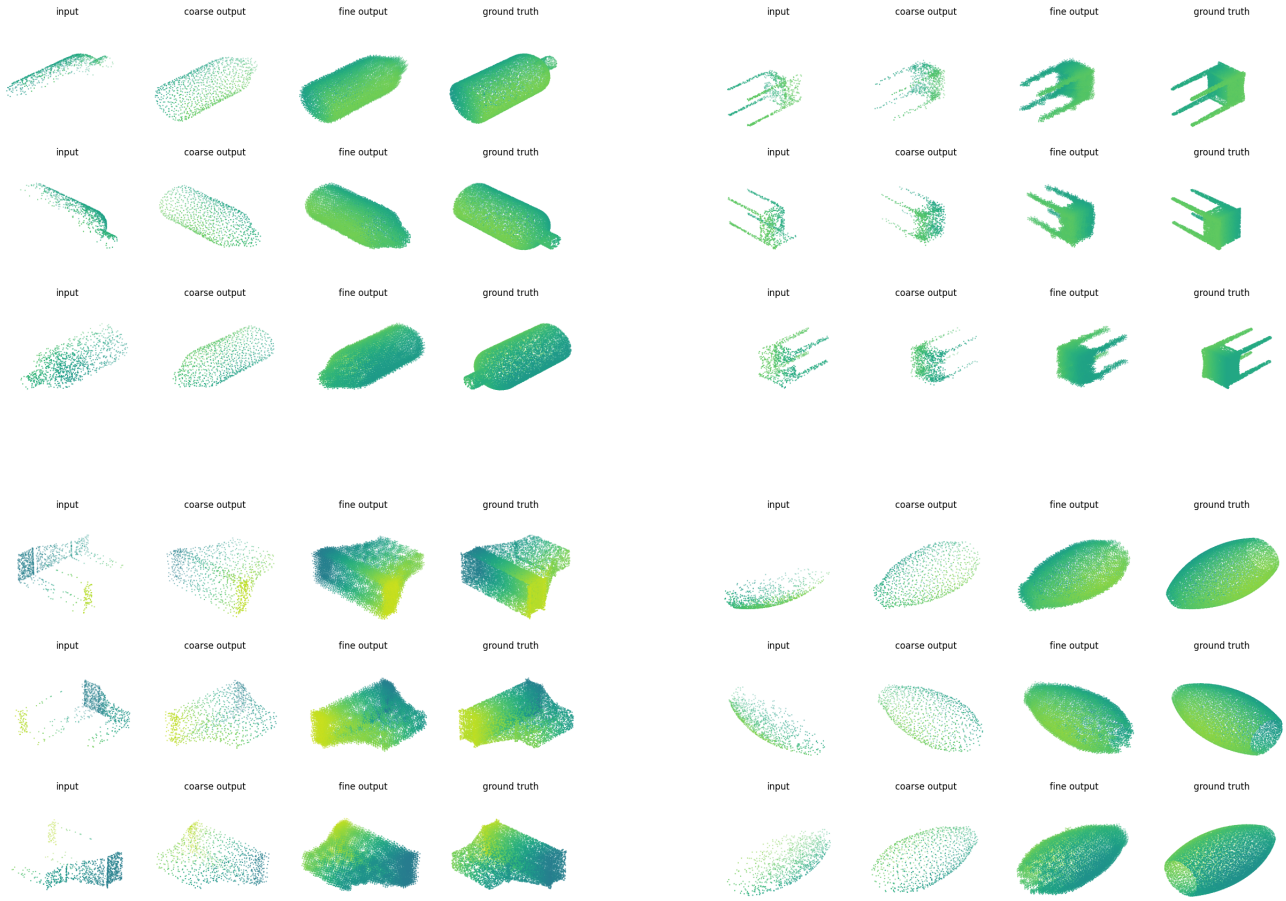

Figure 12: OcCo pre-training with DGCNN encoder on occluded ModelNet40.

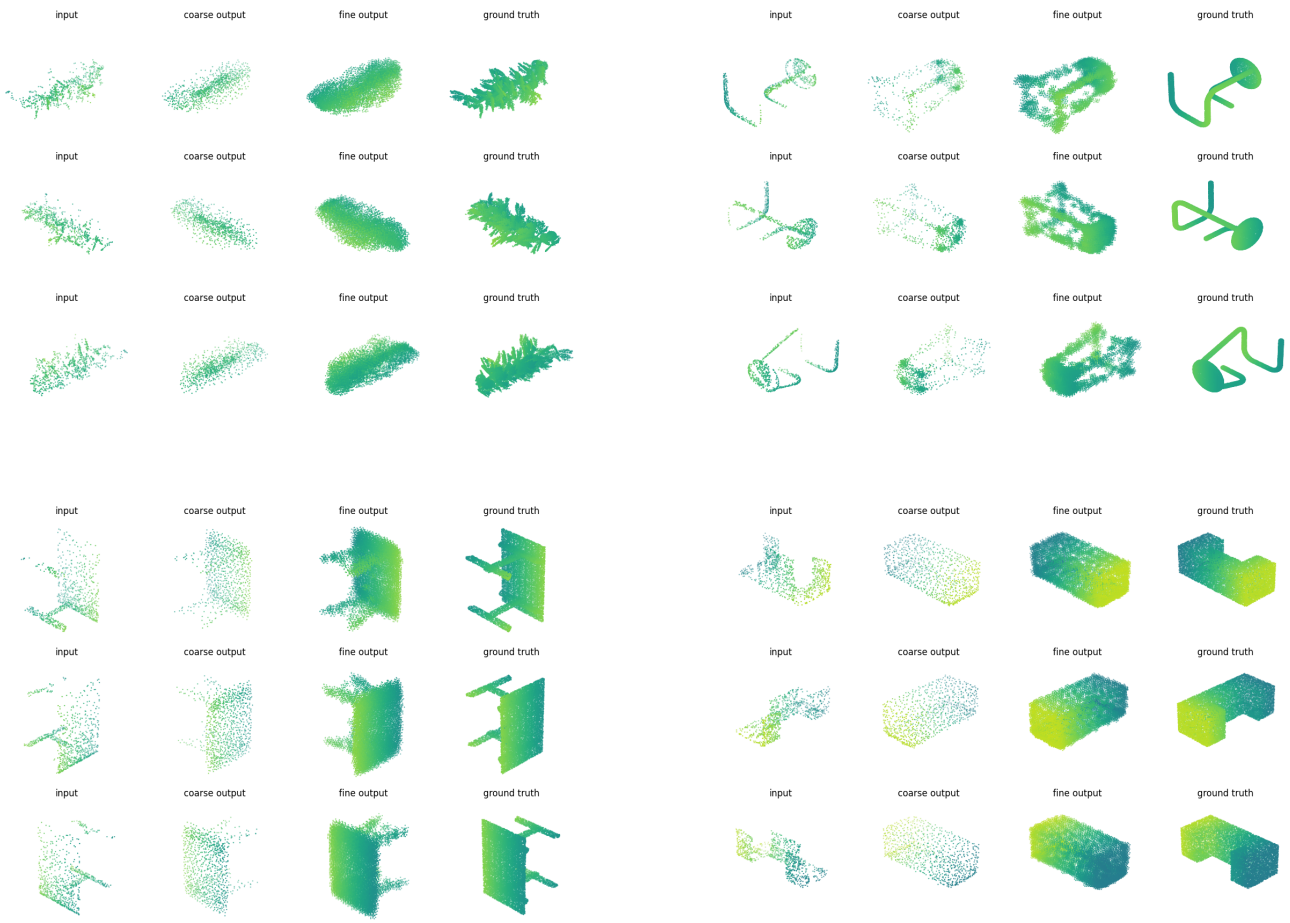

Figure 13: Failure completed examples during OcCo pre-training.