BuildingNet: Learning to Label 3D Buildings
-Supplementary Material-

Appendix A: Building collection

Mining building models. We used the Trimble 3D Warehouse repository [5] to mine 3D building models. Specifically, we used keywords denoting various building categories, following a snapshot from Wikipedia's article on "list of building types" [7]. The article contained 181 common building types, such as "house", "hotel", "skyscraper", "church", "mosque", "city hall", "castle", "office building", and so on, organized into basic categories, such as residential, commercial, industrial, agricultural, military, religious, educational, and governmental buildings. For each keyword, we retrieved the first $10K$ models. Since some keyword searches returned much fewer buildings, and since identical models were retrieved across different searches (e.g., a building can have both tags "house" and "villa"), we ended up with $48,439$ models. The models were stored in the COLLADA file format.

Mesh-based filtering. Low-poly meshes often represent low-quality or incomplete buildings, and they often cause problems in rendering and geometry processing. Thus, we removed models with less than $3K$ faces and also removed models with extremely large number of faces (more than $1M$ faces) that tend to significantly slow down mesh processing and rendering for interactive segmentation (total $13,628$ models removed). Since our UI relies on labeling mesh subgroups (submeshes) stored in the leaf nodes of the COLLADA hierarchy, we excluded under-segmented models with less than 50 mesh subgroups, and over-segmented models with more than $5K$ mesh subgroups, which would be more challenging to label (total $4,958$ models removed). As a result, the filtered dataset contained $29,853$ models.

Crowdworker-based filtering. The above keyword searches can be affected by noisy metadata, such as erroneous and irrelevant tags not describing the actual shape class. As a result, most of the retrieved models did not represent buildings. Some models also contained entire neighborhoods or multiple buildings. Thus, our next step was to filter 3D models that did not represent single buildings. We resorted to crowdworkers from Amazon Mechanical Turk (MTurk) to verify whether each model is a single building or not, and also classify it into basic categories following Wikipedia's categorization. To this end, we created web questionnaires showing each model from four viewpoints with elevation 0 degrees from the ground plane, and azimuth difference 90^o degrees. We asked MTurk participants (MTurkers) to select a category that best describe the model (see Figure 1 for an example of rendered views, and basic categories we used). We instructed them to answer "can't tell" if the displayed model did not represent a single building, or when they could not recognize it.

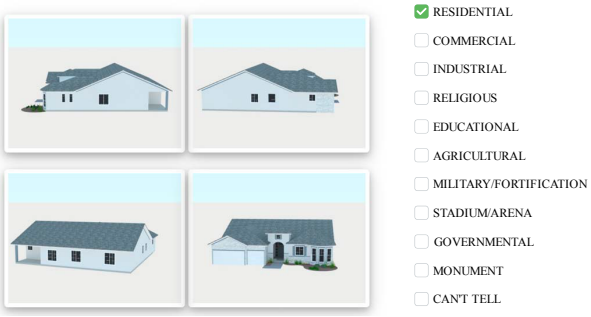

Figure 1: Web questionnaire for classifying a model into basic building categories

Each participant was asked to complete a questionnaire with 20 queries randomly picked from our filtered set of models. Each query showed one model (Figure 1). Queries were shown in a random order. Each query was repeated twice in the questionnaire in a random order to detect unreliable participants providing inconsistent answers (i.e., we had 10 unique queries per questionnaire). We filtered out unreliable MTurk participants who gave two inconsistent answers to more than 3 out of the 10 unique queries in the questionnaire. Each participant was allowed to answer one questionnaire at most to ensure participant diversity. We had total $4,344$ different, reliable MTurk participants in this study. For each of the models, we gathered consistent votes from 7 different MTurk participants. We accepted a building category for a model, if it was voted by at least 5 out of 7 MTurkers. We note that this majority is statistically significant: given 10 categories, the probability of a model getting 5 out of 7 votes given random answers is negligible accord-

Table 1: From left to right: number of models per basic building category after filtering (original buildings), number of buildings whose parts were labeled by crowdworkers in our dataset (labeled buildings), number and percentage of training, hold-out validation and test buildings

Category	#orig. build.	#label. build.	# train. (%)	# val. (%)	# test (%)
Residential	1424	1266	1007 (62.9%)	133 (66.5%)	126 (63.0)%
Commercial	153	131	104 (6.5%)	16 (8.0%)	11 (5.5%)
Religious	540	469	386 (24.1%)	38 (19.0%)	45 (22.5%)
Civic	67	61	45 (2.8%)	8 (4.0%)	8 (4.0%)
Castles	85	73	58 (3.6%)	5 (2.5%)	10 (5.0%)
Total:	2286	2,000	1600 (80%)	200 (10%)	200 (10%)

ing to a binomial test ($p < 0.001$). We removed models lacking majority votes (i.e., they were not buildings, or the category could not be determined with high agreement).

The categories "agricultural", "industrial", "stadium" had less than 40 buildings, thus, we decided to exclude them since their part variability and corresponding labels, would not be sufficiently represented in training, validation, and test splits of the segmentation dataset. We also decided to merge the "educational" and "governmental" buildings into a single broader category, called "civic" buildings commonly used to characterize both types of buildings, since we observed that the exterior of a governmental building (e.g., town hall) is often similar to the exterior of an educational one (e.g., public library or college). The remaining number of models characterized as buildings from our study was $2,286$. We note that all models in our dataset are stored as COLLADA files, and have hierarchy tree depth ≥ 2 (excl. the root). We refer the reader to Table 1 for statistics per basic category in our dataset and its splits.

Mesh pre-processing. The meshes in the above dataset were pre-processed to (a) *detect and remove interior structure* for each building (since we aimed to gather annotations of building exteriors), (b) *detect exact duplicates of subgroups* useful for label propagation, as discussed in Section 3.1 (interface for labeling) in our main paper. To detect whether a subgroup is interior, we sample 10 points per each triangle in the subgroup and shoot rays to 50 external viewpoints from all these sample points. If a single ray escapes from the subgroup, it is marked as external, otherwise it is internal. We remove all subgroups marked as internal. For duplicate detection, we process all-pairs of subgroups in a building. Specifically, for each pair of subgroups, we exhaustively search for upright axis rotations minimizing Chamfer distance. The optimal translation is computed from the difference of the vertex location barycenters. After factoring out the rigid transformation, we compute one-to-one vertex correspondences based on closest pairs in Euclidean space. If all closest pairs have distance less than 10^{-6} of the average OBB diagonals of the subgroups, we also check if their mesh connectivity matches i.e., the subgroup mesh adjacency matrix is the same given the corresponding vertices. If they match, the pair is marked as duplicate. Finally, all such pairs are merged into sets containing subgroups found to be duplicates of each other.

Table 2: Statistics regarding mesh resolution in our dataset. From left to right: building category, average/median number of faces and vertices.

Category	avg. # faces	med. # faces	avg. # vertices	med. # vertices
Residential	58,522.7	32,295.5	18,830.6	10,684.0
Commercial	49,248.5	28,862.0	16,722.6	10,041.0
Religious	51,882.7	25,979.0	16,687.4	8,654.0
Civic	40,380.1	20,512.0	13,910.2	7,281.0
Castles	70,731.2	26,493.0	21,050.0	8,822.0
Whole Set	56,250.4	29,741.5	18,120.9	9,845.0

Appendix B: Part labels

To determine a set of *common* labels used to identify parts in buildings, we created a variant of our UI that asked users to explicitly type tags for selected components instead of selecting labels from a predefined list. We gathered tags from people who have domain expertise in the fields of building construction or design. Specifically, we asked 10 graduate students in civil engineering and architecture to tag components in a set of 100 buildings uniformly distributed across the different categories. Each student labeled 3-10 different buildings. We selected tags that appeared at least in 0.5% of the labeled components to filter out uncommon tags. We concatenated the remaining tags with the most frequent tags appearing in the COLLADA leaf nodes (appearing at least in 0.5% of subgroups). We merged synonyms and similar tags.

The resulting list had 39 tags. During the main phase of annotation of our 2K buildings, 8 tags were used very sparsely: less than 0.05% subgroups throughout the dataset were annotated with these tags: "ramp", "canopy", "tympanum", "crepidoma", "entablature", "pediment", "bridge", and "deck". We decided them to exclude them from our dataset since the number of train or test subgroups with these labels would be too low (less than 10, or they existed in only one building). Any subgroups annotated with these tags were considered as "unlabeled" (undetermined) ones.

Appendix C: Additional dataset statistics

As discussed in our main paper, we gathered $10,000$ annotations from qualified MTurkers for $2,000$ buildings (5 annotations per building). Table 2 shows statistics on the polygon resolution of the meshes in our 2K dataset. Table 3 reports the worker consistency per part label, which is

Table 3: Worker consistency for each different part label.

Label	Worker consistency
Window	93.4%
Plant	98.7%
Wall	88.2%
Roof	88.7%
Banister	86.5%
Vehicle	99.2%
Door	84.9%
Fence	85.9%
Furniture	95.1%
Column	87.2%
Beam	76.3%
Tower	81.4%
Stairs	92.0%
Shutters	79.3%
Ground	84.8%
Garage	86.9%
Parapet	82.6%
Balcony	75.9%
Floor	79.1%
Buttress	85.0%
Dome	83.5%
Corridor	70.6%
Ceiling	78.4%
Chimney	93.3%
Gate	90.8%
Lighting	90.9%
Dormer	70.4%
Pool	86.8%
Road	73.8%
Arch	72.1%
Awning	59.5%

measured as the percentage of times that a subgroup label selected by a qualified MTurker agrees wit the majority. Table 4 reports the worker consistency per building category for the training, hold-out validation, and test split. We observe that the worker consistency remains similar across all splits and building categories.

Table 5 reports statistics on the number of subgroups per building category, unique subgroups (counting repeated subgroups with exactly the same mesh geometry as one unique subgroup), and number of annotated subgroups. We note that there were often subgroups that represented tiny, obscure pieces (e.g., subgroups with a few triangles covering a tiny area of a wall, beam, or frame), and these were often not labeled by annotators. As we explained in the main paper, most of the buildings had more than 80% of their area labeled (and all had $> 50\%$ labeled area). Table 6 presents more statistics on the labeled components (merged, adjacent subgroups with the same label) of the $2K$ building dataset per each basic category.

Appendix D: Network and experiments details

BuildingGNN. We provide more details about the structure of the BuildingGNN network architecture in Table 7.

Table 4: Worker consistency in the training, hold-out validation, test split, and our whole dataset per category.

Category	Worker Consistency			
	train.	val.	test	all
Residential	92.2%	93.3%	91.5%	92.2%
Commercial	87.7%	89.4%	95.1%	88.6%
Religious	91.4%	91.7%	91.7%	91.5%
Civic	93.6%	98.8%	98.0%	94.9%
Castles	94.1%	88.8%	88.3%	92.9%
Average:	**91.8%**	**92.4%**	**92.9%**	**92.0%**

Table 8 presents statistics on the number of edges per type used in BuildingGNN for our training set.

MinkNet-GC. As mentioned in the experiments section of our main paper, we implemented a simple graph-cuts variant, called MinkNet-GC, that incorporates label probabilities from MinkowskiUNet34 as unary terms, and a pairwise term that depends on angles between triangles, inspired by [3]. Specifically, we use the following energy that we minimize using [1]:

$$E(\boldsymbol{y}) = \sum_{i \in \mathcal{F}} \psi(y_i) + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{N}(i)} \phi(y_i, y_j) \qquad (1)$$

where $\boldsymbol{y} = \{y_i\}$ are the label assignments we wish to compute by minimizing the above energy, $\mathcal{F}$ is the set of faces in a mesh, and $\mathcal{N}(i)$ are the adjacent faces of each face i. The unary term is expressed as follows: $\psi(y_i) = -\log f(y_i)$, where $f(y_i)$ is the probability distribution over part labels associated with the face i produced through average pooling of probabilities computed from MinkowskiUNet34 on the triangle's associated points. The pairwise term uses angles between face normals, $\phi'(y_i, y_j) = -\lambda' \cdot \log(\min(\omega_{i,j}/90^o, 1))$, for $y_i \neq y_j$, where $\omega_{i,j}$ is the angle between the normals of faces i, j. The term results in zero cost for right angles between normals indicating a strong edge. The parameter λ is adjusted with grid search in the hold-out validation set.

Average vs max pooling. As discussed in our experiments section of our main paper, one possibility to aggregate probabilities of points associated per triangle or component is average pooling: $\mathbf{q}_t = \sum_{p \in P_t} \mathbf{q}_p / |P_t|$ where $\mathbf{q}_p$ and $\mathbf{q}_t$ are point and triangle probabilities respectively. An alternative is to use max pooling (i.e., replace sum with max above). We experimented with average vs max pooling also per component. As shown in Table 9, average pooling works better for both triangle- and component-based pooling (we experimented with MinkowskiNet per-point probabilities).

Experiments with different losses. We experimented with different losses for our MinkowskiNet variants for the "BuildingNet-Point" and "BuildingNet-Mesh" tracks.

Table 5: Statistics for each building category. From left to right: building category, total number of models, average/median/minimum/maximum number of mesh subgroups over the category's models (leaf nodes of the COLLADA metadata of the building models), average/median/minimum/maximum number of unique (non-duplicate) subgroups, average/median/minimum/maximum number of annotated unique mesh subgroups.

Category	num# models	avg# subgrps	med# subgrps	min# subgrps	max# subgrps	avg# un. subgrps	med# un. subgrps	min# un. subgrps	max# un. subgrps	avg# un. l.subgrps	med# un. l.subgrps	min# un. l.subgrps	max# un. l.subgrps
Residential	1,424	678.7	547	83	1989	167.1	144	61	920	61.4	50.0	7	613
Commercial	153	723.4	606	90	1981	159.8	139	70	907	49.4	44.0	3	223
Religious	540	487.0	348	93	1981	139.9	129	65	667	47.2	45.0	7	139
Civic	67	628.8	480	118	1822	144.4	123	75	618	43.0	43.0	8	106
Castles	85	609.8	485	125	1786	193.0	166	76	590	38.6	37	2	92
Whole Set	2,000	623.6	497.5	83	1989	160.5	140	61	920	55.9	47.0	2	613

Table 6: Statistics per building category regarding components (merged adjacent mesh subgroups). From left to right: building category, total number of models, average/median/minimum/maximum number of annotated components per model, average/median/minimum/maximum number of annotated unique (non-duplicate) components per model.

Category	num# models	avg# l.comp.	med# l.comp.	min# l.comp.	max# l.comp.	avg# un. l.comp.	med# un. l.comp.	min# un. l.comp.	max# un. l.comp.
Residential	1,424	321.8	243.0	13	1970	46.1	42.0	8	371
Commercial	153	408.0	296.0	4	1680	44.6	39.0	3	247
Religious	540	272.2	184.0	18	1469	37.7	35.0	6	135
Civic	67	378.4	263.0	36	1667	39.3	33.0	7	252
Castles	85	295.3	210.0	40	1200	30.5	28.0	2	107
Whole Set	2,000	316.6	231.0	4	1970	43.2	39.0	2	371

Table 7: BuildingGNN architecture: The Node representation combines the OBB - (Object Oriented Bounding Box) , SA - (Surface area), C - (centroid) and MN - (MinkowskiNet pre-trained features) for each sub group. The GNN is composed of (a) an encoder block made of three MLPs having 1, 3 and 5 hidden layers respectively, and (b) a decoder block with one MLP having 1 hidden layer followed by softmax. We refer to the code for more details.

	Layers	Output
Edge	(MLP(11×41, layer=1)))	41
Node	(6D(OBB)+1D(SA)+3D(C)+31D(MN)	41
Input	(Node_i + Edge_{ij} + Node_j)	41
Encoder	(MLP(Input×256, layer=1)))	64
	GN(LeakyReLU(0.2)))	64
	(MLP(64*3×128, layer=3)))	64
	GN(LeakyReLU(0.2)))	64
	(MLP(64*3×128, layer=5)))	64
	GN(LeakyReLU(0.2)))	64
Decoder	(MLP(128×64, layer=1)))	31
	softmax	31

Table 8: Statistics for the number of BuildingGNN edges per type present in the graphs of the training buildings.

Label	max # edges	min # edges	mean # edges	# median edges
Proximity	16317	81	778.0	489.0
Similarity	762156	5	26452.1	4875.5
Containment	26354	71	2,054.5	1,390.0
Support	7234	7	687.5	492.0
All	772878	259	29972.1	7818.0

Table 9: "BuildingNet-Mesh" results using average and max pooling aggregation over triangles and components (weighted cross-entropy loss was used for all these experiments).

Method	Pool.	n?	c?	Part IoU	Shape IoU	Class acc.
MinkNet2Triangle	Avg	✓	×	**28.8%**	**26.7%**	**64.8%**
	Max	✓	×	28.6%	26.1%	64.4%
	Avg	✓	✓	**32.8%**	**29.2%**	**68.1%**
	Max	✓	✓	31.5%	28.1%	66.8%
MinkNet2Sub	Avg	✓	×	**33.1%**	**36.0%**	**69.9%**
	Max	✓	×	30.4%	32.4%	65.6%
	Avg	✓	✓	**37.0%**	**39.1%**	**73.2%**
	Max	✓	✓	32.7%	34.8%	67.4%

Specifically, we experimented with the Weighted Cross-Entropy Loss (WCE) described in our main paper, Cross-Entropy Loss (CE) without label weights, the Focal Loss (FL) [4], α-balanced Focal Loss (α-FL) [4], and Class-Balanced Cross Entropy Loss (CB) [2]. Table 10 and Table 11 show results for the "BuildingNet-Point" and "BuildingNet-Mesh" tracks respectively. We observe that (a) in the case that color is not available, WCE is slightly better than alternatives according to all measures for both tracks (b) when color is available, CB is a bit better in terms of Part IoU, but worse in terms of Shape IoU than WCE in the case of the point cloud track. For the mesh track, CB is

slightly better according to all measures. In general, WCE and CB behave the best on average, yet their difference is small. For the rest of our experiments, we use WCE.

Performance for each part label. Our main paper reports mean Part IoU performance in the experiments section. Table 14 reports the BuildingGNN-PointNet++ and BuildingGNN-MinkNet part IoU performance for each label. We also report the performance of MinkowskiNet and PointNet++ for the point cloud track. We observe that networks do better for common part labels, such as window, wall, roof, plant, vehicle, while the performance degrades for rare parts (e.g., awning, arch), or parts whose shape can easily be confused with other more dominant parts (e.g., garage is often confused with door, wall, or window).

Table 10: "BuildingNet-Point" track results using the Weighted Cross-Entropy Loss (WCE), Cross-Entropy Loss (CE), Focal Loss (FL), α-balanced Focal Loss (α-FL) and finally Class-Balanced Cross Entropy Loss (CB). All these were used to train the MinkowskiUNet34 architecture. For the FL and α-FL experiments the γ hyper-parameter was set to 2.0 and for the α-FL the same weights were used as the weighted cross entropy loss (see Section 4.3 in our main paper). For the CB experiments we set $\beta = 0.999999$.

Method	Loss	n?	c?	Part IoU	Shape IoU	Class acc.
MinkNet	WCE	✓	×	**26.9%**	**22.2%**	**62.2%**
	CE	✓	×	24.5%	21.2%	61.3%
	FL	✓	×	26.1%	21.8%	61.2%
	α-FL	✓	×	22.3%	19.8%	61.5%
	CB	✓	×	26.4%	20.9%	61.4%
MinkNet	WCE	✓	✓	29.9%	24.3%	**65.5%**
	CE	✓	✓	28.5%	24.5%	65.3%
	FL	✓	✓	28.7%	24.9%	65.2%
	α-FL	✓	✓	30.1%	**25.3%**	65.2%
	CB	✓	✓	**30.4%**	24.0%	**65.5%**

Table 11: "BuildingNet-Mesh" results using different loss functions

Method	Loss	n?	c?	Part IoU	Shape IoU	Class acc.
MinkNet2Sub	WCE	✓	×	**33.1%**	**36.0%**	**69.9%**
	CE	✓	×	30.7%	32.7%	68.8%
	FL	✓	×	31.0%	33.4%	67.9%
	α-FL	✓	×	27.2%	28.3%	66.7%
	CB	✓	×	32.9%	34.3%	69.1%
MinkNet2Sub	WCE	✓	✓	37.0%	39.1%	73.2%
	CE	✓	✓	35.6%	39.2%	73.5%
	FL	✓	✓	35.1%	38.4%	73.2%
	α-FL	✓	✓	36.0%	38.2%	72.4%
	CB	✓	✓	**38.0%**	**39.7%**	**73.9%**

Appendix E: BuildingGNN ablation study

We conducted an ablation study involving different node features, and also experimenting with different types of edges in our BuildingGNN. Table 12 present the results for different experimental conditions of our BuildingGNN based on PointNet++ as node features. We first experimented using no edges and processing node features alone through our MLP structure. We experimented with using only OBB-based features ("Node-OBB"), using features from PointNet++ alone ("Node-PointNet++"), and finally using both node features concatenated ("Node-OBB+PointNet++"). We observe that using all combinations of node features yields better performance compared to using either node feature type alone. Then we started experimented with adding each type of edges individually to our network (e.g., "w/ support edges" in Table 12 means that we use node features with support edges only). Adding each type of edge individually further boosts performance compared to using node features alone. Using all edges ("BuildingGNN-PointNet") yields a noticeable 7.1% Part IoU increase and 8.1% Shape IoU increase compared to using node features alone. Table 13 shows the same experiments using MinkowskiNet-based features. We observe that combined node features perform better than using either node feature type alone. Adding each type of edges helps, except for proximity edges that seem to have no improvement when used alone. Using all edges still yields a noticeable 2.6% Part IoU increase and 6.2% Shape IoU increase compared to using node features alone.

Table 12: BuildingGNN ablation study based on PointNet++ node features

Variant	n?	c?	Part IoU	Shape IoU	Class acc.
Node-OBB	✓	✓	10.0%	17.1%	56.5%
Node-PointNet++	✓	✓	14.0%	19.1%	52.2%
Node-OBB+PointNet++	✓	✓	24.4%	27.8%	71.7%
w/ support edges	✓	✓	26.7%	29.2%	71.5%
w/ containment edges	✓	✓	27.9%	30.6%	72.6%
w/ proximity edges	✓	✓	26.4%	29.4%	71.4%
w/ similarity edges	✓	✓	23.1%	28.5%	69.8%
BuildingGNN-PointNet++	✓	✓	**31.5%**	**35.9%**	**73.9%**

Table 13: BuildingGNN ablation study based on MinkowskiNet node features

Variant	n?	c?	Part IoU	Shape IoU	Class acc.
Node-OBB	✓	✓	10.0%	17.1%	56.5%
Node-MinkNet	✓	✓	35.6%	35.9%	67.7%
Node-OBB+MinkNet	✓	✓	40.0%	40.6%	75.8%
w/ support edges	✓	✓	42.0%	43.5%	77.8%
w/ containment edges	✓	✓	41.1%	42.0%	76.8%
w/ proximity edges	✓	✓	39.9%	40.6%	75.6%
w/ similarity edges	✓	✓	41.2%	43.0%	75.8%
BuildingGNN-MinkNet	✓	✓	**42.6%**	**46.8%**	**77.8%**

We also experimented with DGCNN [6] as a backbone in our GNN for extracting node features. Unfortunately, DGCNN could not directly handle our large points clouds (100K points). It runs out of memory even with batch size 1 on a 48GB GPU card. We tried to downsample the point clouds (10K points) to pass them to DGCNN, then propagated the

Table 14: Part IOU performance for each label. BuildingGNN-MinkNet and BuildingGNN-PointNet++ are tested on the mesh track, while MinkNet and PointNet++ are tested on the point cloud track. The left half of the table reports performance when color is available ("n+c"), while the right half reports performance when it is not available ("n").

Label	BuildingGNN MinkNet(n+c)	BuildingGNN PointNet++(n+c)	MinkNet (n+c)	PointNet++ (n+c)	BuildingGNN MinkNet(n)	BuildingGNN PointNet++(n)	MinkNet (n)	PointNet++ (n)
Window	70.5%	71.1%	44.1%	34.8%	70.4%	68.3%	35.6%	0.0%
Plant	81.0%	69.8%	79.6%	70.3%	79.8%	69.8%	79.7%	48.4%
Vehicle	83.7%	77.3%	77.1%	29.7%	82.7%	72.4%	75.8%	19.2%
Wall	78.1%	77.5%	64.5%	57.9%	76.0%	74.4%	63.2%	54.4%
Banister	50.0%	19.9%	44.9%	0.0%	56.5%	22.0%	45.6%	0.0%
Furniture	59.7%	37.0%	56.0%	0.0%	58.3%	43.5%	54.9%	0.0%
Fence	55.5%	34.7%	71.3%	16.5%	64.1%	19.7%	49.5%	9.6%
Roof	78.9%	72.1%	65.3%	58.2%	70.2%	69.0%	67.0%	56.4%
Door	41.7%	37.6%	21.7%	0.0%	39.2%	37.7%	23.8%	0.0%
Tower	53.4%	41.2%	46.5%	2.3%	50.8%	37.5%	43.4%	4.8%
Column	61.5%	27.6%	49.5%	0.6%	53.6%	34.7%	42.9%	1.1%
Beam	24.9%	22.4%	13.8%	0.02%	30.3%	21.5%	17.2%	0.0%
Stairs	38.6%	25.6%	26.9%	0.0%	41.0%	24.1%	27.8%	0.0%
Shutters	1.0%	1.3%	0.0%	0.0%	1.7%	0.0%	0.0%	0.0%
Garage	9.0%	10.6%	3.6%	0.0%	10.6%	8.4%	6.8%	0.0%
Parapet	24.9%	3.9%	11.6%	0.0%	28.6%	2.5%	21.0%	0.0%
Gate	14.0%	16.5%	6.4%	0.0%	7.9%	12.3%	7.9%	0.0%
Dome	53.8%	10.1%	48.0%	1.9%	54.3%	14.2%	54.5%	16.3%
Floor	51.5%	37.7%	47.8%	36.9%	51.2%	30.9%	46.8%	30.0%
Ground	75.0%	65.1%	77.4%	64.1%	61.8%	55.5%	60.8%	42.6%
Buttress	23.8%	9.6%	15.6%	0.0%	38.7%	12.3%	6.1%	0.0%
Balcony	19.6%	9.5%	15.0%	0.0%	15.5%	15.6%	17.3%	0.0%
Chimney	70.0%	50.9%	57.9%	0.0%	53.6%	49.5%	60.1%	0.0%
Lighting	6.4%	9.1%	16.8%	0.0%	24.9%	3.5%	23.3%	0.0%
Corridor	16.3%	10.5%	15.9%	4.2%	7.2%	4.1%	7.2%	0.0%
Ceiling	28.0%	23.8%	22.1%	4.6%	28.0%	20.5%	17.4%	4.6%
Pool	70.8%	53.0%	78.7%	77.8%	38.1%	33.0%	43.0%	0.0%
Dormer	27.3%	20.4%	9.6%	0.0%	22.1%	23.3%	6.8%	0.0%
Road	46.2%	24.1%	53.5%	40.0%	1.9%	16.3%	21.5%	0.0%
Arch	8.4%	5.2%	0.9%	0.0%	3.2%	2.9%	0.8%	0.0%
Awning	1.5%	0%	3.8%	0.0%	1.6%	0.0%	0.0%	0.0%

node features back to the 100K points using nearest neighbor upsampling. The part IoU was 32.5% in the mesh track with color input and using all edges (i.e., the performance is comparable to BuildingGNN-PointNet++, but much lower than BuildingGNN-MinkNet). Still, since other methods were able to handle the original resolution without downsampling, this comparison is not necessarily fair, thus we excluded it from the tables showing the track results in our main paper.

Appendix F: Additional Material

As additional supplementary material of our paper, we include part of our video tutorial that demonstrates our UI functionality (see MP4 video in the supplementary zip file). Finally, we refer readers to our project page www.buildingnet.org for the dataset and source code.

References

[1] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pat. Ana. & Mach. Int.*, 23(11), 2001. 3

[2] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-Balanced Loss Based on Effective Number of Samples. In *Proc. CVPR*, 2019. 4

[3] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3D Mesh Segmentation and Labeling. *ACM Trans. on Graphics*, 29(3), 2010. 3

[4] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *Proc. ICCV*, 2017. 4

[5] Trimble. *3D Warehouse*, 2020. 1

[6] 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 Trans. on Graphics*, 38(5), 2019. 5

[7] Wikipedia. *List of building types*, 2018. 1