In this supplemental material, we provide additional qualitative results in Section A, evaluation details in Section B, additional per-part evaluation in Section C, additional ablation discussion with qualitative visualizations for ablations in Section D, visualization of part interpolations through learned latent part spaces in Section E, part specifications per category in Section F, and implementation details with the description of the network architecture in Section G.

A. Additional qualitative analysis

In Figures 5, 6 and 7, we show additional qualitative results and comparison of NPPs to baseline methods. We can see that StructureNet [35] and Bokhovkin et al. [3] often produce incomplete and inaccurate shapes, can include inconsistent parts (e.g. predicting only one chair arm or predicting two types of legs for one chair). The advanced point cloud segmentation method PointGroup [30] is able to predict consistent part types for shapes but produces fairly noisy geometry for these parts. In addition, when comparing to baselines and NPPs without applying scene-aware constraints, we can clearly see a large amount of diversity within shapes that should be similar or identical within one scan.

Several categories can be more challenging, due to more often appearing with clutter (e.g., tables often have objects on top vs chairs or trash cans); our learned manifolds help to regularize this during TTO. In Fig. 8, we show common failure cases that NPPs produces for different shape categories. Parts with little geometric distinction (e.g., cabinet frame vs drawer often both lie on a flat plane) can be more difficult to optimize, due to more challenging segmentation. The most left case with the cabinet shows erroneous detection (too small), along with an excess wrong part prediction of a cabinet base on the bottom. Missing scanned legs of the chair result in the incorrect type of reconstructed chair legs, while the 4-spoke swivel chair is ground truth. Dense segmentation of real-world scans is often significantly challenging, tending to segment parts of the floor as chair legs, parts of the wall as a bed headboard, or treating a full trash bin as a bin with a top cover part.

B. Evaluation details

For semantic part completion and part segmentation evaluation, we sample $10,000$ points per part from predicted and ground-truth mesh surfaces (within the corresponding MLCVNet bounding box) and transform them to the ScanNet coordinate space. The Chamfer Distance metric is evaluated for every pair of semantically matching parts between predicted and ground-truth meshes. In case there exists a part in a ground-truth mesh that is missing for a predicted mesh or vice versa, we use the center of the mesh as a missing part. After each part is evaluated, we average scores to obtain the final score for a full object.

For segmentation evaluation, we project labels from sampled points onto ScanNet mesh vertices to obtain the set of points not depending on a method. Here, to compute segmentation Chamfer Distance for an object, the predicted and ground-truth projected labeled points are used to get per-part scores and then averaged. For one part IoU evaluation, the corresponding projected points are marked as ones and the rest as zeros, intersection and union scores are computed using these 0-1 sets. We similarly use $10,000$ sampled points per part for these metrics.

C. Per-part evaluation

We provide per-part semantic part completion and part segmentation in Tab. 4, 5, 6, and 7. Our learned part manifolds enable more robust, accurate geometry reconstruction also for individual parts.

	Chamfer Distance (↓) – Accuracy																		
	chair							table							cabinet				
Method	left arm	right arm	back	seat	reg. leg	star leg	surf. base	central supp.	drawer	leg	pedestal	shelf	surface	side panel	door	shelf	frame	base	countertop
SG-NN [12] + MLCVNet [55] + PointGroup [30]	0.081	0.075	0.045	0.012	0.022	0.099	0.200	0.164	0.204	**0.044**	0.245	0.174	0.024	0.204	0.150	0.124	**0.010**	0.376	0.344
MLCVNet [55] + StructureNet [35]	0.041	0.036	0.005	0.008	**0.020**	0.100	0.223	0.123	**0.021**	0.105	0.167	**0.041**	0.022	0.169	0.104	**0.033**	0.028	0.268	0.344
Bokhovkin et al. [3]	0.039	0.039	0.008	0.008	0.057	0.074	0.110	0.044	0.094	0.141	0.167	0.121	0.024	0.175	0.083	0.066	0.032	0.210	**0.229**
Ours	**0.015**	**0.015**	**0.002**	**0.003**	0.026	**0.058**	**0.081**	**0.042**	0.143	0.077	**0.101**	0.146	**0.006**	**0.164**	**0.042**	0.078	0.017	**0.207**	0.306

Table 4. Per-part evaluation of semantic part completion for 'chair', 'table', and 'cabinet' categories on Scan2CAD [1] in comparison to state-of-the-art part segmentation [30,35] and semantic part completion [3].

	Chamfer Distance (↓) – Accuracy														
	bookshelf				bed				bin						
Method	door	shelf	frame	base	frame	side surf.	sleep area	headboard	base	bottom	box	cover	frame	class avg.	inst. avg.
SG-NN [12] + MLCVNet [55] + PointGroup [30]	0.097	0.245	**0.005**	1.298	0.059	0.776	**0.009**	0.890	0.191	0.072	**0.001**	0.133	0.049	0.201	0.077
MLCVNet [55] + StructureNet [35]	0.070	0.186	0.045	1.161	0.081	0.776	0.047	1.375	0.191	0.044	0.003	0.126	0.049	0.188	0.055
Bokhovkin et al. [3]	0.041	0.137	0.090	**0.858**	0.073	0.485	0.127	0.508	0.131	0.038	0.004	0.042	0.041	0.134	0.054
Ours	**0.037**	**0.079**	0.068	1.298	**0.020**	**0.290**	0.051	**0.365**	**0.111**	**0.020**	0.002	**0.012**	**0.037**	**0.123**	**0.033**

Table 5. Per-part evaluation of semantic part completion for 'bookshelf', 'bed', and 'bin' categories on Scan2CAD [1] in comparison to state-of-the-art part segmentation [30,35] and semantic part completion [3].

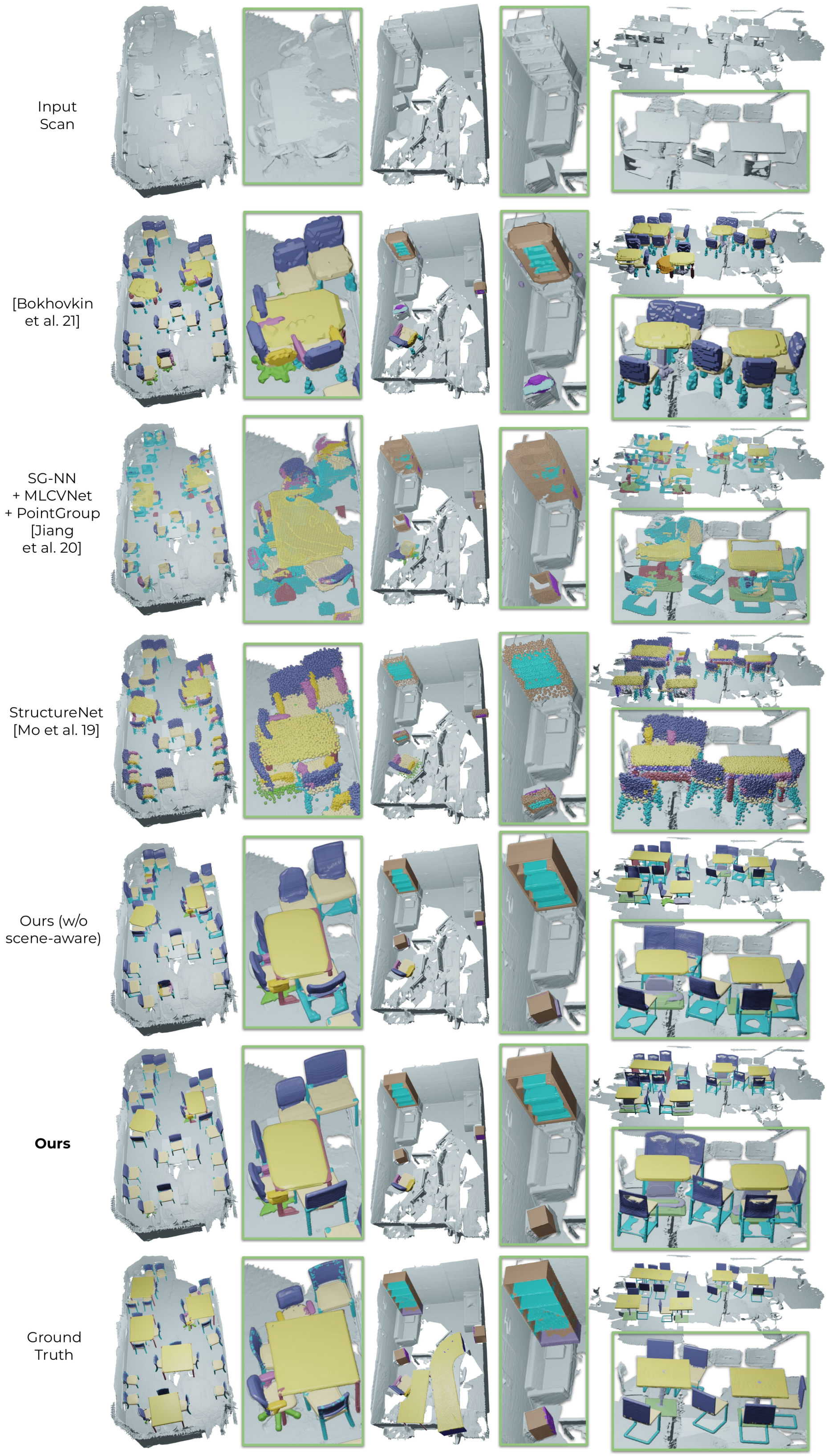

Figure 5. Additional qualitative comparison of NPPs with point [30, 35] and voxel-based [3] state of the art on ScanNet scans with Scan2CAD+PartNet ground truth.

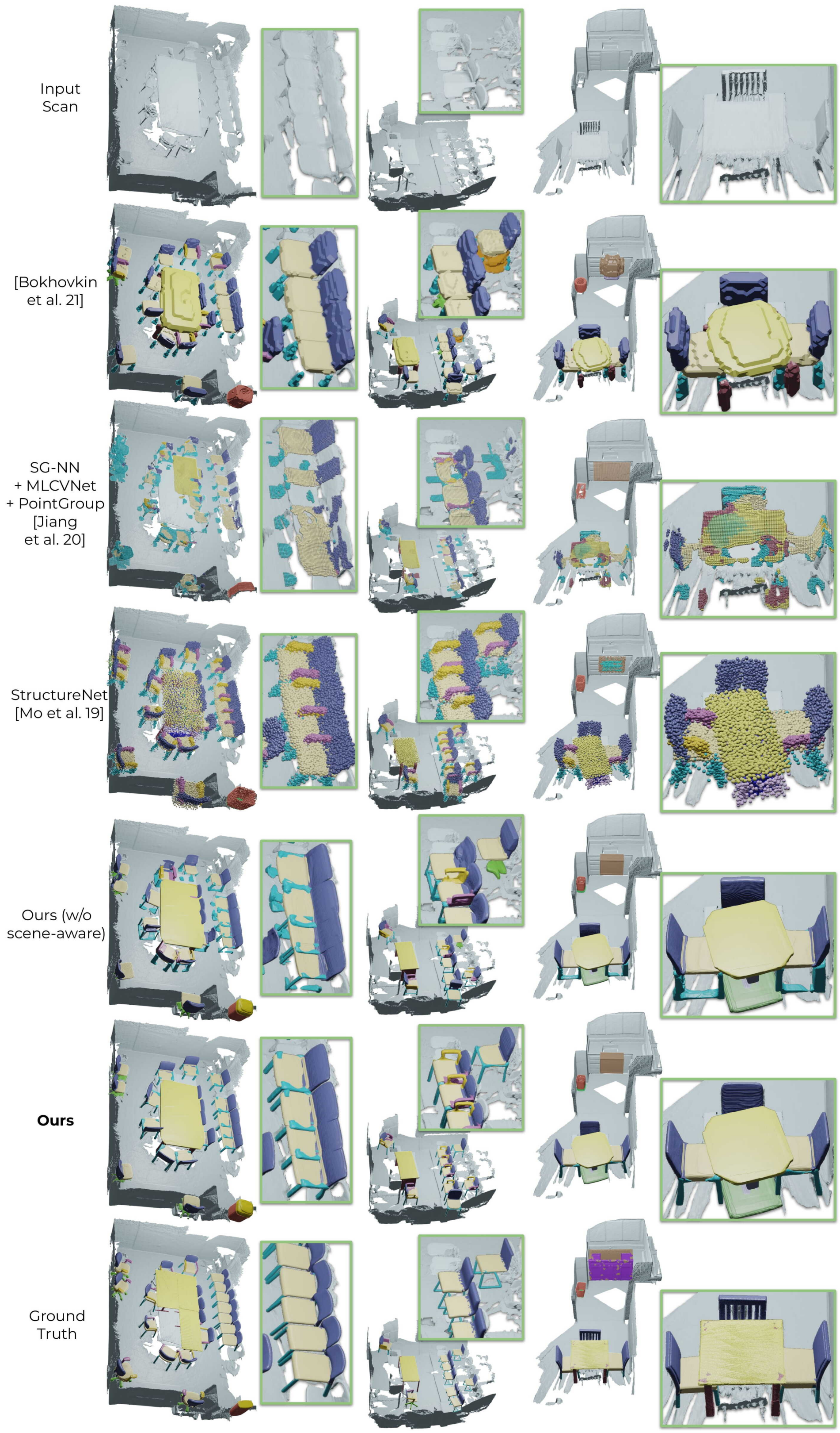

Figure 6. Additional qualitative comparison of NPPs with point [30, 35] and voxel-based [3] state of the art on ScanNet scans with Scan2CAD+PartNet ground truth.

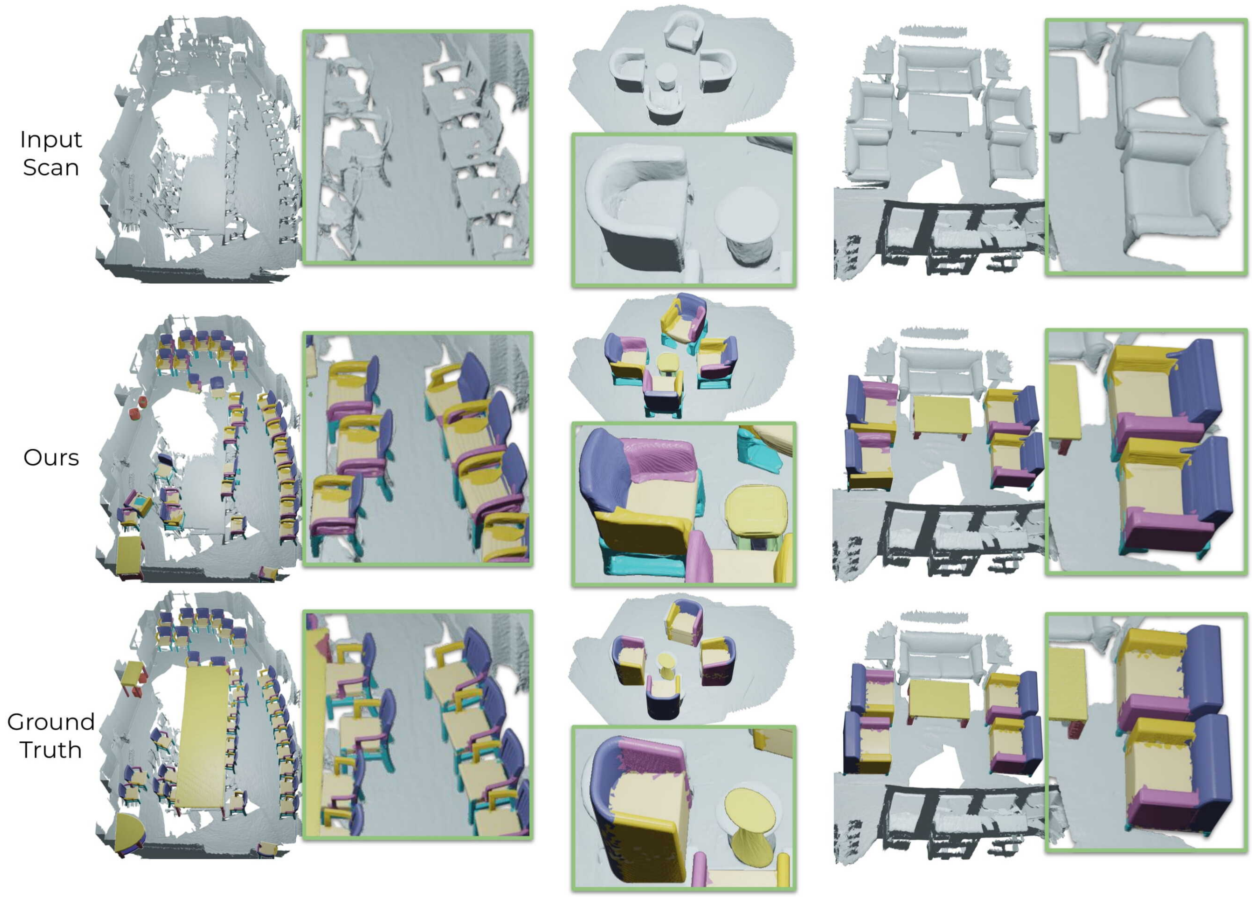

Figure 7. Additional qualitative results on ScanNet [11] with Scan2CAD [1] and PartNet [36] targets, showing our consistent, complete part decompositions.

	Chamfer Distance (↓) – Completion																		
	chair							table							cabinet				
Method	left arm	right arm	back	seat	reg. leg	star leg	surf. base	central supp.	drawer	leg	pedestal	shelf	surface	side panel	door	shelf	frame	base	countertop
SG-NN [12] + MLCVNet [55] + PointGroup [30]	0.096	0.086	0.041	0.014	0.054	0.063	0.161	0.152	0.137	**0.092**	0.305	0.151	0.050	0.203	0.081	0.140	**0.043**	0.384	0.156
MLCVNet [55] + StructureNet [35]	0.051	0.049	0.008	0.008	0.035	0.049	0.126	0.062	0.194	0.158	**0.148**	**0.102**	0.037	0.246	0.104	**0.085**	0.061	0.298	0.156
Bokhovkin et al. [3]	0.046	0.049	0.012	0.011	0.056	0.057	0.139	0.078	0.149	0.165	**0.148**	0.106	0.045	**0.181**	0.094	0.111	0.062	**0.219**	**0.111**
Ours	**0.043**	**0.040**	**0.006**	**0.005**	**0.032**	**0.031**	**0.120**	**0.044**	**0.135**	0.118	**0.148**	0.116	**0.026**	**0.181**	**0.071**	0.159	0.064	0.266	0.152

Table 6. Per-part evaluation of part segmentation for 'chair', 'table', and 'cabinet' categories on Scan2CAD [1] in comparison to state-of-the-art part segmentation [30, 35] and semantic part completion [3].

	Chamfer Distance (↓) – Completion														
	bookshelf				bed				bin						
Method	door	shelf	frame	base	frame	side surf.	sleep area	headboard	base	bottom	box	cover	frame	class avg.	inst. avg.
SG-NN [12] + MLCVNet [55] + PointGroup [30]	0.348	0.143	**0.020**	1.066	0.072	0.567	0.043	0.999	0.284	0.077	**0.004**	0.093	0.038	0.193	0.084
MLCVNet [55] + StructureNet [35]	**0.173**	0.131	0.071	0.942	0.077	0.567	0.057	1.122	0.284	0.052	0.007	0.096	0.038	0.175	0.062
Bokhovkin et al. [3]	0.361	**0.095**	0.096	**0.754**	0.068	0.505	0.109	0.507	**0.153**	0.050	0.008	0.045	0.036	0.144	0.056
Ours	0.520	0.130	0.119	1.066	**0.034**	**0.221**	**0.038**	**0.365**	0.167	**0.029**	**0.004**	**0.020**	**0.033**	**0.140**	**0.043**

Table 7. Per-part evaluation of part segmentation for 'bookshelf', 'bed', and 'bin' categories on Scan2CAD [1] in comparison to state-of-the-art part segmentation [30, 35] and semantic part completion [3].

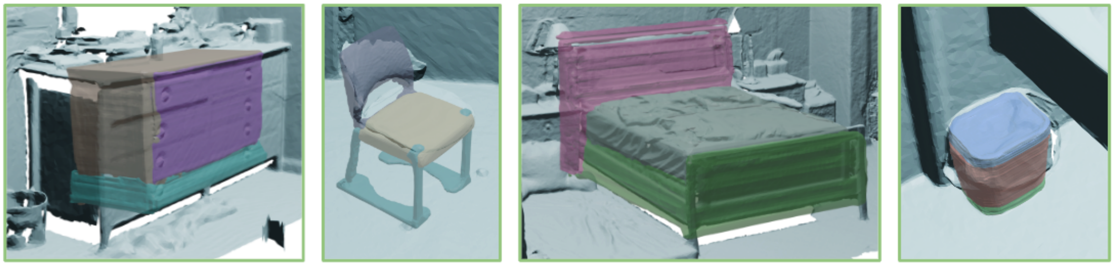

Figure 8. Common failure cases for different shape categories produced by NPPs.

D. Additional ablation discussion

In Tab. 8 we compare results without Scene Consistency constraints to ours only for the instances affected by these constraints. Evaluated only on $\sim 66\%$ of instances and $\sim 57\%$ of corresponding ScanNet scenes, NPPs outperform the results without Scene Consistency constraints by a greater gap compared to Tab. 3.

In Fig. 9 we show the qualitative ablation results corresponding to Tab. 3. Compared to quantitative results of scene-aware constraint ablation, we see a more noticeable effect in qualitative effect, with much more consistent part decompositions for similar objects in a scene, even when seen under fairly different partial views (i.e., matching left and right chair arms, consistent joint of the table surface and the table stand).

Without latent projection, arbitrary initializations for TTO often land outside the basin of convergence (i.e., starting with too discrepant parts for the trash bin and the cabinet), resulting in poorer performance, as low-level geometric constraints may be ambiguous for resolving large structural differences. TTO then improves significantly the fitting accuracy enabling the prediction of geometry that lies outside of the learned manifold (i.e., round table surface, missing pillows on the bed). By leveraging TTO and projection initialization, we can achieve the best representation of the input scan as its part decomposition.

We evaluate the effect of synthetic pre-training of the part segmentation in Tab. 3 (*w/o Synthetic Pretrain*). The additional quantity and diversity of data help to avoid overfitting to more limited real data (i.e., very poor results for ’bed’ and ’trashcan’ categories due to limited real-world data).

The full-shape constraint helps to maintain consistency between the global shape and the optimized parts during test-time optimization (i.e. not connected box and cover parts for the trash bin, inconsistent joints for the chair and the cabinet). Dense segmentation guides the TTO optimization constraints and prevents self-intersections between parts.

E. Interpolation properties of learned latent part spaces

In Figure 10, we show the interpolation capabilities of the part latent spaces that we use in NPPs to traverse during test-time optimization. Although each part space has been learned individually, their interpolations can produce consistent shapes.

F. Part types per category

In Figure 11 we present the shape categories and the corresponding parts that we use in our framework. There are 6 shape categories and 28 part types in total.

G. Implementation details

We provide further implementation details; note that parameters reported here are for the ‘chair’ category, and other category parameter differences are specified in Table 9.

G.1. Pretrain decoders

We first train our latent part and shape spaces on the synthetic PartNet [36] dataset. This corresponds to the tasks ‘Train decoder (shape)’, ‘Train decoder (parts)’ in Table 9. The part and shape decoders are all MLPs composed of 8 linear layers of 512 dimensions each, using ReLU non-linearities with a final tanh for SDF output. The detailed architecture is shown in Tables 14, 15. To train the shape decoder, we use an Adam optimizer with a batch size of 24 and learning rate of 5e-5 (‘lr’ in the Table 9) for network weights (with a factor 0.5 (‘lr factor’) and decay interval of 500 epochs (‘lr decay int.’)) and 1e-4 for latent parameters (with a factor 0.5 and decay interval of 500 epochs), and train for 2000 epochs. For the part decoder, we extend every part latent with one-hot encoded part type and train the part decoder using an Adam optimizer with a batch size of 48 and learning rate of 5e-5 for network weights (with a decay factor of 0.5 and decay interval of 400 epochs) and 1e-4 for latent parameters (with a decay factor 0.5 and decay interval of 400 epochs), and train for 2000 epochs.

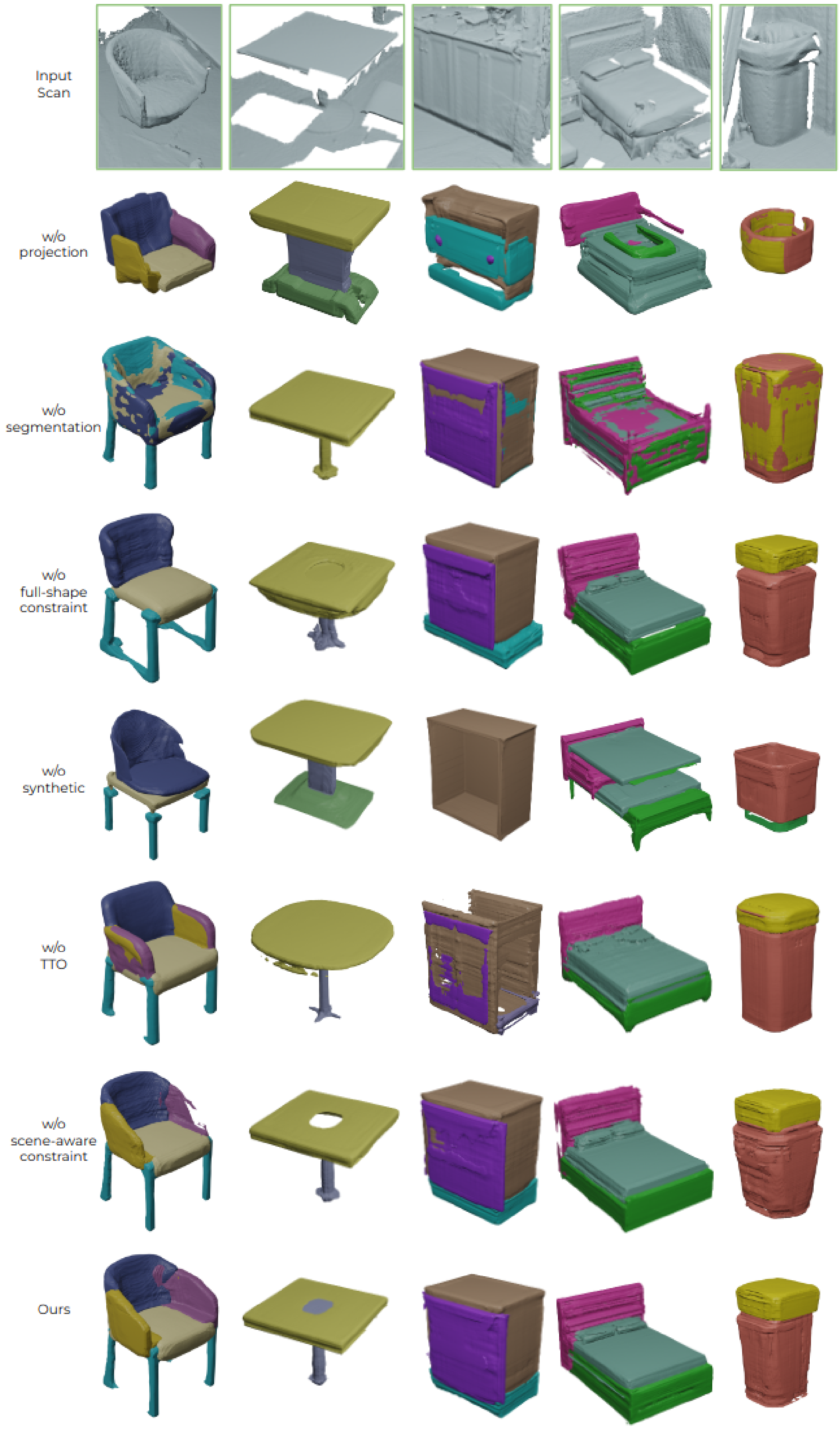

Figure 9. Qualitative ablation results on ScanNet [11] with Scan2CAD [1] and PartNet [36] targets, showing the importance of every design choice in our method.

	Chamfer Distance (↓) – Accuracy								Chamfer Distance (↓) – Completion							
Method	chair	table	cab.	bkshlf	bed	bin	class avg	inst avg	chair	table	cab.	bkshlf	bed	bin	class avg	inst avg
# scenes	126 / 189	30 / 127	14 / 94	13 / 39	14 / 47	10 / 74	164 / 285	164 / 285	126 / 189	30 / 127	14 / 94	13 / 39	14 / 47	10 / 74	164 / 285	164 / 285
# instances	764 / 904	81 / 190	32 / 140	24 / 54	28 / 61	18 / 85	947 / 1434	947 / 1434	764 / 904	81 / 190	32 / 140	24 / 54	28 / 61	18 / 85	947 / 1434	947 / 1434
w/o Scene Consistency	0.016	0.055	**0.054**	0.174	0.129	0.043	0.078	0.028	0.020	0.065	**0.112**	**0.225**	0.137	0.042	0.100	**0.033**
Ours	**0.011**	**0.047**	0.057	**0.163**	**0.101**	**0.036**	**0.069**	**0.023**	**0.019**	**0.063**	0.119	0.257	**0.099**	**0.035**	**0.098**	**0.033**

Table 8. Ablation study evaluating semantic part completion on Scan2CAD [1] including only the instances affected with Scene Consistency constraints. We also provide information of how many shape instances and scenes are affected with scene-aware constraints for each category.

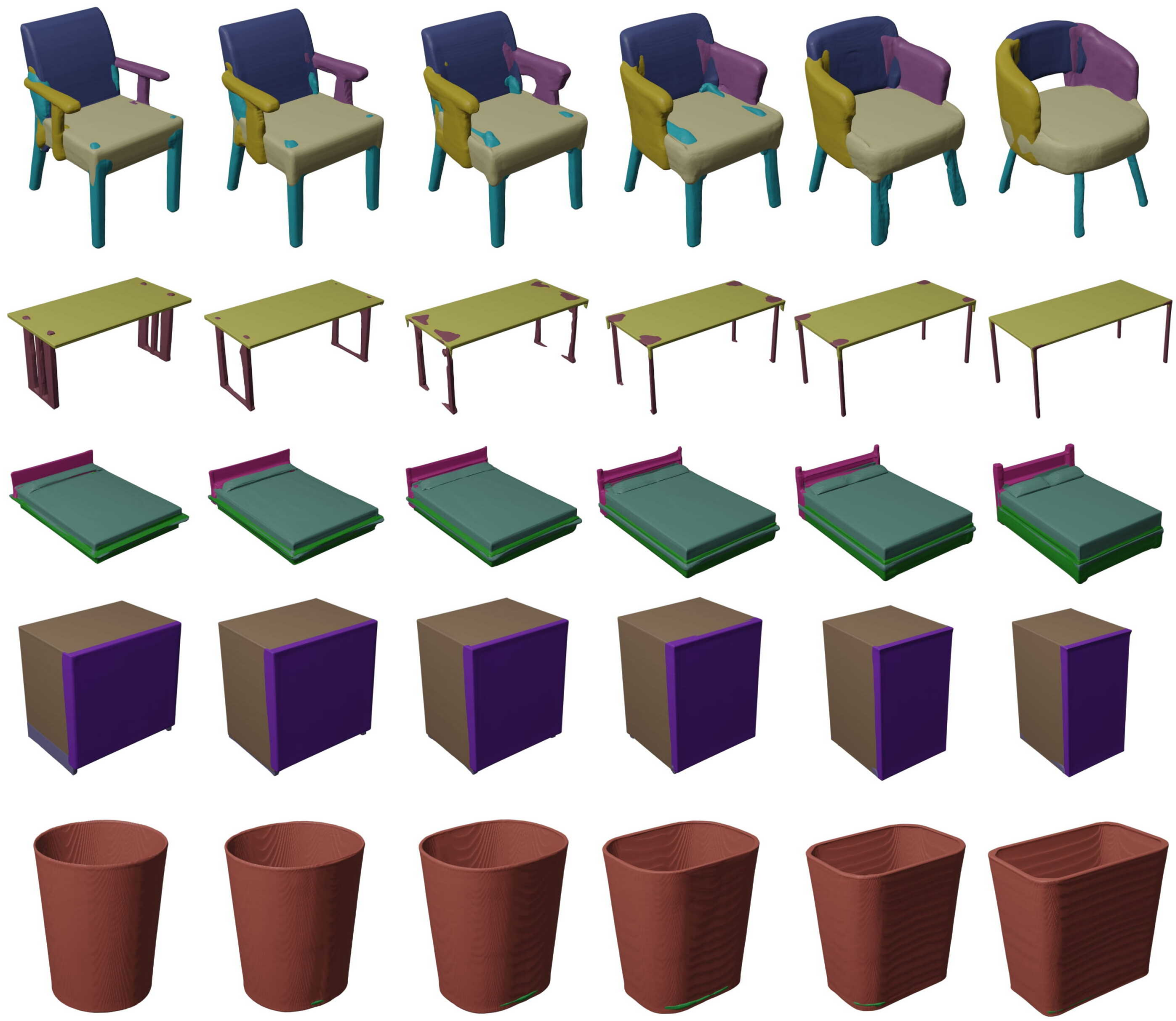

Figure 10. Part interpolations through our learned latent part spaces for different shape classes.

G.2. Pre-training for latent projection and part segmentation

We then train the projection mapping into the learned part and shape spaces as well as the part segmentation. This part corresponds to the task 'Train projection' in Table 9. Our model is pre-trained on synthetic PartNet data using virtually scanned incomplete inputs to take advantage of a large amount of synthetic data. We use an Adam optimizer with batch size 64 and learning rate 1e-3 ('lr' in the Table 9) decayed by half ('lr factor') every 12 epochs ('lr decay int.') for 35 epochs. We use a large and a small PointNet-based [44] network (small ('PN-small') and large ('PN-big')) to segment an input TSDF into parts and background. We refer to Tables 12, 13 as architectures of 'PN-small' and 'PN-big' denoted in Table 9.

Figure 11. Part specifications per category for the parts used in our approach. Note that 'cabinet' and 'bookshelf' have the same set of parts.

G.3. Fine-tuning on ScanNet data

To apply to real-world observations, we fine-tune the projections and part segmentation on ScanNet [11] data using MLCVNet [55] detections on train scenes. We use an Adam optimizer with batch size 64, learning rate 2e-4 ('lr' in the Table 9) decayed by a factor of 0.2 ('lr factor') every 40 epochs ('lr decay int.') for 80 epochs.

G.4. Test-time optimization

For test-time optimization, we optimize for part and shape codes using an Adam optimizer with learning rate of 3e-4 ('lr' in Table 9) for 500 iterations. The learning rate is multiplied by a factor of 0.1 ('lr factor') after 300 iterations ('lr decay int.')). This part corresponds to the task 'Test-time opt.' in Table 9.

To enable more flexibility to capture input details, we enable optimization of the decoder weights for parts and shape after 400 iterations. We have used the first and the second linear layers of part decoder ('part dec. layers opt.') to optimize simultaneously with latent vectors optimization using Adam optimizer with learning rate of 3e-4 ('lr') for 100 iterations. The learning rate is multiplied by a factor of 0.1 ('lr factor (part dec.)') after 300 iterations ('lr decay int. (part dec.)')).

In Eqs. (5), (6) we use a weight w_{trunc} for points close to and further away from the surface. We have a set $\mathcal{A}_{unif.noise}$ of points that have a distance to surface greater than $d_{trunc} = 0.16$m. Having decoded the projection of the shape $\{\tilde{\mathbf{z}}^{\mathrm{s}}\}$, we uniformly sample points around decoded shape no closer than 0.2m to the surface of this shape, and assign truncation distance d_{trunc} to these points. We also add them to the set $\mathcal{A}_{unif.noise}$. For the shape decoder and for the set $\mathcal{A}_{unif.noise}$ we set $w_{trunc} = 5.0$ ('w_{trunc} (shape unif. noise)' in the Table 9); for part decoder we set $w_{trunc} = 20.0$ ('w_{trunc} (parts unif. noise)'). Additionally, while optimizing the particular part k during test-time optimization we also use the points corresponding to other parts as noise with distance d_{trunc} and denote this set of points as $\mathcal{A}_{partnoise}$. Adding this set into optimization is necessary to decrease intersections between different part geometries after test-time optimization. We set $w_{trunc} = 5.0$ ('w_{trunc} (part noise)') for this set of points. Finally, in Eq. (4) we use an additional weight for loss consistency term, for which we set $w_{cons} = 200.0$.

To encourage geometric completeness during test-time optimization, we sample points with distances to surface from the decoded shape $\mathcal{S}$ and parts $\{\mathcal{P}_k\}$ (decoded from $\{\tilde{\mathbf{z}}^{\mathrm{s}}\}$ and $\{\tilde{\mathbf{z}}^{\mathrm{p}}_k\}$), and add them to TSDF D ('add pts. to shape' in the Table 9) or $\{D^{\mathrm{p}}\}_{\mathrm{p}=1}^{N_{parts}}$ ('add pts. to parts') to the regions where points in $\mathcal{S}$ or $\{\mathcal{P}_k\}$ are present and non-background points with distance $d < d_{trunc}$ in D and $\{D^{\mathrm{p}}\}_{\mathrm{p}=1}^{N_{parts}}$ (which we call *meaningful* points) are missing. We add only those points from $\mathcal{S}$ and $\{\mathcal{P}_k\}$ which are not closer than d_{thr} to meaningful points.

Finally, we scale the coordinates of input TSDF with a scale factor ('scale factor') to align better to the learned canonical space of synthetic shapes.

Optimization for each part takes approximately 25 seconds.

G.5. Hyperparameters search

We determined hyperparameters for training and TTO on a hold-out validation set. There are parameters that affect training and TTO more than others, such as the number of decoder layers to use parameters from for TTO for both shape and part decoders and learning rate for both training and TTO. Parameters of decoder layers enable more flexibility in optimization, but decrease implicit regularization from learned part priors, resulting in less geometry consistency. The proper choice of the learning rate for TTO is important for accurate geometry reconstruction and avoiding unstable optimization. All weights for the loss components in TTO have a wide range of appropriate parameters (± 20 for w_{trunc} and ± 50 for w_{cons}).

G.6. Network architecture

We also provide extensive information about the architecture of every submodel that we use in our framework. Table 10 shows the architecture of voxel encoder that we use to encode an input occupancy grid. Table 11 shows the architecture of a module that predicts the part decomposition of an input object. The architectures of a small PointNet-like [44] network and a big PointNet-like network that we use to segment an input TSDF are shown in Tables 12, 13. Finally, we provide details about the architecture of shape and parts MLP decoders in Tables 14, 15.

Task	Parameter	Chair	Table	Cabinet	Bookshelf	Bed	Trashcan
Train decoder (shape)	# epochs	2000	2400	8000	8000	16000	16000
Train decoder (shape)	batch size	24	24	24	24	24	24
Train decoder (shape)	optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Train decoder (shape)	lr (weights)	5e-5	1e-4	1e-4	1e-4	1e-4	1e-4
Train decoder (shape)	lr factor (weights)	0.5	0.5	0.5	0.5	0.5	0.5
Train decoder (shape)	lr decay int. (weights)	500	600	1500	1500	4000	3000
Train decoder (shape)	lr (lat.)	1e-4	2e-4	2e-4	2e-4	2e-4	2e-4
Train decoder (shape)	lr factor (lat.)	0.5	0.5	0.5	0.5	0.5	0.5
Train decoder (shape)	lr decay int. (lat.)	500	600	1500	1500	4000	3000
Train decoder (parts)	# epochs	1100	1400	2000	2000	10000	10000
Train decoder (parts)	batch size	48	48	48	48	48	48
Train decoder (parts)	optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Train decoder (parts)	lr (weights)	5e-5	1e-4	1e-4	1e-4	1e-4	1e-4
Train decoder (parts)	lr factor (weights)	0.5	0.5	0.5	0.5	0.5	0.5
Train decoder (parts)	lr decay int. (weights)	400	400	800	800	3600	3600
Train decoder (parts)	lr (lat.)	1e-4	2e-4	2e-4	2e-4	2e-4	2e-4
Train decoder (parts)	lr factor (lat.)	0.5	0.5	0.5	0.5	0.5	0.5
Train decoder (parts)	lr decay int. (lat.)	400	400	800	800	3600	3600
Train projection	# epochs	35	30	60	60	250	200
Train projection	batch size	64	64	64	64	64	64
Train projection	optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Train projection	lr	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
Train projection	lr factor	0.5	0.5	0.5	0.5	0.5	0.5
Train projection	lr decay int.	12	20	40	40	150	120
Train projection	segm. network	PN-small	PN-big	PN-big	PN-big	PN-big	PN-big
Fine-tune	# epochs	80	80	120	120	125	70
Fine-tune	batch size	64	64	64	64	64	64
Fine-tune	optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Fine-tune	lr	2e-4	2e-4	2e-4	2e-4	2e-4	2e-4
Fine-tune	lr factor	0.2	0.2	0.2	0.2	0.2	0.2
Fine-tune	lr decay int.	40	40	60	60	60	40
Test-time opt.	# iterations	500	500	500	500	500	500
Test-time opt.	optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Test-time opt.	lr (lat.)	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4
Test-time opt.	lr factor (lat.)	0.1	0.1	0.1	0.1	0.1	0.1
Test-time opt.	lr decay int. (lat.)	300	300	300	300	300	300
Test-time opt.	shape dec. layers opt.	-	-	-	-	1,2	1,2
Test-time opt.	lr (shape dec.)	-	-	-	-	1e-4	1e-4
Test-time opt.	lr factor (shape dec.)	-	-	-	-	0.1	0.1
Test-time opt.	lr decay int. (shape dec.)	-	-	-	-	300	300
Test-time opt.	part dec. layers opt.	1,2	1,2	1,2	1,2	-	-
Test-time opt.	lr (part dec.)	3e-4	3e-4	3e-4	3e-4	-	-
Test-time opt.	lr factor (part dec.)	0.1	0.1	0.1	0.1	-	-
Test-time opt.	lr decay int. (part dec.)	300	300	300	300	-	-
Test-time opt.	w_{trunc} (shape unif. noise)	5.0	3.0	3.0	3.0	1.0	1.0
Test-time opt.	w_{trunc} (parts unif. noise)	20.0	12.0	12.0	12.0	1.0	5.0
Test-time opt.	w_{trunc} (part noise)	5.0	3.0	10.0	10.0	10.0	5.0
Test-time opt.	w_{cons}	200.0	300.0	300.0	300.0	30.0	200.0
Test-time opt.	add pts. to shape	✓	✓	✓	✓	✓	-
Test-time opt.	dist thr. (shape)	0.16m	0.16m	0.5m	0.5m	0.75m	-
Test-time opt.	add pts. to parts	-	✓	✓	✓	✓	-
Test-time opt.	dist thr. (part)	-	0.16m	0.5m	0.5m	0.75m	-
Test-time opt.	scale factor	1.2	1.2	1.1	1.1	1.2	1.2

Table 9. Hyperparameters used for training submodels used in our framework.

Encoder	Input Layer	Type	Input Size	Output Size	Kernel Size	Stride	Padding
conv0	scan occ. grid	Conv3D	(1, 32, 32, 32)	(32, 16, 16, 16)	(5, 5, 5)	(2, 2, 2)	(2, 2, 2)
gnorm0	conv0	GroupNorm	(32, 16, 16, 16)	(32, 16, 16, 16)	-	-	-
relu0	gnorm0	ReLU	(32, 16, 16, 16)	(32, 16, 16, 16)	-	-	-
pool1	relu0	MaxPooling	(32, 16, 16, 16)	(32, 8, 8, 8)	(2, 2, 2)	(2, 2, 2)	(0, 0, 0)
conv1	pool1	Conv3D	(32, 8, 8, 8)	(64, 8, 8, 8)	(3, 3, 3)	(1, 1, 1)	(1, 1, 1)
gnorm1	conv1	GroupNorm	(64, 8, 8, 8)	(64, 8, 8, 8)	-	-	-
relu1	gnorm1	ReLU	(64, 8, 8, 8)	(64, 8, 8, 8)	-	-	-
pool2	relu1	MaxPooling	(64, 8, 8, 8)	(64, 4, 4, 4)	(2, 2, 2)	(2, 2, 2)	(0, 0, 0)
conv2	pool2	Conv3D	(64, 4, 4, 4)	(128, 2, 2, 2)	(5, 5, 5)	(2, 2, 2)	(2, 2, 2)
gnorm2	conv2	GroupNorm	(128, 2, 2, 2)	(128, 2, 2, 2)	-	-	-
relu2	gnorm2	ReLU	(128, 2, 2, 2)	(128, 2, 2, 2)	-	-	-
pool3	relu2	MaxPooling	(128, 2, 2, 2)	(128, 1, 1, 1)	(2, 2, 2)	(2, 2, 2)	(0, 0, 0)
conv3	pool3	Conv3D	(128, 1, 1, 1)	(256, 1, 1, 1)	(3, 3, 3)	(1, 1, 1)	(1, 1, 1)
gnorm3	conv3	GroupNorm	(256, 1, 1, 1)	(256, 1, 1, 1)	-	-	-
relu3	gnorm3	ReLU	(256, 1, 1, 1)	(256, 1, 1, 1)	-	-	-
shape feature	relu3	Flatten	(256, 1, 1, 1)	(256)	-	-	-

Table 10. Layer specification for detected object encoder.

Child decoder	Input Layer	Type	Input Size	Output Size
lin_proj	shape feature	ReLU(Linear)	256	256
node feature	lin_proj	ReLU(Linear)	256	256
lin0	node feature	Linear	256	2560
relu0	lin0	ReLU	2560	2560
reshape0	relu0	Reshape	2560	(10, 256)
node_exist	reshape0	Linear	(10, 256)	(10, 1)
concat0	(reshape0, reshape0)	Concat.	(10, 256), (10, 256)	(10, 10, 512)
lin1	concat0	Linear	(10, 10, 512)	(10, 10, 256)
relu1	lin1	ReLU	(10, 10, 256)	(10, 10, 256)
edge_exist	relu1	Linear	(10, 10, 256)	(10, 10, 1)
mp	(relu1, edge_exist, reshape0)	Mes. Passing	(10, 10, 256), (10, 10, 1), (10, 256)	(10, 768)
lin2	mp	Linear	(10, 768)	(10, 256)
relu2	lin2	ReLU	(10, 256)	(10, 256)
node_sem	relu2	Linear	(10, 256)	(10, #classes)
lin3	relu2	Linear	(10, 256)	(10, 256)
(10, child feature)	lin3	ReLU	(10, 256)	(10, 256)
lin4	node feature	ReLU(Linear)	256	256
rotation_cls	lin3	Linear	256	12

Table 11. Layer specification for decoding an object into its semantic part structure.

Pts. classifier (small)	Input Layer	Type	Input Size	Output Size
input feature	(TSDF, node feature, rotation_cls)	Concat.	(#pts, 4), 256, 12	(#pts, 272)
lin_cls_0	input feature	ReLU(Linear)	(#pts, 272)	(#pts, 128)
lin_cls_1	lin_cls_0	ReLU(Linear)	(#pts, 128)	(#pts, 128)
lin_cls_2	lin_cls_1	ReLU(Linear)	(#pts, 128)	(#pts, 128)
glob_feat_0	lin_cls_2	MaxPooling1D	(#pts, 128)	(1, 128)
glob_feat_1	glob_feat_0	Repeat	(1, 128)	(#pts, 128)
lin_cls_3	(lin_cls_1, glob_feat_1)	Concat	(#pts, 128), (#pts, 128)	(#pts, 256)
lin_cls_4	lin_cls_3	ReLU(Linear)	(#pts, 256)	(#pts, 128)
lin_cls_5	lin_cls_4	ReLU(Linear)	(#pts, 128)	(#pts, 128)
lin_cls_6	lin_cls_5	Linear	(#pts, 128)	(#pts, #classes)

Table 12. Layer specification for segmenting input TSDF using small PointNet-like network.

Pts. classifier (big)	Input Layer	Type	Input Size	Output Size
input feature	(TSDF, node feature, rotation_cls)	Concat.	(#pts, 4), 256, 12	(#pts, 272)
lin_cls_0	input feature	ReLU(Linear)	(#pts, 272)	(#pts, 256)
lin_cls_1	lin_cls_0	ReLU(Linear)	(#pts, 256)	(#pts, 128)
lin_cls_2	lin_cls_1	ReLU(Linear)	(#pts, 128)	(#pts, 128)
glob_feat_0	lin_cls_2	MaxPooling1D	(#pts, 128)	(1, 128)
glob_feat_1	glob_feat_0	Repeat	(1, 128)	(#pts, 128)
lin_cls_3	lin_cls_2	ReLU(Linear)	(#pts, 128)	(#pts, 64)
lin_cls_4	lin_cls_3	ReLU(Linear)	(#pts, 64)	(#pts, 64)
glob_feat_2	lin_cls_4	MaxPooling1D	(#pts, 64)	(1, 64)
glob_feat_3	glob_feat_2	Repeat	(1, 64)	(#pts, 64)
lin_cls_5	(lin_cls_1, glob_feat_1, glob_feat_3)	Concat	(#pts, 128), (#pts, 128), (#pts, 64)	(#pts, 320)
lin_cls_6	lin_cls_5	ReLU(Linear)	(#pts, 320)	(#pts, 128)
lin_cls_7	lin_cls_6	ReLU(Linear)	(#pts, 128)	(#pts, 64)
lin_cls_8	lin_cls_7	Linear	(#pts, 64)	(#pts, #classes)

Table 13. Layer specification for segmenting input TSDF using big PointNet-like network.

Implicit decoder	Input Layer	Type	Input Size	Output Size
lin_proj_0	node feature	ReLU(Linear)	256	512
lin_proj_1	lin_proj_0	ReLU(Linear)	512	512
lin_proj_2	lin_proj_1	ReLU(Linear)	512	512
lin_proj_3	lin_proj_2	ReLU(Linear)	512	512
lin_proj_4	lin_proj_3	Linear	512	256
lin_pts_0	(lin_proj_4, TSDF pts.)	Concat.	256, 63	319
lin_pts_1	lin_pts_0	Linear	319	512
lin_bn_1	lin_pts_1	BatchNorm	512	512
lin_relu_1	lin_bn_1	ReLU	512	512
lin_drop_1	lin_relu_1	Dropout	512	512
lin_pts_2	lin_pts_1	Linear	512	512
lin_bn_2	lin_pts_2	BatchNorm	512	512
lin_relu_2	lin_bn_2	ReLU	512	512
lin_drop_2	lin_relu_2	Dropout	512	512
lin_pts_3	lin_pts_2	Linear	512	512
lin_bn_3	lin_pts_3	BatchNorm	512	512
lin_relu_3	lin_bn_3	ReLU	512	512
lin_drop_3	lin_relu_3	Dropout	512	512
lin_pts_4	lin_pts_3	Linear	512	512 - dim(lin_pts_0)
lin_bn_4	lin_pts_4	BatchNorm	512 - dim(lin_pts_0)	512 - dim(lin_pts_0)
lin_relu_4	lin_bn_4	ReLU	512 - dim(lin_pts_0)	512 - dim(lin_pts_0)
lin_drop_4	lin_relu_4	Dropout	512 - dim(lin_pts_0)	512 - dim(lin_pts_0)
lin_pts_5	(lin_pts_0, lin_drop_4)	Concat.	dim(lin_pts_0), 512 - dim(lin_pts_0)	512
lin_bn_5	lin_pts_5	BatchNorm	512	512
lin_relu_5	lin_bn_5	ReLU	512	512
lin_drop_5	lin_relu_5	Dropout	512	512
lin_pts_6	lin_pts_5	Linear	512	512
lin_bn_6	lin_pts_6	BatchNorm	512	512
lin_relu_6	lin_bn_6	ReLU	512	512
lin_drop_6	lin_relu_6	Dropout	512	512
lin_pts_7	lin_pts_6	Linear	512	512
lin_bn_7	lin_pts_7	BatchNorm	512	512
lin_relu_7	lin_bn_7	ReLU	512	512
lin_drop_7	lin_relu_7	Dropout	512	512
lin_pts_8	lin_pts_7	Linear	512	1
lin_tanh_7	lin_pts_8	Tanh	1	1

Table 14. Layer specification for implicit shape decoder.

Implicit decoder	Input Layer	Type	Input Size	Output Size
lin_proj_0	child feature	ReLU(Linear)	256	512
lin_proj_1	lin_proj_0	ReLU(Linear)	512	512
lin_proj_2	lin_proj_1	ReLU(Linear)	512	512
lin_proj_3	lin_proj_2	ReLU(Linear)	512	512
lin_proj_4	lin_proj_3	Linear	512	256
lin_pts_0	(lin_proj_4, part cls. one-hot, TSDF pts.)	Concat.	256, #parts, 63	319 + #parts
lin_pts_1	lin_pts_0	Linear	319 + #parts	512
lin_bn_1	lin_pts_1	BatchNorm	512	512
lin_relu_1	lin_bn_1	ReLU	512	512
lin_drop_1	lin_relu_1	Dropout	512	512
lin_pts_2	lin_pts_1	Linear	512	512
lin_bn_2	lin_pts_2	BatchNorm	512	512
lin_relu_2	lin_bn_2	ReLU	512	512
lin_drop_2	lin_relu_2	Dropout	512	512
lin_pts_3	lin_pts_2	Linear	512	512
lin_bn_3	lin_pts_3	BatchNorm	512	512
lin_relu_3	lin_bn_3	ReLU	512	512
lin_drop_3	lin_relu_3	Dropout	512	512
lin_pts_4	lin_pts_3	Linear	512	512 - dim(lin_pts_0)
lin_bn_4	lin_pts_4	BatchNorm	512 - dim(lin_pts_0)	512 - dim(lin_pts_0)
lin_relu_4	lin_bn_4	ReLU	512 - dim(lin_pts_0)	512 - dim(lin_pts_0)
lin_drop_4	lin_relu_4	Dropout	512 - dim(lin_pts_0)	512 - dim(lin_pts_0)
lin_pts_5	(lin_pts_0, lin_drop_4)	Concat.	dim(lin_pts_0), 512 - dim(lin_pts_0)	512
lin_bn_5	lin_pts_5	BatchNorm	512	512
lin_relu_5	lin_bn_5	ReLU	512	512
lin_drop_5	lin_relu_5	Dropout	512	512
lin_pts_6	lin_pts_5	Linear	512	512
lin_bn_6	lin_pts_6	BatchNorm	512	512
lin_relu_6	lin_bn_6	ReLU	512	512
lin_drop_6	lin_relu_6	Dropout	512	512
lin_pts_7	lin_pts_6	Linear	512	512
lin_bn_7	lin_pts_7	BatchNorm	512	512
lin_relu_7	lin_bn_7	ReLU	512	512
lin_drop_7	lin_relu_7	Dropout	512	512
lin_pts_8	lin_pts_7	Linear	512	1
lin_tanh_7	lin_pts_8	Tanh	1	1

Table 15. Layer specification for implicit part decoder.