This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Unified Vector Floorplan Generation via Markup Representation

Kaede Shiohara Toshihiko Yamasaki
The University of Tokyo
{shiohara, yamasaki}@cvm.t.u-tokyo.ac.jp

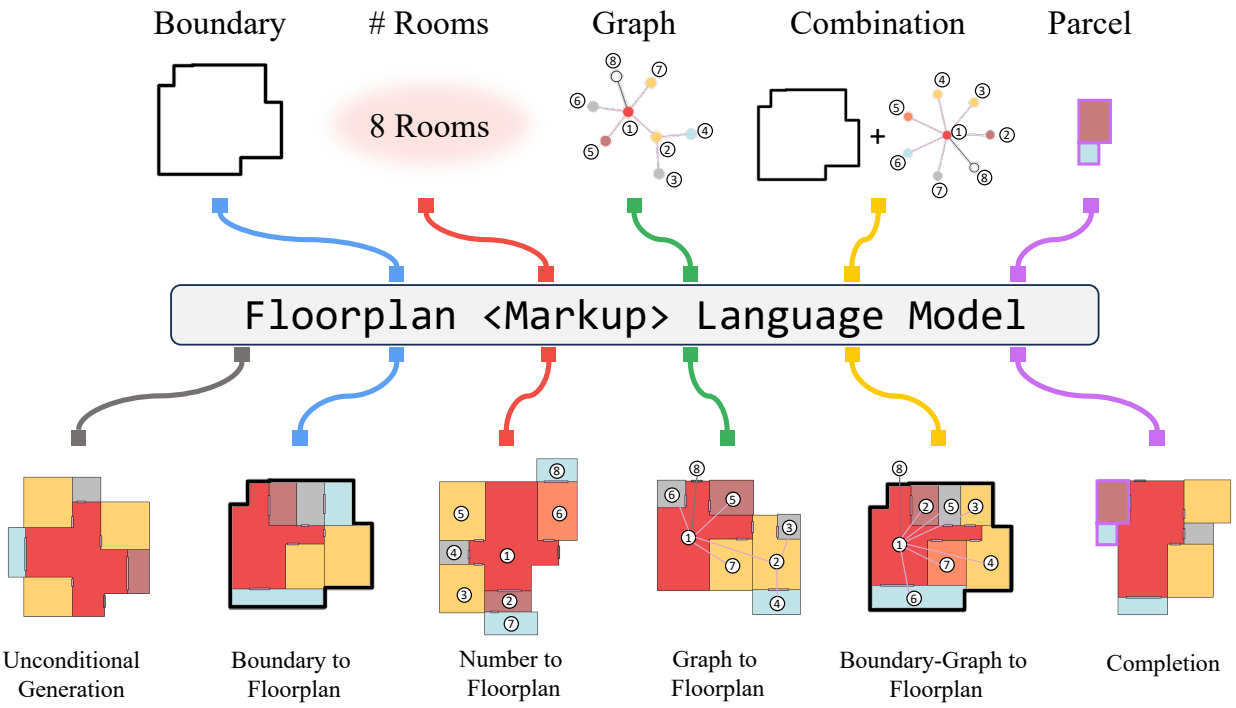


Figure 1. Our Floorplan Markup Language Model directly generates vector floorplans under the wide range of situations.

Abstract

Automatic residential floorplan generation has long been a central challenge bridging architecture and computer graphics, aiming to make spatial design more efficient and accessible. While early methods based on constraint satisfaction or combinatorial optimization ensure feasibility, they lack diversity and flexibility. Recent generative models achieve promising results but struggle to generalize across heterogeneous conditional tasks, such as generation from site boundaries, room adjacency graphs, or partial layouts, due to their suboptimal representations. To address this gap, we introduce Floorplan Markup Language (FML), a general representation that encodes floorplan information within a single structured grammar, which casts the entire floorplan generation problem into a next token prediction task. Leveraging FML, we develop a transformer-based generative model, FMLM, capable of producing high-fidelity and functional floorplans under diverse conditions. Comprehensive experiments on the RPLAN dataset demonstrate that FMLM, despite being a single model, surpasses the previous task-specific state-of-the-art methods.

1. Introduction

Automatically generating residential floorplans has long been a goal at the intersection of architecture and computer graphics. It promises to lower design costs, accelerate early-stage exploration, and empower non-experts to test multiple layout options quickly. A key challenge is to generate layouts that satisfy practical design constraints while remaining plausible. This typically means honoring

user-specified inputs such as site boundaries, room types, or adjacency graphs while producing functionally valid and architecturally realistic layouts.

Early systems [18, 20, 33] formulated floorplan generation as constraint satisfaction or combinatorial search, arranging rooms under adjacency, area, and circulation rules. While these approaches provide hard feasibility guarantees, they require extensive expert-crafted heuristics and offer limited stylistic diversity and suffer from computational overheads for each single floorplan generation. With the rapid progress of generative models [5, 9, 17], data-driven approaches have emerged. Some methods generate rasterized segmentation masks of room types [22, 23] while others operate directly in vector space to synthesize polygonal layouts [11, 15, 26] empowered by diffusion models [9]. However, despite these efforts, previous approaches suffer from poor generalizability across different types of conditional generation tasks such as boundary, number of rooms, room adjacency graph, completion from partial layouts, and their combinations. This is because previous methods rely on suboptimal representations that are less compatible with the structural information of floorplan data, which significantly makes the previous approaches inefficient and redundant by requiring conversion from raster to vector floorplans [22, 23] and multi-stage pipelines [11, 14, 15].

To address this issue, we propose a general representation called Floorplan Markup Language (FML). Inspired by HyperText Markup Language (HTML), FML represents a floorplan and its conditions such as a boundary and graph as a single sequence of tagged elements, where the tag structure explicitly constrains which elements can appear next. The structured, tag-based design of FML provides a clear and regular syntax, which naturally constrains the generation process and guides the model toward valid and coherent layouts. Leveraging this representation, our autoregressive transformer model trained to generate FML sequences, which is called the Floorplan Markup Language Model (FMLM), can produce high-fidelity floorplans across a wide variety of tasks, as illustrated in Fig. 1.

We compare our model with the previous state-of-the-art methods such as Graph2Plan [14], HouseGAN++ [23], HouseDiffusion [26], and GSDiff [15] on the RPLAN dataset [34]. Extensive experiments show that our unified model outperforms the previous task-specific models in a wide range of tasks in terms of FID [8], IoU, and GED [1].

2. Related Work

2.1. Floorplan Generation

Optimization-based approaches. Early work tried to generate floorplans by iterative optimization on hand-crafted rules and user-specified constraints such as property boundaries, size of rooms, and adjacencies between rooms. Liu *et al*. [20] proposed an interactive pipeline with an active-set optimizer that minimizes the costs for area fidelity, aspect-ratio regularization, boundary utilization, and a fabrication term that aligns wall lengths to a catalog of precast slab widths. IP-Layout [33] introduced a hierarchical, coarse-to-fine sub-domain refinement that scales to large layouts, *e.g*., offices, malls, and supermarkets, solved with a mixed integer quadratic programming solver. Optimizer [18] integrated genetic optimization into constraint programming, which improves the functionality and architectural fidelity such as room shape alignment and circulation efficiency.

However, these methods suffer from the trade-off between fidelity and diversity; more rules decrease the diversity of floorplans while less rules degrade the fidelity.

Data-driven approaches. More recent work focuses on data-driven approaches with generative models [5, 9, 17]. HouseGAN [22] and HouseGAN++ [23] adopt convolutional message passing neural networks [35] to generate raster floorplans from graph conditions. Graph2Plan [14] proposed a coarse-to-fine pipeline that first predicts coarse bounding boxes of rooms and then refines them by simultaneously generated floorplan images. FloorplanGAN [21] introduced a differentiable renderer that rasterizes floorplans images from their vector representations to apply a raster-based GAN framework on vector floorplans. HouseDiffusion [26] proposed a diffusion-based approach that diffuses the positions of room vertices conditioned on room adjacency graphs. Cons2Plan [11] introduces a two-stage model in which the first stage produces graphs from site boundaries, and the subsequent stage generates room vertices in a similar fashion to HouseDiffusion. GSDiff [15] supports variable-length room polygons by introducing a multi-stage pipeline that includes room vertex generation, edge prediction, and room type assignment.

However, these methods suffer from poor generalization ability across different generation tasks due to their suboptimal representations and network designs. For example, the recent diffusion-based methods [11, 15, 26] mainly focus on the diffusion of the coordinates of room vertices, which forces the system either to pre-condition the graph structure in advance specifying the number of rooms and the num-

Method	Uncond.	Conditional				Compl.
		Boundary	Number	Graph	B & G	
HouseGAN*				✓		
HouseGAN++				✓		
Graph2Plan		✓			✓	
WallPlan		✓		✓	✓	
HouseDiffusion				✓		
Cons2Plan		✓		✓	✓	
GSDiff*	✓	✓		✓		
FMLM (Ours)	✓	✓	✓	✓	✓	✓

Table 1. **Supported tasks.** Our model handles various types of floorplan generation tasks while previous methods perform only specific tasks. * indicates that the models could not generate doors.

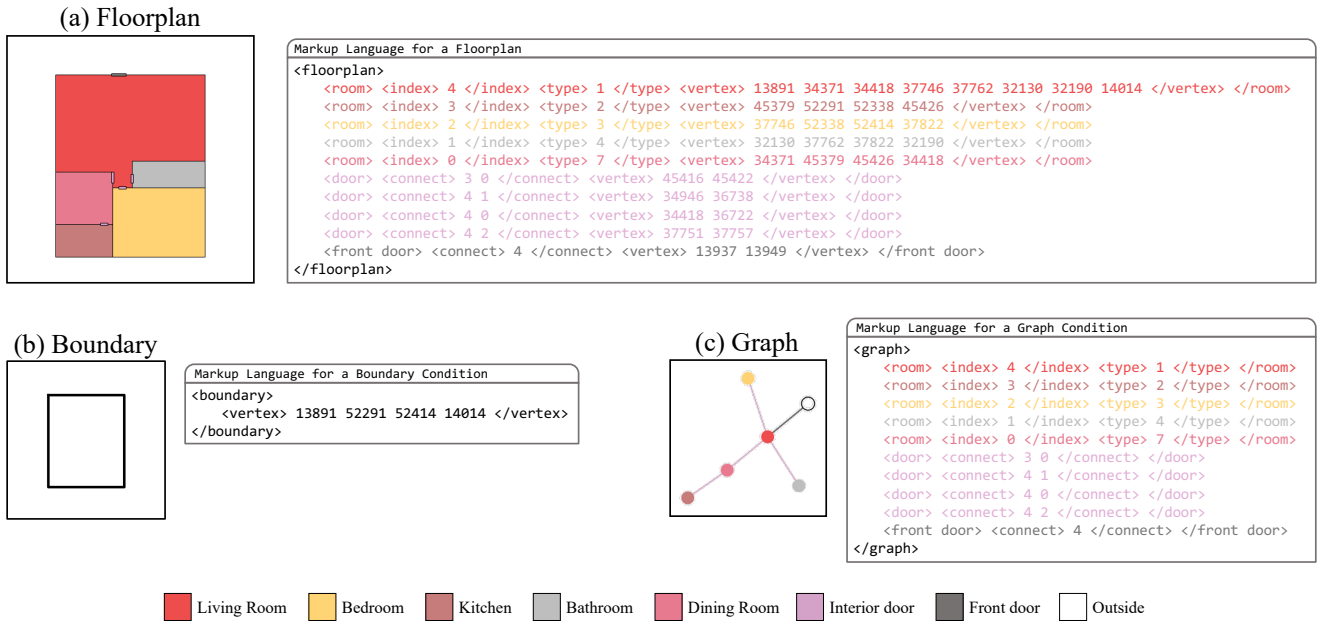


Figure 2. **Floorplan Markup Language (FML).** We represent floorplans, boundaries, and graphs in a markup manner, which unifies the various floorplan generation tasks into a single task of FML sequence generation.

ber of vertices each room should have [11, 26], or to require additional networks to infer the underlying structures including edge extraction and room type assignment from the generated potential vertices [15]. Such task-specific network designs make it difficult for the models to generalize to multiple generation tasks as shown in Table 1. In contrast, we represent floorplans and conditions such as boundaries and graphs in single sequences, enabling our model to operate in a single stage and generalize to a wide range of floorplan generation tasks.

2.2. Autoregressive Modeling

Autoregressive (AR) models generate data sequentially by conditioning each output element on previously generated ones. Early approaches based on recurrent neural networks (RNNs) such as long short-term memory (LSTM) [10] modeled sequential data such as handwriting synthesis [7] and machine translation [27]. The Transformer [31] further established AR modeling as a general-purpose paradigm for high-dimensional generative modeling, leading to powerful large language models [3, 6, 31]. In the computer vision and graphics, autoregressive frameworks have been successfully applied to pixel-level [30], patch-level [4], and scale-level [29] image generation, video generation [28], and geometry reconstruction [32].

In this work, we base our model on autoregression so that it can generate variable numbers of rooms, doors, and their vertices, taking advantages against non-AR models that generate only fixed numbers of them [11, 26] or require additional steps for room allocation [15].

3. Proposed Method

Our goal is to generate diverse and plausible vector floorplans under various situations such as unconditional generation, conditional generation on the site boundary, the number of rooms, and the room adjacency graph, and completion of partial floorplans. To achieve this, we introduce a new representation called Floorplan Markup Language (FML) that represents the entire structure information of a floorplan as a single sentence written in a markup language (Sec. 3.1). With FML, any floorplan generation tasks can be formulated as a sequential generation task of FML. Therefore, our introduced simple autoregressive transformer trained to generate FML sequences, which we named Floorplan Markup Language Model (FMLM), performs unified vector floorplan generation without raster-based representation [22], discretization from continuous representation [26], and multi-staging [15] (Sec. 3.2).

3.1. Floorplan Markup Language

We present Floorplan Markup Language (FML) that represents floorplans and conditions such as boundary and graph in a tag-structured manner as shown in Fig. 2.

Preliminary. A floorplan with N_r rooms and N_d interior doors is represented in 2D space as a set of room polygons with associated room type labels $\{(P_i, t_i)\}_{i=1}^{N_r}$, interior door lines $\{D_j\}_{j=1}^{N_d}$, and a front door line F in 2D space. Each room polygon $P_i \in \mathbb{R}^{n_i \times 2}$ has a room-specific number of vertices n_i and each interior or front door is represented as a line connecting an initial point and an ending point,

i.e., $D_j, F \in \mathbb{R}^{2\times 2}$. The interior doors are placed on the edges between rooms, while the front door is placed on the edge between a room and the outside region. A boundary condition is represented as a polygon $B \in \mathbb{R}^{n_b \times 2}$ with the number of vertices n_b. A graph condition is represented as an adjacency matrix $G \in \{0,1\}^{N_r \times N_r}$ where each element $G_{i,j} = 1$ indicates that rooms i and j are connected via an interior door, and $G_{i,j} = 0$ otherwise.

Grammar. In FML, we define four types of tokens including tag, coordinate, room index, and room type. We define the grammar of FML as follows:

Rule 1: FML starts with `<sequence>` and ends with `</sequence>`.

Rule 2: Between `<sequence>` and `</sequence>`, the tag `<floorplan>` is used to start to describe floorplan information and the tag `</floorplan>` is used to end it.

Rule 3: Between `<floorplan>` and `</floorplan>`, the tags `<room>`, `<door>`, and `<front door>` are used to start to describe the information about each room, door, and front door, respectively. FML always describes them in the order of rooms $\rightarrow$ doors $\rightarrow$ a front door. Also, we use `</room>`, `</door>`, and `</front door>` at the end of the respective descriptions.

Rule 4: Between `<room>` and `</room>`, FML first places `<index>` and `</index>`. A single number between them represents the room index in descending order.

Rule 5: After `</index>`, FML places `<type>` and `</type>`. A single number between them represents the room type (*e.g.*, living room, kitchen, etc.).

Rule 6: After `</type>`, FML places `<vertex>` and `</vertex>`. Multiple numbers between them represent the vertex positions of room polygons. Each 2D coordinate (x, y) is converted into a 1D value $z = x + y * W$ where W denotes the width of the 2D space.

Rule 7: Between `<door>` and `</door>`, FML first places `<connect>` and `</connect>`. Two numbers between them represent the room indices that the door connects.

Rule 8: After `</connect>`, FML places `<vertex>` and `</vertex>`. Two numbers between them represent the beginning and ending coordinates of the door. We use the same coordinate space as room polygons.

Rule 9: Front doors are described in the same manner as doors by `<front door>` and `</front door>`, except that only a single number is placed between `<connect>` and `</connect>` because a front door connects a single room to the outside.

In addition to the basic rules above to represent floorplans, we define the following rules to support conditional generation tasks such as boundary and graph as follows:

Rule 10: Before the floorplan description, FML can place tags `<condition>` and `</condition>` that correspond to the beginning and ending of condition information, respectively. FML can use `<boundary>` and `<graph>` to start to describe the information about boundary and graph condition, respectively. FML always describes in the order of boundary $\rightarrow$ graph. Also, we use `</boundary>` and `</graph>` at the end of the respective descriptions.

Rule 11: Between `<boundary>` and `</boundary>`, FML describes a boundary condition by the set of boundary vertices in the same manner as room vertices.

Rule 12: Between `<graph>` and `</graph>`, FML represents a room adjacency graph by the same manner as floorplans but without vertices information.

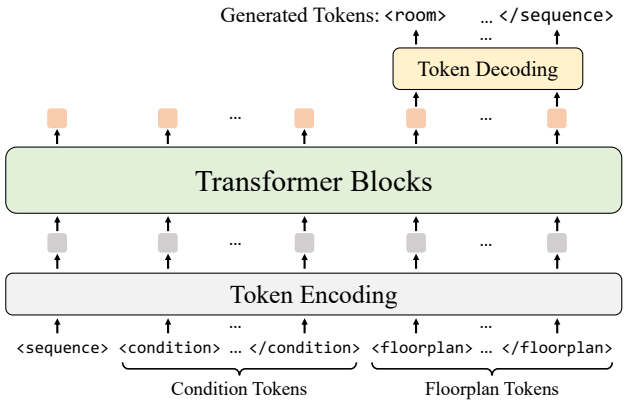


Figure 3. **Overview of FMLM.** Similarly to LLMs, FMLM is based on a simple transformer model that is trained by next token prediction and performs autoregressive inference.

3.2. Floorplan Markup Language Model

We introduce Floorplan Markup Language Model (FMLM) to generate FML sequences. The overview of the architecture is illustrated in Fig. 3. Our model is based on an autoregressive transformer that takes a token sequence and predicts its next token. The transformer block is the same architecture as LLaMA-3 [6] which consists of self-attention [31], layer normalization [2], and multi-layer perceptron. We encode and decode tokens as follows:

Encoding. We encode tags, room indices, and room types by assigning a learnable vector to each class. For coordinates, we encode 2D values into a single vector by sinusoidal positional embedding and apply a learnable linear projection to it.

Decoding. To decode the markup language from output tokens, we apply a learnable linear projection head $W \in \mathbb{R}^{(C_{\text{tag}}+C_{\text{coord}}+C_{\text{index}}+C_{\text{type}})\times C}$ where C_{tag} is the number of tag classes, $C_{\text{tag}} = H * W$ with the height H and width W of the 2D coordinate space, C_{index} is the maximum number of rooms per floorplan in the dataset, C_{type} is the number of room types, and C is the dimension size of the output embedding. Let $\boldsymbol{x} = \{x_i\}_{i=1}^{L}$ be an input FML sequence with a length of L and f be the transformer that takes $\{x_i\}_{i=1}^{l}$ and outputs embedding $f(\{x_i\}_{i=1}^{l}) \in \mathbb{R}^C$ in the teacher-

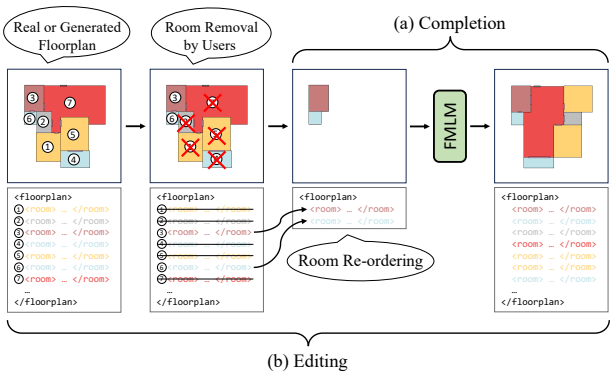


Figure 4. **Floorplan completion and editing.** (a) Our model complements incomplete floorplans by just starting with an incomplete sequence. (b) With this capability, our model can be incorporated into interactive editing with users.

forcing manner to predict the next token x_{l+1}. The embedding $f(\{x_i\}_{i=1}^{l})$ is cast into the class probability with the head W and the softmax operation:

$$p(x_{l+1} \mid \{x_i\}_{i=1}^{l}) = \text{softmax}(W f(\{x_i\}_{i=1}^{l})). \quad (1)$$

3.3. Training

We train our model with the cross-entropy loss:

$$\mathcal{L} = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}} \left[-\sum_{l=l_1}^{L-1} \log p(x_{l+1} \mid \{x_i\}_{i=1}^{l}) \right], \quad (2)$$

where $\mathcal{D}$ is the dataset and l_1 denotes the token index of `<floorplan>` in $\boldsymbol{x}$; we apply the loss only on the floorplan tokens as shown in Fig. 3.

Unlike recent diffusion-based models [11, 26], our model is originally permutation-sensitive for room orders, which is incompatible with the permutation-equivalent nature of floorplans. Therefore, we introduce a simple yet effective data augmentation technique for FMLM, called room permutation, that randomizes the order of rooms in floorplans to learn permutation-equivalency.

3.4. Constrained Decoding

To generate consistent floorplans during inference, we force our model to strictly follow the grammar of FML by setting the probabilities of improper classes to 0. For example:

1. Doors should have just two vertices.
2. Room vertices should be placed outside the previously generated rooms.

The full list is found in appendix. Enabling integration of such heuristic rules into the generator is one of the advantages of autoregressive modeling compared to the previous non-AR generative approaches [15, 23, 26].

Condition	Method	FID (↓)	IoU (↑)
Boundary	Graph2Plan	34.20	95.87
	FMLM (Ours)	**6.51**	**97.86**

Table 2. **Comparison with Graph2Plan on boundary condition.**

Condition	Method	FID (↓)	IoU (↑)
Boundary	GSDiff	11.26	97.21
	FMLM (Ours)	**4.61**	**98.06**

Table 3. **Comparison with GSDiff on boundary condition.**

3.5. Unconditional Generation

Once inputting tags `<sequence> <floorplan>` into our model, it autoregressively generates subsequent tokens without any conditions.

3.6. Conditional Generation

Number conditions. Our model naturally performs conditional generation by the number of rooms using the `<index>` tag. Concretely, when we aim to generate a room that has five rooms, we start autoregressive generation with "`<sequence> <floorplan> <room> <index>` 4".

Boundary conditions. As shown in Fig. 2(b), we input a sequence of vertex coordinates of the boundary.

Graph conditions. As shown in Fig. 2(c), we represent a graph condition that constrains the number and types of rooms. To encourage our model to learn the correlation between graph conditions and target floorplans, we sort the rooms in each ground truth floorplan to match the order in the corresponding graph condition.

3.7. Completion and Editing.

Our model is also capable of floorplan completion and editing as follows:

Completion. As shown in Fig. 4(a), by starting with an incomplete floorplan sequence, our model complements the sequence to generate its full floorplan.

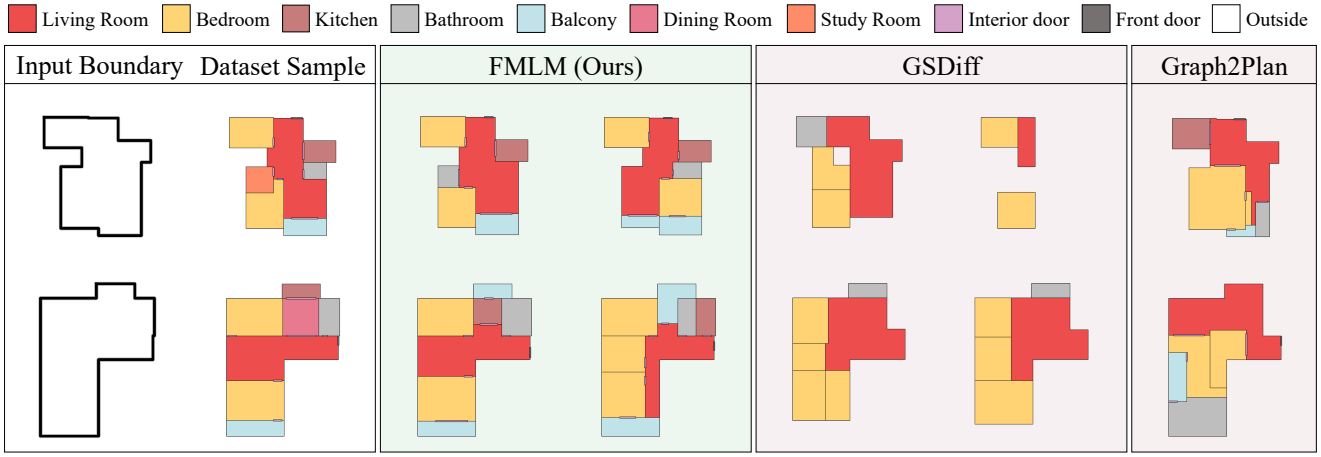
Editing. Floorplan editing is performed by combining a removal process with completion as shown in Fig. 4(b).

4. Experiments

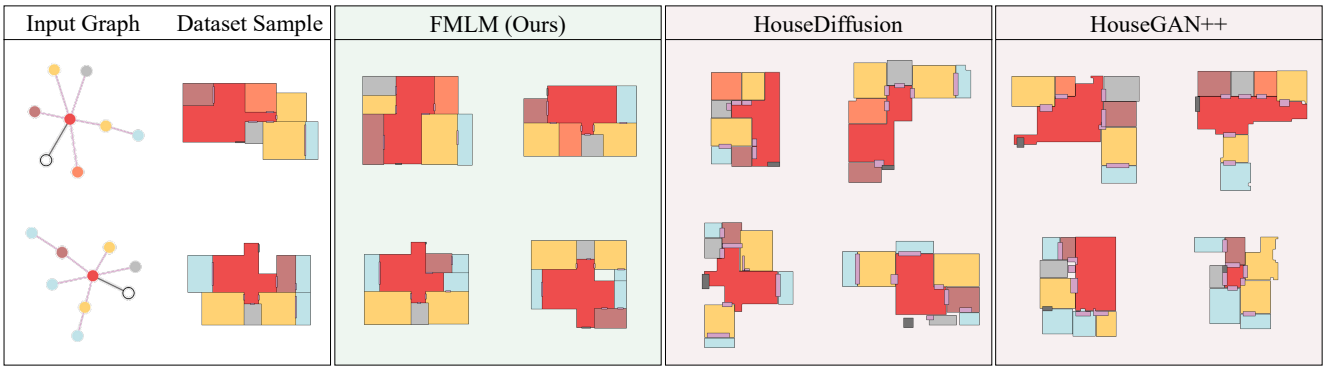
4.1. Setup

Dataset. We used RPLAN dataset [34] that comprises 80k floorplans with dense annotations. Following previous work [23, 26], our model is trained to generate nine room classes *i.e.*, *Living Room*, *Bedroom*, *Bathroom*, *Dining Room*, *Kitchen*, *Study Room*, *Entrance*, *Storage*, and *Balcony*, and two door classes, *i.e.*, *Interior Door* and *Front Door*.

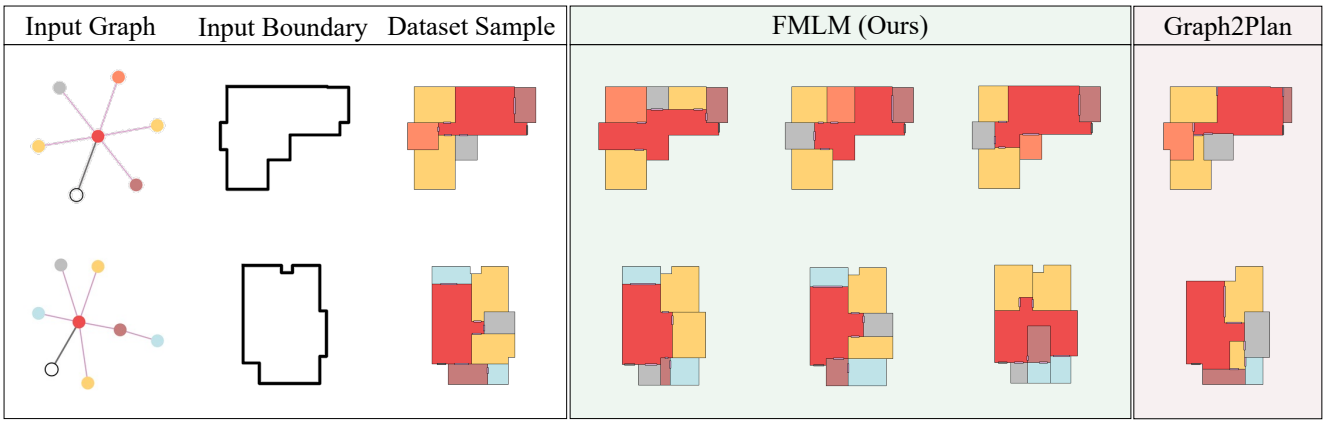
Metrics. Commonly on all the floorplan generation tasks, we use Frechet Inception Distance (FID) [8] to evaluate the



(a) Boundary-conditional generation.



(b) Graph-conditional generation



(c) Multi-conditional generation

Figure 5. **Qualitative comparison with Graph2Plan and HouseDiffusion.** Our model generates more consistent and realistic examples from various types of conditions, (a) boundary, (b) graph, (c) boundary and graph, than task-specific models such as Graph2Plan and HouseDiffusion. Note that Graph2Plan generates only a single floorplan for each condition. Best viewed in zoom.

distribution distance between real floorplan images and generated ones. On boundary conditional generation tasks, we introduce Intersection over Union (IoU) to evaluate the distance between internal regions of boundaries and union of generated room regions. On graph conditional generation tasks, we adopt Graph Edit Distance (GED) [1] to evaluate the distance between condition graphs and reconstructed ones from generated floorplans.

Baselines. We compare our model with the previous approaches including HouseGAN++ [23], HouseDiffu-

Condition	Method	FID (↓)					GED (↓)				
		5	6	7	8	ALL	5	6	7	8	ALL
Graph	HouseGAN++	44.63	46.88	49.62	54.49	48.44	1.56	2.04	2.51	3.25	2.57
	HouseDiffusion	32.98	27.15	31.61	30.43	29.31	0.97	1.34	1.52	**1.87**	1.55
	FMLM (Ours)	**6.97**	**5.20**	**4.07**	**4.64**	**3.41**	**0.49**	**0.69**	**1.13**	1.96	**1.21**

Table 4. **Comparison with HouseDiffusion on graph condition.**

Condition	Method	FID (↓)					GED (↓)					IoU (↑)				
		5	6	7	8	ALL	5	6	7	8	ALL	5	6	7	8	ALL
B & G	Graph2Plan	44.32	28.84	29.53	36.07	22.87	2.30	2.95	3.55	4.26	3.43	93.72	93.56	92.69	92.29	92.96
	FMLM (Ours)	**29.20**	**14.88**	**21.85**	**30.64**	**14.17**	**0.45**	**0.71**	**1.30**	**2.01**	**1.24**	**98.42**	**98.31**	**97.52**	**96.64**	**97.59**

Table 5. **Comparison with Graph2Plan on boundary-graph condition.**

sion [26], Graph2Plan [14], and GSDiff [15].

Implementation details. We implement our model with PyTorch [24]. All the experiments are conducted on a single NVIDIA A100 GPU. C_{tag}, C_{index}, C_{type}, H, and W are set to 15, 8, 9, 256, and 256, respectively. Note that we exclude `<sequence>`, `<condition>`, `</condition>`, `<boundary>`, `</boundary>`, `<graph>`, `</graph>`, and `<floorplan>` from the tag classes predicted by the model because they never appear as targets in Eq. 2. We train our model from scratch for 50 epochs with Adam [16] optimizer. The batch size and learning rate are set to 32 and 1.0^{-4}, respectively. The boundary and graph conditions are dropped out by a chance of 50%, respectively. Therefore, our model is trained without conditions, with only the boundary condition, with only the graph condition, and with both boundary and graph conditions, each with a probability of 25%. More details are included in appendix.

4.2. Unconditional Generation

We compare our model with GSDiff [15] on unconditional generation. Since GSDiff could not generate doors, we remove doors in our predicted floorplans and ground truth ones during computing FID for a fair comparison. We observe that our method achieves **7.22** while GSDiff achieves 15.02, indicating that our method outperforms GSDiff.

4.3. Conditional Generation

Boundary. We evaluate our model on the boundary conditional generation task in comparison to Graph2Plan [14] and GSDiff [15]. We give the numerical comparisons with Graph2Plan in Table 2 and with GSDiff in Table 3. Note that because the official pre-trained models of Graph2Plan and GSDiff adopted different train/test splits, we train our model on each training set and evaluate them on each test set for fair comparison. We show the generated examples in Fig. 5a. We can see that our model is more consistent with the boundary conditions than Graph2Plan which often lacks interior doors to connect rooms. Also, our method can generate more plausible floorplans than GSDiff that lacks layout fidelity and diversity. Our method outperforms them in both FID and IoU, indicating our model generates higher-fidelity floorplans from boundary conditions.

Graph. We then evaluate our model on the graph conditional generation task in comparison to HouseGAN++ [23] and HouseDiffusion [26]. To prevent models from simply outputting the same examples as in training data, we first divided floorplans into five groups depending on the number of rooms, *i.e.*, five, six, seven, and eight rooms. Then, for each group, we divided the samples into training and testing sets so that identical graphs do not overlap between training and testing. We train our model, HouseGAN++ [23], and HouseDiffusion [26] on the combined training samples of all groups. Note that we could not compare GSDiff [15] as it does not generate doors and, therefore, the definition of room adjacency of GSDiff is different from that of our model, HouseGAN++, and HouseDiffusion.

We show the generated examples in Fig. 5b. We can see that all the methods succeed in generating plausible floorplans from graph conditions. However, HouseGAN++ and HouseDiffusion sometimes place interior doors in locations that are not the boundaries between rooms, which does not occur in our method because of constrained decoding. We also give the quantitative comparison in Table 4. Similarly with Table 3, we remove doors when we compute FID for a fair comparison because there is a difference in door representations, where HouseGAN++ and HouseDiffusion represent doors by polygons while our method represents them by lines. Our model outperforms HouseGAN++ and HouseDiffusion in all the metrics except GED on eight rooms where our model slightly underperforms HouseDiffusion. This may be because the number of samples with eight rooms in the training set is much fewer than the others, which can be improved by some techniques such as weighted sampling during training.

Boundary and Graph. We here evaluate our model conditioned by both boundary and graph. We used the same checkpoints of our model and Graph2Plan as in boundary-conditional generation. Because Graph2Plan requires the

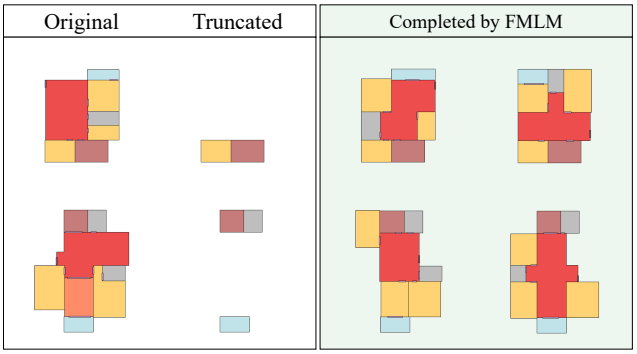


Figure 6. **Floorplan completion.**

Condition	Setting	FID (↓)	GED (↓)	IoU (↑)
B & G	w/o Permutation	24.36	2.35	95.82
	Ours	**14.17**	**1.24**	**97.59**

Table 6. **Effect of room permutation.**

Condition	Setting	FID (↓)				
		5	6	7	8	ALL
Number	Ascending Order	105.78	112.29	114.80	125.13	94.57
	Ours	**47.17**	**46.89**	**49.38**	**48.82**	**25.50**

Table 7. **Effect of indexing in descending order.**

front door positions in advance while our method does not, we exclude the effect of front doors when we evaluate GED. The generated examples of our model and Graph2Plan can be found in Fig. 5c. Our model generates high-fidelity floorplans from graph and boundary conditions while Graph2Plan tends to generate unnatural layouts and fails to generate expected room types. We also give the quantitative comparison in Table 5, where our approach outperforms Graph2Plan in all the metrics including FID, GED, and IoU for all the numbers of rooms.

Number. Due to space limitations, we show the generated examples conditioned by the number of rooms in appendix.

4.4. Completion

Our model is also capable of the completion of partial floorplans. As shown in Fig. 6, our model generates diverse samples from the same part of rooms. This result indicates that our method empowers users to interactively edit floorplans.

4.5. Ablation Studies

Effect of room permutation. We compare our full model and its variant without room permutation in Table 6. The augmentation improves all the metrics, demonstrating the importance of learning permutation-equivalency.

Effect of indexing in descending order. Another key design of FML is to describe room indices in descending order. To demonstrate this, we train a variant where room indices are described in ascending order and compare it with

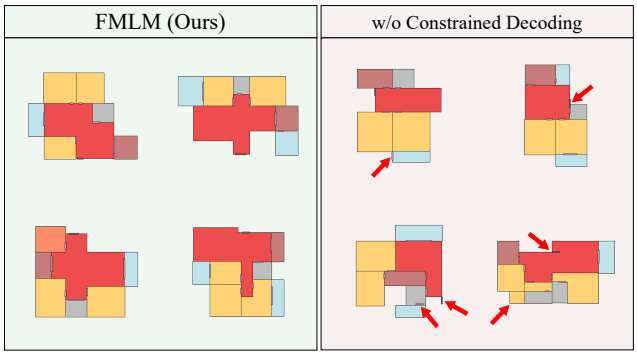


Figure 7. **Effect of constrained decoding.** Best viewed in zoom.

our model on number-conditional generation in Table 7. For the variant, we stop generating rooms once the number of the generated rooms reaches the conditioned number. The variant significantly drops FID; this is because, in ascending order setting, the model could not determine in advance how many rooms should be generated.

Effect of constrained decoding. We also show the effect of constrained decoding in Fig. 7 by comparing the cases with and without it during inference. We generate rooms without any conditions by two models where one of them is our full model and the other is our model without constrained decoding. We can see that without it, floorplans are often generated with overlapping rooms and misaligned doors that appear outside the edges between rooms.

5. Limitations and Future Work

Our approach still has some limitations that point to promising directions for future research. First, the integration of FML into LLMs presents an opportunity to enable floorplan generation directly from natural-language descriptions. This direction could substantially broaden the accessibility and controllability of layout synthesis. Second, our current model is restricted to single-story floorplans. Extending FMLM to handle multi-story designs is a natural next step, which may be achieved by introducing additional structural tags such as `<story>` and `</story>` to represent vertical hierarchy and inter-floor relationships.

6. Conclusion

In this paper, we present Floorplan Markup Language (FML) for unified vector floorplan generation. FML describes floorplans and constraints such as site boundaries and room adjacency graphs in a markup manner, which unifies various floorplan generation tasks into a FML sequence generation task. Extensive experiments show that our autoregressive transformer model trained with FML outperforms the previous methods designed for specific tasks, demonstrating the strong generalization ability of our representation across different generation tasks.

Acknowledgments

This work was partially financially supported by JST ASPIRE Program, Japan, Grant Number JPMJAP2303.

References

[1] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *ICPRAM*, 2015. 2, 6

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv:1607.06450*, 2016. 4

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 2020. 3

[4] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021. 3

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014. 2

[6] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024. 3, 4

[7] Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013. 3

[8] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 2017. 2, 5

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 2

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997. 3

[11] Shibo Hong, Xuhong Zhang, Tianyu Du, Sheng Cheng, Xun Wang, and Jianwei Yin. Cons2plan: Vector floorplan generation from various conditions via a learning framework based on conditional diffusion models. In *MM*, 2024. 2, 3, 5

[12] Sepidehsadat Hosseini and Yasutaka Furukawa. Floorplan restoration by structure hallucinating transformer cascades. In *BMVC*, 2023. 2

[13] Sepidehsadat Hosseini, Mohammad Amin Shabani, Saghar Irandoust, and Yasutaka Furukawa. Puzzlefusion: Unleashing the power of diffusion models for spatial puzzle solving. In *NeurIPS*, 2023. 2

[14] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. Graph2plan: Learning floorplan generation from layout graphs. *TOG*, 2020. 2, 7

[15] Sizhe Hu, Wenming Wu, Yuntao Wang, Benzhu Xu, and Liping Zheng. Gsdiff: Synthesizing vector floorplans via geometry-enhanced structural graph generation. In *AAAI*, 2025. 2, 3, 5, 7

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 7

[17] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013. 2

[18] Graziella Laignel, Nicolas Pozin, Xavier Geffrier, Loukas Delevaux, Florian Brun, and Bastien Dolla. Floor plan generation through a mixed constraint programming-genetic optimization approach. *Automation in Construction*, 2021. 2

[19] Sicong Leng, Yang Zhou, Mohammed Haroon Dupty, Wee Sun Lee, Sam Joyce, and Wei Lu. Tell2design: A dataset for language-guided floor plan generation. In *ACL*, 2023. 2

[20] Han Liu, Yong-Liang Yang, Sawsan Alhalawani, and Niloy J. Mitra. Constraint-aware interior layout exploration for pre-cast concrete-based buildings. *Vis. Comput.*, 2013. 2

[21] Ziniu Luo and Weixin Huang. Floorplangan: Vector residential floorplan adversarial generation. *Automation in Construction*, 2022. 2

[22] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *ECCV*, 2020. 2, 3

[23] 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 *CVPR*, 2021. 2, 5, 6, 7

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019. 7

[25] Mohammad Amin Shabani, Weilian Song, Makoto Odamaki, Hirochika Fujiki, and Yasutaka Furukawa. Extreme structure from motion for indoor panoramas without visual overlaps. In *ICCV*, 2021. 2

[26] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *CVPR*, 2023. 2, 3, 5, 7

[27] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *NeurIPS*, 2014. 3

[28] Hansi Teng, Hongyu Jia, Lei Sun, Lingzhi Li, Maolin Li, Mingqiu Tang, Shuai Han, Tianning Zhang, WQ Zhang, Weifeng Luo, et al. Magi-1: Autoregressive video generation at scale. *arXiv:2505.13211*, 2025. 3

[29] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *NeurIPS*, 2024. 3

[30] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *NeurIPS*, 2016. 3

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 3, 4

[32] Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A Efros, and Angjoo Kanazawa. Continuous 3d perception model with persistent state. In *CVPR*, 2025. 3

[33] Wenming Wu, Lubin Fan, Ligang Liu, and Peter Wonka. MIQP-based Layout Design for Building Interiors. *Computer Graphics Forum*, 2018. 2

[34] Wenming Wu, Lubin Fan, Ligang Liu, and Peter Wonka. Miqp-based layout design for building interiors. *TOG*, 2019. 2, 5

[35] Fuyang Zhang, Nelson Nauata, and Yasutaka Furukawa. Conv-mpn: Convolutional message passing neural network for structured outdoor architecture reconstruction. In *CVPR*, 2020. 2