Unified Vector Floorplan Generation via Markup Representation

Supplementary Material

A. More Implementation Details

Hyper-parameters. We give the additional information about the hyper-parameters of our model in Table 8. *Temperature* and *Top P* are the hyper-parameters used to determine the predicted class from the probability distribution with randomness, seen frequently in the context of LLMs. We introduce such randomness when the variation is required, as shown in Table 9.

Hyperparameter	Value
Transformer Dimension	512
MLP Dimension	2048
Num Heads	32
Num Layers	24
Temperature	0.6
Top P	0.8

Table 8. **Hyper-parameters.**

Token Type	Uncond.	Boundary	Number	Graph	G&B	Compl.
Tag						
Coordinate	✓	✓	✓	✓	✓	✓
Room index	✓	✓				✓
Room type	✓	✓	✓			✓

Table 9. **Randomness.**

To give the randomness to specific token types, we determine which token type should appear next by the following process: 1) If the token type can be uniquely determined by the grammar of FML, we adopt it. 2) If not, we compute the sum of probabilities of each token type and we adopt the token type whose sum is the largest. Once the token type determined, we sample the class only from the type.

Dataset size. In Table 10, we give the numbers of train/test samples in our experiments in the main paper.

Sec. 4.2	Table 2	Table 3	Table 4	Table 5	Table 6	Table 7
65763 / 100	74995 / 2880	65763 / 378	59232 / 9895	74995 / 2880	74995 / 2880	74995 / 400

Table 10. **Dataset size.**

Constrains. We list the constrains in decoding in Table 11.

1. An interior and front door should have just two vertices.
2. Room vertices should be placed outside the previously generated rooms.
3. Interior door vertices should be placed on a edge between two different rooms.
4. Front door vertices should be placed on a edge between a room and the outside region.
5. A room should have four or more vertices.

Table 11. **Constrains used in decoding.**

Pre-processing. We follow the pre-processing code provided by HouseGAN++. Also, since FML requires that every two rooms supposed to be adjacent must share an edge for a door, we inflated the room polygons so that they have the shared edge. We also re-computed the adjacency graph to filter out incorrect annotations after inflation.

B. User Study

For further evaluation, we conduct a user study on Amazon Mechanical Turk. We show 20 users the generated floorplans and ask them to "select the most functional and natural floorplan". We uniformly pick 100 sets of generated results from our model, HouseGAN++, and HouseDiffusion on the graph conditional generation task. Table 12 shows the winning rate, *i.e.*, the percentage of cases each method was selected as the most functional and natural floorplan by the users. Note that ties can occur when multiple methods receive the same number of votes, so the total does not necessarily sum to 100%. We can see that our method is much more preferable than the previous methods, indicating our superior functionality and naturalness.

Ours	HouseGAN++	HouseDiffusion
51% (51/100)	24% (24/100)	32% (32/100)

Table 12. **Winning rate on the user study.**

C. More Experiments

Effect of multi-task learning. In Table 13, we train additional variants by dropping out some learning tasks and evaluate them on the graph-conditional task. The result gives us an interesting finding: the task-specific variant (a) achieves the best GED, and the variant trained on uncond.&graph (b) significantly worsens GED; however, the more tasks we add, the better GED we obtain, as observed in (c) and (d).

Evaluation Task	Setting	Learning Tasks				GED (↓)
		Uncond.	Boundary	Graph	B & G	
Graph	(a)			✓		0.99
	(b)	✓		✓		1.41
	(c)	✓	✓	✓		1.34
	(d)	✓	✓	✓	✓	1.21

Table 13. **Effect of multi-task learning.**

Computation cost. In Table 14, we compare our model with HouseGAN++ and HouseDiffusion in terms of training time and per-sample inference time for each room number (*i.e.*, 5, 6, 7, and 8) on a single NVIDIA A100 GPU. We compute the inference time by averaging 100 samples for each room number. Our method is trained much faster

Method	Training Time	Inference Time (5/6/7/8)
HouseGAN++	12h	0.12s / 0.13s / 0.14s / 0.16s
HouseDiffusion	106h	10.0s / 10.0s / 10.0s / 10.1s
Ours	27h	3.2s / 4.0s / 4.7s / 5.6s

Table 14. **Computation cost on a single NVIDIA A100 GPU.**

than the previous state-of-the-art HouseDiffusion method. For inference time, we observed that 1) in HouseDiffusion, the iterative denoising process is more dominant than the number of rooms and 2) our inference time scales linearly with the number of rooms.

Evaluation on doors. To assess the alignment quality of doors, we compute GED only on doors by setting the room editing cost to 0. We use the same generated floorplans as Table 4. The results of HouseGAN++, HouseDiffusion, and ours are 1.98, 1.47, and **1.15**, respectively, showing that our model produces better-aligned doors.

Quantitative gain by constrained decoding. We evaluate our model without constrained decoding on graph conditions using the same test set as Table 4. We obtain a GED of 1.64 for the variant, where our full model achieves 1.21 while HouseDiffusion achieves 1.55. The result indicates that constrained decoding is important to achieve better GED. This is mainly because, in FML which represents doors as lines, an adjacency is not until that a door completely overlaps on a shared edge between two rooms.

D. More Related Work

In addition to the related work referred to in Sec. 2, we further mention additional studies on floorplan generation.

Completion. A prior study [12] tackles floorplan generation from panorama images of rooms. During training, the model is trained to reconstruct entire floorplans from incomplete floorplans. During inference, input panorama images are processed using SfM [25] and converted into partial floorplans. After that, partial floorplans are complemented by the trained model.

Room arrangement. Our model also can be applied to the room arrangement task formulated as spatial puzzle solving as introduced in PuzzleFusion [13]. In this case, each room should be given in a relative coordinate in condition tokens rather than absolute coordinates as boundary conditions.

Text-driven synthesis. Tell2Design [19] generates floorplans from text descriptions with an encoder-decoder architecture. The core difference between FML and Tell2Design is that Tell2Design conditions floorplans mainly by "natural language", which sacrifices strictness in exchange for flexibility. For example, it would suffer from its ambiguity when distinguishing between two rooms that belong to the same room type. FML takes advantage of eliminating such ambiguity by imposing the strict markup-based grammar as defined in Sec. 3.1.

E. Failure Cases

We provide typical failure cases in Fig. 8. It can be observed that 1) our method generates rooms that are not perfectly aligned with boundary conditions and 2) our method places two rooms, that are supposed adjacent, but in distant positions.

F. More Generated Examples

We show the generated floorplans on number-conditional generation in Fig. 9 and additional generated examples on the other conditional generation tasks in Fig. 10.

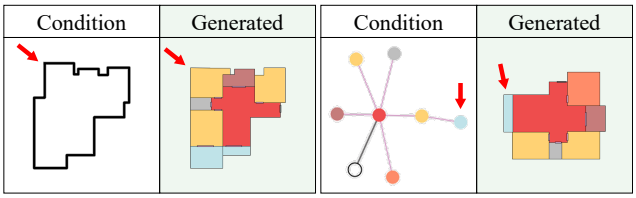

Figure 8. **Failure cases.**

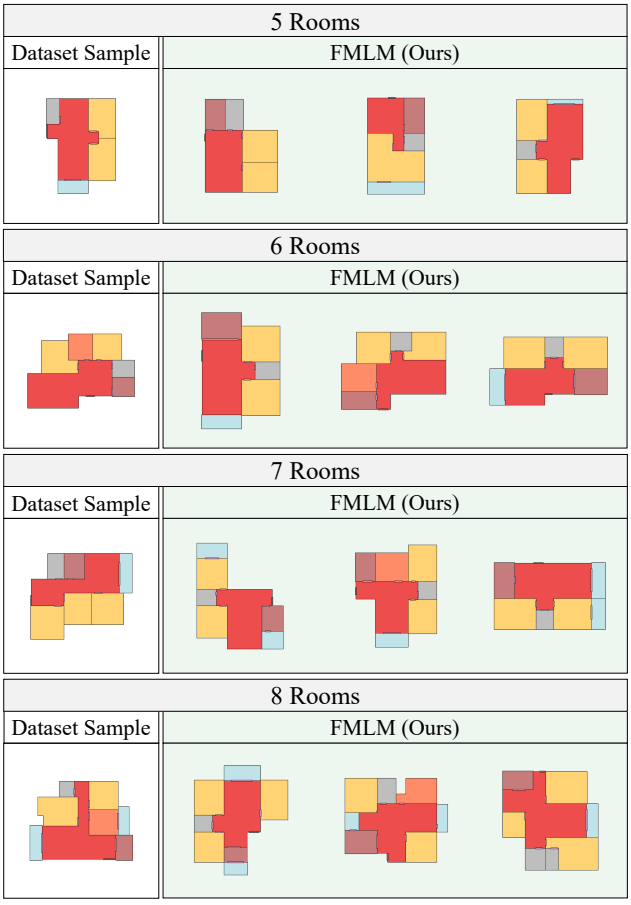

Figure 9. **Number-conditional generation.**

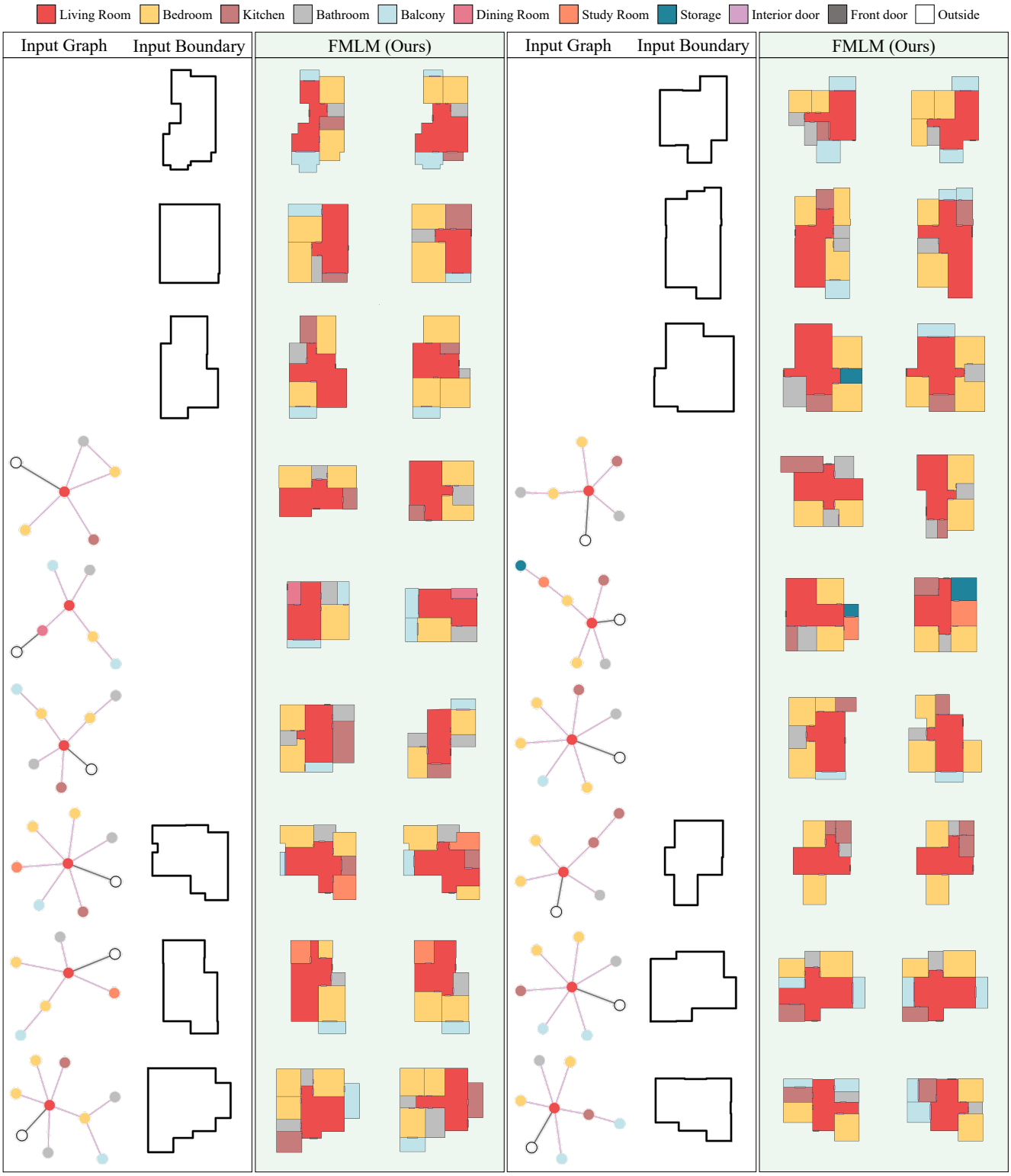

Figure 10. **Additional generated examples.**