Floorplan Restoration by Structure Hallucinating Transformer Cascades Supplementary material

BMVC 2023 Submission # 90

The supplementary document provides details of the datasets (Appendix A), the full specifications of our neural architecture (Appendix B), discussion on computational complexity (subsection B.1), details of our loss functions (subsection B.2), additional details on refinement process (subsection B.3), results under different IOUs (Figure 1), and more quantitative and qualitative results (Table 5 and Figures 2 and 3).

A Datasets details

SUN360 is a popular indoor panorama dataset [7]. For floorplans, RPLAN provides sixty thousand synthetic samples in a vector-graphics format [6]. LIFULL HOME's dataset provides five million real samples in a raster format [4]. For both panorama images and floorplans, Structured3D provides 3,500 synthetic houses/apartments [8]. For real samples, Matterport3D provides 90 houses [2], while their coverage is extremely dense.

ZIND dataset is the closest to ours with 1,500 real houses/apartments, where the panorama coverage is sparse, and human interventions were required for camera pose estimation and floorplan reconstruction [3]. Nonetheless, ensuring a panorama(s) is taken in every room (except stairs and corridors) makes the setup small-scale and highly controlled. Our data comes from uncontrolled crowd-sourcing, where many rooms are not photographed, posing fundamental challenges to existing techniques.

In our proposed dataset, the number of panoramas per house ranges from 1 to 7. Specifically, (31, 129, 222, 199, 118, 2) houses contain (1, 2, 3, 4, 5, 7) panoramas, respectively. The test set contains (9, 19, 15, 7) houses containing (2, 3, 4, 5) panorama images, respectively. The number of invisible rooms varies between 1 and 17. The number of invisible doors varies between 0 and 11.

As it has been stated in main paper, in order to solve class imbalance problem we use a weighing constant ($w(i)$) that is inversely proportional to the number of samples in the training set, which is (1.135, 3.7, 0.6, 0.63, 0.47, 1.2, 2.2, 0.76, 0.48, 0.18, 0.763, 0.524, 0.33, 0.7, and 0.1) for (living room, kitchen, bedroom, toilet, balcony, corridor, tatami, washroom. bathroom, closet, closet door, open door, door, entrance, and no-room), respectively. Table 1 present more detail on dataset statistic.

To evaluate the robustness across different datasets, we also used RPLAN [6] datsaset which is a public dataset. We use Housegan++ [5] data parser to parse RPLAN, which results in 60K houses; the number of rooms per house varies between 5 to 8. We divide

Table 1: Dataset statistics. We divide the set into five groups based on the number of rooms (2-5, 6-9, 10-13, 14-17, 18+). In each group, the table reports (Top) the number of samples; (Middle) the ave/std of the number of three types of rooms; and (Bottom) the ave/std of the number of two types of doors.

# of Rooms		2-5	6-9	10-13	14-17	18+
# of Samples		60	243	298	84	16
Room	Visible	2.4/0.9	3.5/1.2	3.7/1.6	3.9/1.5	3.4/1.0
	Invis. direct	1.6/0.5	3.7/0.9	5.2/1.6	6.4/2.2	7.2/2.4
	Invis. indire.	0.7/0.9	1.4/1.4	3.7/2.2	5.0/2.8	8.2/3.2
Door	Visible	3.0/1.2	7.2/1.6	8.4/2.5	10.6/4.0	11.4/2.3
	Invisible	0.8/1.0	1.5/1.7	4.4/3.3	5.9/3.2	9.1/3.5

the dataset into 55K for training and 5K for testing. Using our dataset's statistics, we flag the number of rooms per house as invisible. The number of visible rooms per house varies between 2 to 7, the number of invisible rooms varies between 1 to 6, and the number of invisible doors varies between 0 to 5.

B Network Architecture

Category Wise CNN takes an input image of the resolution $800 \times 800 \times 14$. For the house with X number of visible rooms, We will have X number of C-wise CNN for rooms category were the output will be $X \times 256 \times 25 \times 25$, then we apply max function which result output of that to be $256 \times 25 \times 25$. Same will be applied to door and room (both) category. For doors only category there is no need for max, as we give all doors at same time to one C-wise CNN block. Then we concatenate them resulting output of all to be 3×256, then we reshape them to get input sequence for encoder to be 1875×256, let's refer it as *src*.

Table 2: Category Wise CNN (C-Wise CNN) block

Layer name/type	Specification	Output size
Conv1	7×7, 64, stride 2	$64 \times 400 \times 400$
Conv block2	3×3, maxpool, stride 2 $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$256 \times 200 \times 200$
Conv block3	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$512 \times 100 \times 100$
Conv block4	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$1024 \times 50 \times 50$
Conv block5	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$2048 \times 25 \times 25$
Conv6	1×1, 256	$256 \times 25 \times 25$

Encoder has 6 layers (See Table 3), it receives *src*, position, and type embedding as input. As

Table 3: One layer of encoder, we have 6 layers

Layer name/type	input	Specification	Output size
Multi head attention	src	embed_dim=256, num_heads=8	1875×256 (src2)
Dropout	src2	dropout=0.1	1875×256 (src3)
LayerNorm	src3+src	normalized_shape=256	1875×256 (src4)
Linear1, ReLU	src4	in_features=256, out_features=2048	1875×2048 (src5)
Dropout	src5	dropout=0.1	1875×2048 (src6)
Linear2	src6	in_features=2048, out_features=256	1875×256(src7)
Dropout	src7	dropout=0.1	1875×256 (src8)
LayerNorm	src8+src4	normalized_shape=256	1875×256 (src9)

positional embedding shape is 625×256, we concatenate it by itself three times to produce 1875×256 positional embedding. Type embedding shape is 3×256, we concatenate each embedding related to each category by itself 625 times and concatenate all them together to produce 1875×256 type embedding. Encoder input is summation of *src* and those two embeddings.

Final output of encoder will be 1875×256. We only use the ones for room-door (both) category as decoder input which will be 625×256, lets refer this as *memory (Mem)*.

Decoder has three cascaded blocks and 6 layers in each block. Table 4 shows one layer in one block. Let us assume that we have Y dangling doors. then number of queries for first decoder will be Y, second will be Y+15 and third will be Y+30. Lets show each cascade input queries as *tgt*. The first cascade input is Y dummy queries and encoder output (*Mem*). Then first cascade output will be concatenate with 15 dummy queries the result is input queries for second decoder cascade, second cascade also receives *Mem* as input. For last cascade, second cascade output will be concatenate with 15 dummy queries and same as other two cascades, third cascade also receives *Mem* too.

Table 4: One layer of decoder, we have 6 layers, Y is number of input queries, in first cascade it is equal to number of dangling doors, in second it is number of dangling doors+15, and in third number of dangling doors+30

Layer name/type	input	Specification	Output size
Multi head attention	tgt	embed_dim=256, num_heads=8	$Y \times 256$ (tgt2)
Dropout	tgt2	dropout=0.1	$Y \times 256$ (tgt3)
LayerNorm	tgt3+tgt	normalized_shape=256	$Y \times 256$ (tgt4)
Multi head attention	Mem, tgt	embed_dim=256, num_heads=8	$Y \times 256$ (tgt5)
Dropout	tgt6	dropout=0.1	$Y \times 256$ (tgt7)
LayerNorm	tgt7+tgt4	normalized_shape=256	$Y \times 256$ (tgt8)
Linear1, ReLU	tgt8	in_features=256, out_features=2048	$Y \times 2048$ (tgt9)
Dropout	tgt9	dropout=0.1	$Y \times 2048$ (tgt10)
Linear2	tgt10	in_features=2048, out_features=256	$Y \times 256$(tgt11)
Dropout	tgt11	dropout=0.1	$Y \times 256$ (tgt12)
LayerNorm	tgt8+tgt12	normalized_shape=256	$Y \times 256$ (tgt13)

B.1 Computational complexity

The number of network parameters is 50 million ResNet 50. As a reference, the network size of DETR [3] resp. 40 million. Note that our network size is not very different from other

standard Transformer based architecture. The reason for choosing a batch size of 1 is not due to memory constraints, but to accommodate the varying number of visible doors/rooms per instance during implementation.

B.2 Loss functions

Without loss of generality, we use the second cascade as an example to define loss functions, where reconstructed invisible room instances are matched against the corresponding ground-truth. The loss functions consist of the three terms for the room-type classification, the bounding box parameter regression, and the segmentation mask estimation:

$$L = L_{type} + L_{bbox} + L_{seg}, \tag{1}$$

$$L_{type} = \frac{1}{|\mathcal{I}_{all}|} \sum_{i \in \mathcal{I}_{all}} -w(i) log(p_i), \tag{2}$$

$$L_{bbox} = \frac{1}{|\mathcal{I}_{match}|} \sum_{i \in \mathcal{I}_{match}} 5 \left\| b_i - \hat{b}_i \right\|_1 - 2 \, \mathrm{IOU}(b_i, \hat{b}_i), \tag{3}$$

$$L_{seg} = \frac{5}{|\mathcal{I}_{match}|} \sum_{i \in \mathcal{I}_{match}} \left[L_{dice}(m_i, \hat{m_i}) + \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} L_{focal}(m_i^p, \hat{m_i^p}) \right]. \tag{4}$$

L_{type} is a standard softmax loss with per-category weighing. $\mathcal{I}_{all}$ is a set of indexes of all the reconstructed room instances. With abuse of notation, p_i denotes the classification score of the GT room type (i.e., the room type of the matched GT). $w(i)$ denotes a weight associated with the GT room type to compensate for the imbalance in the training data. Concretely, the weight is set inversely proportional to the number of samples in the training set.

L_{bbox} sums the discrepancies of the bounding box estimation over the matched instances. The first term is the L1 norm of the 4-dimensional bounding box parameter vector (the center, the width, and the height in the normalized image coordinate). b_i denotes the estimated parameter vector and $\hat{b}_i$ denotes the corresponding GT vector. The second term is the intersection over the union score between the reconstructed and the ground-truth bounding boxes.

L_{seg} sums the discrepancy of the segmentation mask and the matched instance. The first term is the standard dice loss between the estimated mask (m_i) and the GT ($\hat{m_i}$). The second term is the average focal loss of the per-pixel mask value. $\mathcal{P}$ denotes the set of pixels in the domain. m_i^p (resp. $\hat{m_i^p}$) denotes the estimated (resp. GT) per-pixel mask value.

B.3 Final floorplan refinement

Here we give a few more details on our refinement process. RPLAN dataset [6] was initially used to train House-GAN++ but is synthetic and may exhibit incompatible database bias. We use LIFULL HOME'S dataset [1] instead. Since the goal is the refinement without overall arrangement changes, we use fully-connected graphs for training and testing. The reconstructed floorplan from our cascading decoder is specified as the input constraint. In the first iteration, we give our last cascade output as inputs mask to Housegan++. The predicted mask of the previous iteration will be input for each node from the second iteration, and we continue this to the 10th iteration. By doing this, as we are limiting Housegan++, masks only get refined and curvy edges will change to manhattan shapes.

C Additional Experimental Results

C.1 LPIPS and FID Results

As discussed in the main paper, FID and LPIPS are not as powerful metrics as much as our main metric to show if our method is successful in our unique task however we show them here for the reference.

Table 5: FID and LPIPS results. We compare against MAT, Mask-RCNN (M_{rcnn}), House-GAN++, and DETR with ResNet-50 backbone. Input partial floorplans are ground-truth.

Method	↓ FID	↓ LPIPS
MAT	80.9	0.245
M_{rcnn}	76.0	0.241
House-GAN++	45.6	0.220
DETR	79.1	0.243
Ours	76.8	0.242
Our*	42.4	0.214

C.2 Additional results on different IOU-thresholds

In Figure 1 we are providing recall and precision graph for different IOUs starting from 0 going up to 1, increasing by 0.1. Both the precision and the recall decrease as we increase the IOU threshold. As our plots are based on the IOU threshold on the metric computation, by decreasing the IOU threshold, number of True positive increases in both precision and recall which cause the increase in both of them, in all IOU thresholds our method has the highest performance.

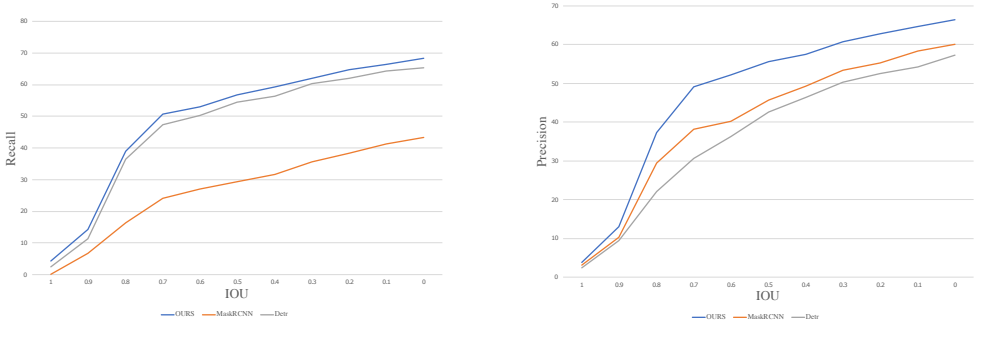

Figure 1: Both the precision and the recall decrease as we increase the IOU threshold in Fig5, which is not what a normal precision/recall curve shows, for example in a detection task. This is because, our plots are based on the IOU threshold on the metric computation, as opposed to a confidence threshold of a detection in a normal precision/recall curve.

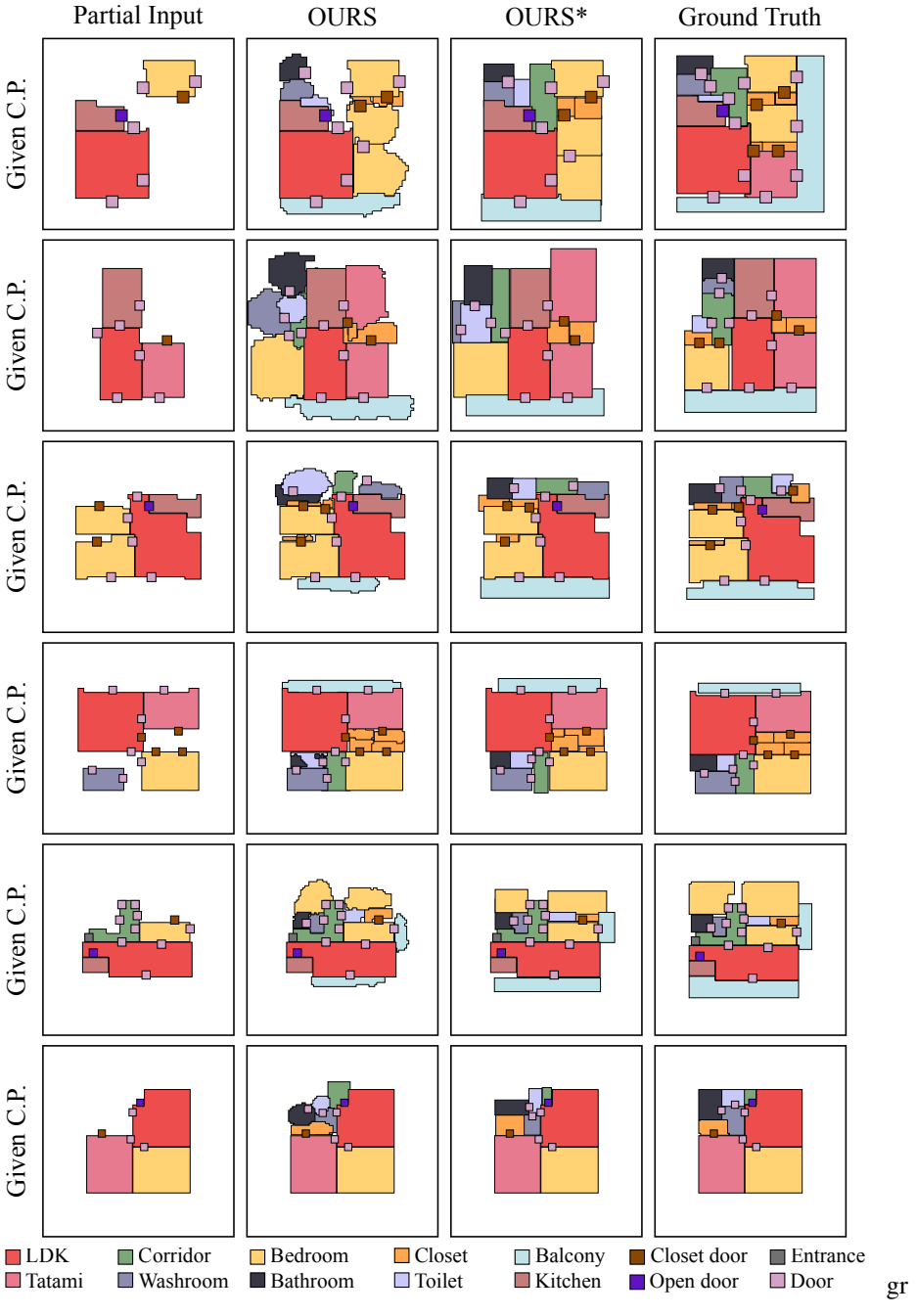

Figure 2: More qualitative results. Reconstructed floorplans before and after the refinement for our method by House-GAN++ and several heuristics for more samples. The input partial reconstructions are derived either by using given Camera Pose in this figure.

C.3 More qualitative results

Figure 2 shows reconstructed floorplans by our method before and after applying House-GAN++. and 3 present additional output layout samples by our system and other baselines before and after applying refinements. In first row we don't use any refinement, while second row per sample present first and third rows output after applying refinements and House-GAN++.

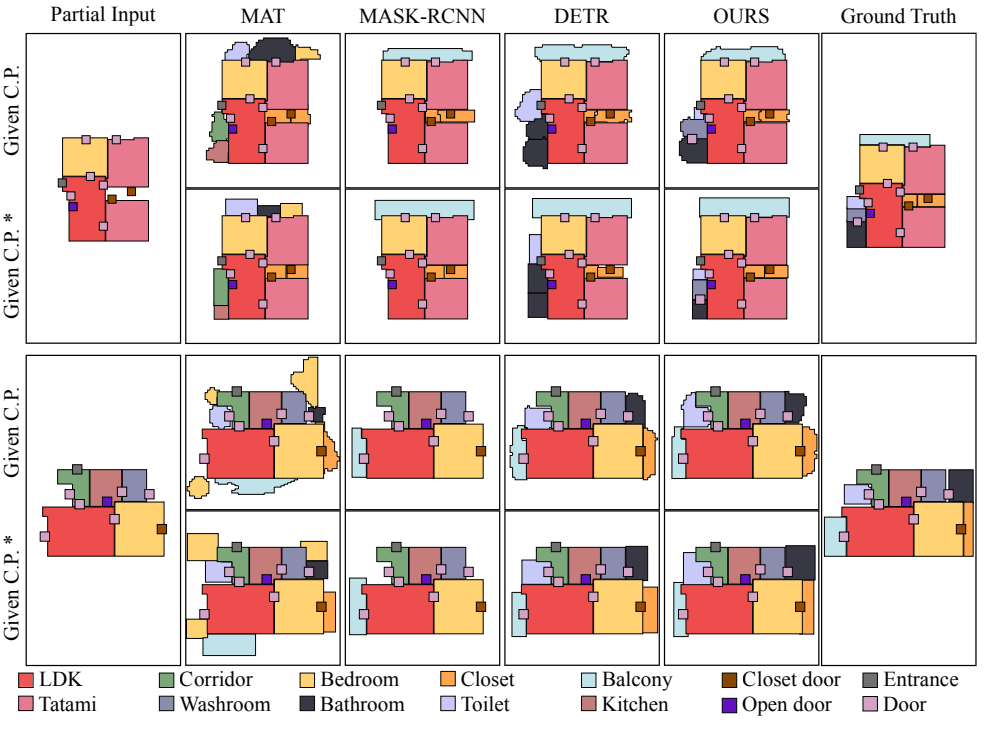

Figure 3: More qualitative results. Reconstructed floorplans before and after the refinement for different baselines and ours by House-GAN++ and several heuristics for more samples. The same refinement process is used for all the methods. The input partial reconstructions are derived either by using given Camera Pose in this figure.

References

[1] Liful dataset. https://www.nii.ac.jp/dsc/idr/lifull.

[2] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[3] Steve Cruz, Will Hutchcroft, Yuguang Li, Naji Khosravan, Ivaylo Boyadzhiev, and Sing Bing Kang. Zillow indoor dataset: Annotated floor plans with 360deg panora-

mas and 3d room layouts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2133–2143, June 2021.

[4] Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2214–2222, 2017. doi: 10.1109/ICCV.2017.241.

[5] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13632–13641, 2021.

[6] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 38(6), 2019.

[7] Jianxiong Xiao, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Recognizing scene viewpoint using panoramic place representation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2695–2702. IEEE, 2012.

[8] Jia Zheng, Junfei Zhang, Jing Li, Rui Tang, Shenghua Gao, and Zihan Zhou. Structured3d: A large photo-realistic dataset for structured 3d modeling. In *Proceedings of The European Conference on Computer Vision (ECCV)*, 2020.