FloorplanSBS: Synthesizing Vector Floorplans by Patch-Based Floorplan Segmentation

Wenming Wu
Hefei University of Technology
Hefei, China
wwming@hfut.edu.cn

Tianlei Sheng
Hefei University of Technology
Hefei, China
2023110572@mail.hfut.edu.cn

Gaofeng Zhang
Hefei University of Technology
Hefei, China
g.zhang@hfut.edu.cn

Liping Zheng*
Hefei University of Technology
Hefei, China
zhenglp@hfut.edu.cn

Abstract

Automated vector floorplan generation is valuable for designers to explore potential spatial designs. However, existing learning-based methods rely on complex post-processing or optimization to obtain plausible vector floorplans, which disrupts the end-to-end design flow. In this paper, we propose *FloorplanSBS*, a patch-based segmentation framework for directly synthesizing vector floorplans. Our method leverages the strengths of box-based representation and segmentation-based generation, following a division-and-labeling scheme. The framework operates in two stages: given input design constraints, a division model first divides the design space into rectangular patches, followed by a labelling model that assigns semantic labels to each patch. *FloorplanSBS* supports constraints such as boundaries and layout graphs. Extensive evaluations show that it surpasses state-of-the-art methods in generating high-quality vector floorplans. With its end-to-end neural framework, *FloorplanSBS* eliminates the need for post-processing, offering a simple, efficient, and user-friendly tool for vector floorplan design.

CCS Concepts

• Applied computing → Computer-aided design; **• Computing methodologies → Shape modeling**.

Keywords

Floorplan Generation; Vector Floorplan; Neural Network

ACM Reference Format:
Wenming Wu, Tianlei Sheng, Gaofeng Zhang, and Liping Zheng. 2025. FloorplanSBS: Synthesizing Vector Floorplans by Patch-Based Floorplan Segmentation. In *Proceedings of the 33rd ACM International Conference on Multimedia (MM '25), October 27–31, 2025, Dublin, Ireland*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3746027.3754556

1 Introduction

Architectural floorplans are typically created using design software such as AutoCAD and are represented in vector graphics, which are widely used in industrial design. However, manually drawing floorplans requires precise sketching of floorplan structures and defining room types and dimensions, which is time-consuming and labor-intensive. Automated floorplan generation techniques have garnered significant attention [15, 17, 28], helping designers quickly explore potential design solutions. In recent years, deep learning techniques [12, 25, 29] have led to significant advances in the automatic generation of vector floorplans.

Image-based methods [19, 25, 29] directly predict floorplan images instead of vector floorplans. However, pixel-level segmentation often introduces noise and neglects regular geometry elements in vector floorplans, resulting in jagged edges and discontinuous segmentation. Box-based methods [5, 12, 20], which represent rooms with bounding boxes, simplify the problem and are easier to vectorize but suffer from low-level geometry issues, such as misalignment, overlap, and displacement. Image-based methods can handle local details effectively, but they are difficult to vectorize. Box-based methods are more efficient to vectorize, but have limited representation and suffer from geometric issues. Therefore, existing methods rely on complex post-processing to obtain vector floorplans, which not only hinders the end-to-end design flow from requirements to floorplans but also introduces additional issues, such as parameter tuning and computational instability.

Our idea is to combine box-based representation and segmentation-based generation. To this end, we propose a novel floorplan patch representation, which divides the design space of floorplans into semantic patches of varying sizes, similar to superpixels in floorplan images, where the pixel value represents the semantics of floorplans. We aim to generate vector floorplans through *FloorplanSBS*, a novel generation framework, which directly synthesizes vector floorplans by patch division and labeling. Our framework follows a division-and-labeling scheme. Given the design constraint, we first divide the design space of floorplans into a set of rectangular patches with a division model. This stage also supports user-defined space division schemes. Then, we use a labeling model to predict the semantic label for each patch. The final vector floorplan can be directly obtained from the labeled patches. The division stage provides a clear structural framework for floorplan generation, while the labeling stage refines semantic labels, demonstrating the method's structure-aware capability. For sure, it naturally avoids common geometric issues like misalignment and overlapping.

Our method supports the generation of various constraints, such as boundary constraints and graph constraints. Experimental results, including both qualitative and quantitative analyses, demonstrate that our method outperforms state-of-the-art methods in generating high-quality floorplans. Our main contributions include:

*Corresponding author.

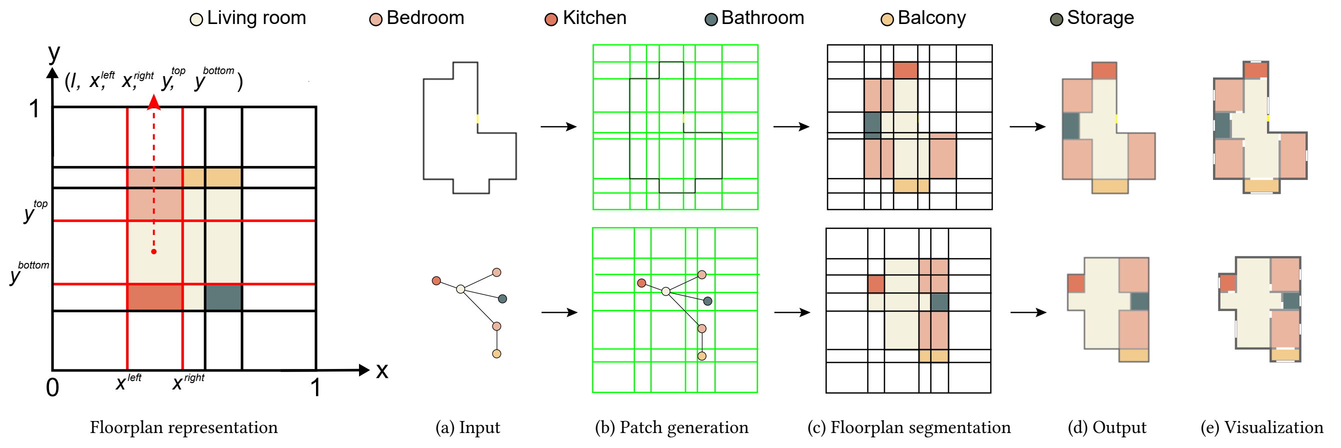


Figure 1: Overview of *FloorplanSBS*. Left: the patch-based vector floorplan representation. (a-e): the pipeline of our framework. Given the design constraint as input (a), our framework first divides the design space of the floorplan into patches (b). Then, we label the generated patches via floorplan segmentation (c). The final vector floorplan (d) can be directly obtained from the labeled patches. For visualization (e), the windows and interior doors can be added using a ruler-based method.

- a novel patch-based floorplan segmentation method that combines the box-based representation with segmentation-based generation and balances detailed generation with the ease of vectorization.
- a neural-network-only framework for vector floorplan generation, without any post-processing, and is simple to implement and user-friendly.

2 Related work

2.1 Optimization for floorplan generation

Floorplan generation can be achieved through layout optimization. Researchers have developed various intelligent optimization algorithms and procedural methods [1, 3, 21, 24, 27] to address this task. [17] uses a Bayesian network to generate bubble diagrams of flooprplan configurations. A stochastic optimization algorithm then transforms the diagram into a detailed floorplan. [15] considers functional requirements, design preferences, and manufacturing constraints such as cost and construction feasibility, and uses a nonlinear programming algorithm for floorplan optimization. [28] introduces a mixed-integer quadratic programming framework for floorplans. Rooms are represented as combinations of rectangles, and the framework iteratively refines floorplans from coarse to detailed. Optimization-based methods rely on carefully modeling and tedious parameter tuning, limiting their generalization. Their dependence on manual intervention reduces flexibility in practical applications. In contrast, our neural-based framework eliminates the need for manual constraint design and optimization, learning from data to provide a more generalizable solution.

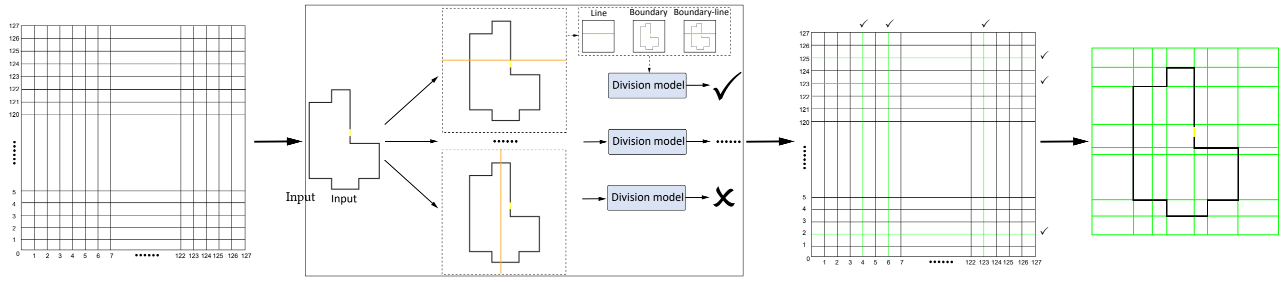
2.2 Deep learning for floorplan generation

Deep learning methods eliminate the need for tedious manual intervention in generating floorplans [4, 5, 16, 20]. Given the building boundary, *RPLAN* [29] iteratively predicts the category and position of rooms. Then, a semantic segmentation model is employed to predict the semantic map of walls, which is vectorized through post-processing to obtain the floorplan. *Graph2Plan* [12] uses a neural network to convert a bubble diagram into a floorplan. However, it focuses on generating room boxes and still requires post-processing to obtain floorplans. *WallPlan* [25] represents vector floorplans as wall maps and transforms into a graph generation problem. However, its generation process is image-based. Generative models (GAN [7] and diffusion model [10]) are widely applied in floorplan generation. [18, 19] take a topological graph as input and use the GAN to generate floorplan images, which require complex post-processing. *HouseDiffusion* [23] applies the diffusion model but encounters challenges with predefined room shape topology and difficulties stitching room polygens, limiting its generation quality. *Cons2Plan* [11] builds upon and extends HouseDiffusion by introducing conditional diffusion models to support more generation constraints. However, the generation quality issues remain. *GSDiff* [13] synthesizes vector floorplans through structural graph generation. However, the diffusion process struggles to ensure the alignment and completeness of wall junctions, and the semantic prediction is also complex and prone to errors. *FloorplanDiffusion* [32] adopts a diffusion model framework and supports multi-conditional inputs for floorplan generation. However, it can not directly produce vectorized outputs. Existing learning methods require post-processing (e.g., vectorization, alignment), increasing computational cost and instability. In contrast, our neural-network-only framework directly generates floorplans, reducing post-processing and ensuring constraint-aware, semantically coherent vector floorplans.

3 Method

3.1 Overview

Floorplan representation. The vector floorplan is placed in a unit-sized design space, which is then segmented using scanning lines along edges of each room polygon (Figure 1(a)). Horizontal and vertical scanning lines divide the design space into a grid with m rectangles, called a patch. Additionally, the semantic label of



(a) Discrete design space (b) Classification-based division model (c) Output (d) Floorplan patch

Figure 2: Patch generation. Given a discrete design space (a), we use a division model to perform the binary classification for each candidate scanning lines under the input design constraint (b), the output of our division model is a set of selected scanning lines (c), which are used to construct floorplan patches (d).

each patch is determined based on its position within the floorplan. Specifically, if a patch lies within the floorplan, it must be within a particular room, and the label of that room is assigned to this patch. If the patch lies outside the floorplan, an “*External*” label is assigned. Finally, a vector floorplan can be represented as a set of patches, denoted $\mathcal{F} = \{p_1, p_2, \ldots, p_m\}$, where each labeled patch is constructed by four surrounding scanning lines, denoted $p_i = \{l_i, x_i^{\text{left}}, x_i^{\text{right}}, y_i^{\text{top}}, y_i^{\text{bottom}}\}$, $x_i^{\text{left}}, x_i^{\text{right}}, y_i^{\text{top}}, y_i^{\text{bottom}} \in (0, 1)$. The semantic label is represented as an n-dimensional one-hot vector $l_i \in \mathbb{R}^n$, where n is the number of semantic labels of floorplans.

Methodology. Our goal is to generate high-quality vectorized floorplans from given design constraints. The main challenges we face include simplifying the generation process while ensuring high-quality generation, balancing the preservation of details with ease of vectorization, and avoiding reliance on complex post-processing and optimization. To this end, we propose a patch-based floorplan segmentation method (Figure 1). This method effectively leverages the powerful capabilities of neural networks for spatial partitioning and semantic prediction, combining the box-based representation and the segmentation-based generation. First, we use a division model to partition the floorplan design space into rectangular patches and then use a labeling model to predict the semantic label for each patch, ultimately obtaining the final vectorized floorplan. For simplicity, we will only introduce the architectural boundaries with a front door as input, with other constraints introduced later.

3.2 Patch generation

In this stage, we divide the design space of floorplans into a set of patches using a division model (Figure 2). The generation of patches plays a crucial role in determining the basic layout of the floorplan, as it defines the spatial organization of the entire design. Each patch is characterized by its spatial boundaries within the overall floorplan layout, and the division model is trained to predict these boundaries along with the potential positions of the patches.

Division model. Each patch is defined by four scanning lines. Instead of treating the prediction of these scanning lines as a complex, unstable, and difficult-to-optimize regression task, we formulate patch generation as a classification task for scanning lines. We discretize the unit-sized design space into a 128×128 grid, following

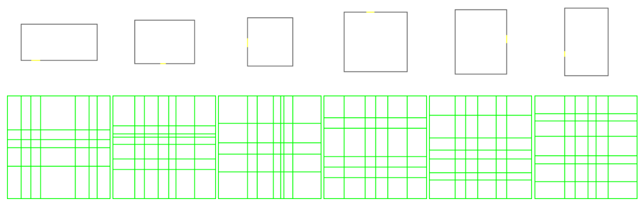


Figure 3: Patch generation with rectangular boundaries.

the resolution of floorplan annotations in the dataset. This process divides the design space into 128 horizontal scanning lines and 128 vertical scanning lines, with each scanning line serving as the candidate. The goal is to select a subset of these candidate scanning lines to construct patches. Rather than performing a 128-class classification task, we treat each candidate scanning line as an independent binary classification task. For each scanning line, a classification network predicts whether it is selected or not. By converting the patch generation problem into a binary classification problem for all scanning lines, we avoid the complexity of regression tasks.

In our classification network, we use a modified version of *ResNet-34* [8] as the backbone network. We represent the input boundary as a (128×128) image. Therefore, the input to the classification network is a (128×128) multi-channel image (defaults to 0), including the mask image of the boundary and the candidate scanning line:

- *Line mask*: the mask of the candidate scanning line.
- *Boundary mask*: the mask of the input boundary, with different pixel values for the exterior walls and the front door.
- *Boundary-line mask*: the union mask of *Boundary mask* and *Line mask*.

Training and inference. In the training, for each candidate scanning line, the network performs binary classification to determine whether the candidate scanning line should be part of the final patch boundary under the input constraint. We use the standard cross-entropy loss function and the Adam optimizer [14] for training. In the inference, given the input constraint, through iteratively performing the binary classification of each candidate scanning line, the final output is a set of selected scanning lines, which are used to define the boundaries of floorplan patches. While our patch division is guided by the input boundary, it is not rigidly constrained by it.

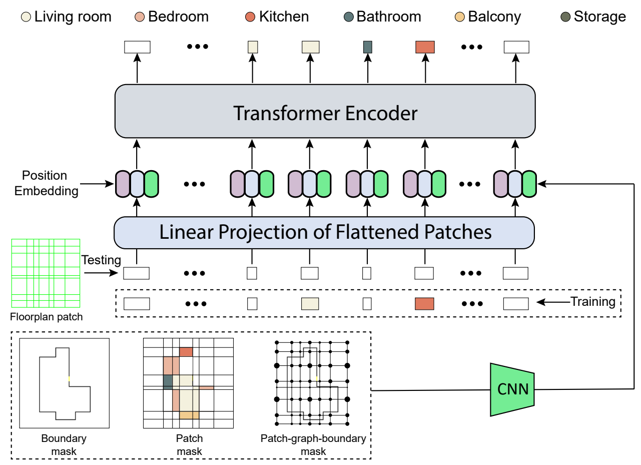


Figure 4: Floorplan segmentation. We adopt a ViT-based labeling model for floorplan segmentation. In the training, we adopt a progressive training strategy. We randomly retain a certain proportion of the semantic labels of the patches as input and predict the semantic labels for the remaining patches. During the inference phase, the network outputs the semantic labels for all patches in a single pass.

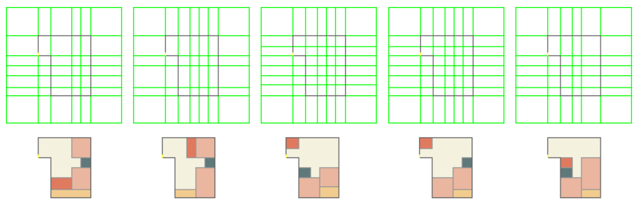


Figure 5: Our generated floorplans from the same boundary with different patch divisions.

The division model learns spatial priors from data and generates scanning lines based on both geometry and semantic context. As a result, even with a simple rectangular boundary, the model can produce diverse and meaningful patches, as shown in Figure 3.

3.3 Floorplan segmentation

Given the design constraint and the pre-divided patches, the task of floorplan segmentation is to use a labeling model to assign semantic labels to each patch, which naturally avoids common geometric issues like misalignment and overlapping in floorplan generation.

Labeling model. Our labeling model is inspired by Vision Transformer (ViT) [6]. Just as ViT divides the image into small patches for processing, we have partitioned the floorplan design space into multiple rectangular patches. However, instead of regular image patches, we use semantically meaningful regions derived from prior-based architectural divisions. The sequence of flattened patches, as well as the input constraint, is fed into the labeling model, as shown in Figure 4. The self-attention mechanism effectively captures spatial relationships between patches in the floorplan, allowing the model to focus on relevant patches and better understand complex layouts. After the self-attention processing, the patch embeddings undergo further processing through a series of feed-forward neural networks to extract additional contextual information. Ultimately, the labeling model outputs a prediction vector for each patch, using it to generate a semantic label. We use a modified version of *Transformer* [26] as the backbone network. Moreover, the processing aligns naturally with our patch-wise representation, enabling accurate semantic labeling of structured patches.

We represent the input boundary as a (128×128) image (defaults to 0), including the mask image of the boundary and the candidate scanning line. We further incorporate spatial topology via a graph-based patch adjacency mask and embed design constraints into the input, enabling conditional and structure-aware generation. The details are as follows:

- *Boundary mask*: the mask of the input boundary, with different pixel values for the exterior walls and the front door.
- *Patch mask*: the mask of the divided patches, with a pixel value of 0 for the unlabeled patches.
- *Patch-graph-boundary mask*: the union mask of *Boundary mask* and the mask of the patch graph. The patch graph is represented as a graph, where the center of each patch serves as a node, visualized as a circle proportional to the size of the patch. Adjacent patch nodes are connected by edges, which are visualized as line segments.

Figure 4 provides an example of these masks. The input constraint is encoded into a 128-dimensional feature vector (conditional feature) using a modified version of *ResNet-34* [8]. The sequence of flattened patches is first padded to the same dimension. Then, it is encoded into a 128-dimensional embedding via a linear projection, while position encodings are added, resulting in a 128-dimensional embedding (patch feature). The patch and conditional features are concatenated and input to the Transformer encoder, outputting the predicted probabilities for all labels for each floorplan patch.

Training and inference. Directly training the network to predict semantic labels from totally unlabeled patches is challenging. Therefore, we adopt a progressive training strategy to address the sparse supervision in patch-based floorplan labeling. Unlike standard segmentation tasks with dense labels, our model learns semantics from partial labels and spatial layout. During training, we gradually reduce the proportion of labeled patches, allowing the model to learn to propagate contextual information. This is particularly effective for Transformer-based models, which leverage self-attention to capture long-range dependencies and semantic relations across patches. Specifically, during the training process, we randomly retain a certain proportion of the patch labels as input and predict the semantic labels for the remaining patches. As training progresses, we gradually decrease the proportion of labeled patches until zero. This progressive training strategy aids in stabilizing the training process and accelerating convergence. In this way, our model can better handle the complexity of predicting semantic labels. In the training, we optimize the labeling network by minimizing the standard cross-entropy loss. The Adam optimizer [14] is used. During the inference phase, the semantics prediction network takes the constraints and divides the unlabeled patches as input. The network outputs the semantic labels for all patches in a single pass. Furthermore, this stage supports user-defined space division, meeting specific design requirements. We show floorplans generated from the same boundary with different patches in Figure 5, illustrating the impact of different patch divisions.

3.4 Floorplan vectorzation

Through patch-based floorplan segmentation, we convert the design space into a series of labeled patches, which simplifies the floorplan vectorization process. By merging adjacent patches with the same semantic label, we can easily reconstruct the floorplan. Single patches labeled with specific room types, such as the bathroom and balcony, directly form rectangular rooms. When multiple patches with the same label are combined, they form more complex polygonal-shaped rooms, such as living rooms and bedrooms. This method enables efficient vectorization by treating each room or functional area as a geometric primitive (e.g., rectangle or polygon). The result is a vectorized floorplan that preserves the spatial and functional structure of the original design.

By formulating floorplan generation as a segmentation task over predefined patches, both semantic and spatial consistency are handled within the network. The explicit patch division guarantees well-aligned regions, while the labeling model directly predicts semantic labels without requiring geometric correction. As a result, our method produces clean vector outputs in a single forward pass, avoiding unstable and error-prone post-processing.

4 Experiment

Our two models, the division model and the labeling model, are implemented using Pytorch and separately trained on an NVIDIA GeForce GTX 4090 GPU. Training requires about 2.5 hours (300 epochs) for the division model (21 million parameters) and 6.5 hours (600 epochs) for the labeling model (30 million parameters). For more implementation details, we would like to release the code. We use a large-scale residential floorplan dataset *RPLAN*, containing more than 80K residential floorplans with dense annotation, which is presented in [29]. We use the processed verison of *RPLAN* dataset by *Graph2Plan* [12], with the same dataset splits of *Graph2Plan*: 70%, 15%, and 15% for training, validation, and testing, respectively. There are 13 room categories in the dataset: *LivingRoom*, *MasterRoom*, *SecondRoom*, *GuestRoom*, *ChildRoom*, *StudyRoom*, *DiningRoom*, *Bathroom*, *Kitchen*, *Balcony*, *Storage*, *Wall-in*, and *Entrance.* During progressive training, the unlabeled patch ratio is gradually increased: 0.5 before epoch 200, 0.7 before epoch 300, 0.9 before epoch 400, and 1.0 after epoch 400.

Constrained generation. *FloorplanSBS* supports various types of design constraints. During both training and inference, we can easily adapt the framework by replacing the input constraint. Our framework remains unchanged and adapts to different constraints simply by modifying the input masks. In our experiments, we further explore floorplan generation under layout graph constraints [12], where nodes represent rooms and edges denote room adjacencies. Each node also carries additional attributes, such as the room's center position and size. Since our framework is built on image-based neural networks, we convert the layout graph into a raster image to match our input format. Specifically, the graph is visualized at a fixed geometric scale, with each node rendered as a labeled circle representing a room, and the size of each circle is proportional to the room's area. Adjacent nodes are connected by line segments to indicate spatial relationships. For layout graph constraints, the boundary is still required as input. Thus, the division and labeling models take as input the boundary mask, patch-graph-boundary mask, and graph mask, combined as a multi-channel input. For topological graph constraints, the boundary is unnecessary, so we simply exclude the boundary mask from the input.

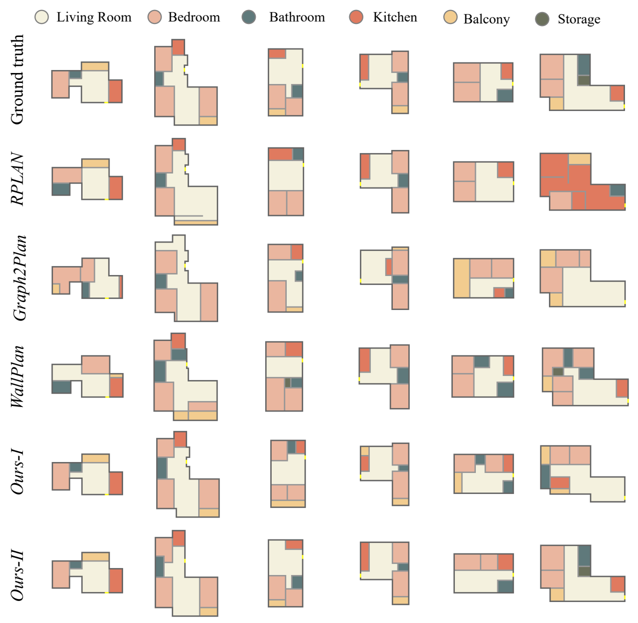


Figure 6: Qualitative evaluation of floorplan generation based on the boundary constraint.

4.1 Qualitative evaluation

We present two generation versions of our framework: *Ours-I*, which generates floorplans based on the generated floorplan patches using our division model, and *Ours-II*, which generates floorplans based on the ground-truth floorplan patches.

Boundary-constrained generation. We qualitatively evaluate our method by comparing with *RPLAN* [29], *Graph2Plan* [12] and *WallPlan* [25] under the boundary constraint. Figure 6 shows the generated floorplans of various methods. *RPLAN* often produces unclosed walls, leading to misclassifications (e.g., misclassifying the living room) and unnecessary walls, as seen in the second and sixth columns. *Graph2Plan* generates walls corresponding to rooms, but some room locations are illogical, and certain room categories are missing. In the third and fourth columns, rooms appear in illogical positions, reducing layout plausibility. *Graph2Plan* does not omit essential rooms but often creates rooms with unreasonable sizes or excessive numbers of rooms, as shown in the first and second columns. In contrast, our method consistently generates reasonable layouts. Additionally, by constructing rooms from multiple patches, we directly model the relationship between room type and

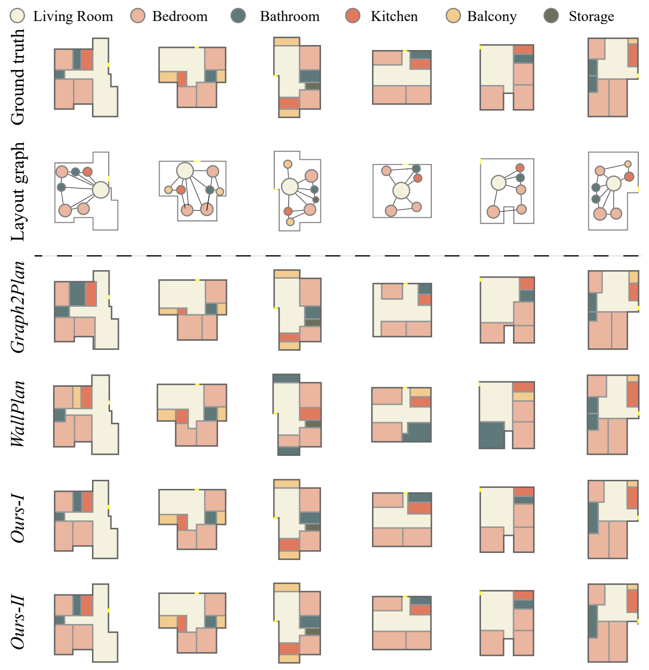


Figure 7: Qualitative evaluation of floorplan generation based on the layout graph constraint.

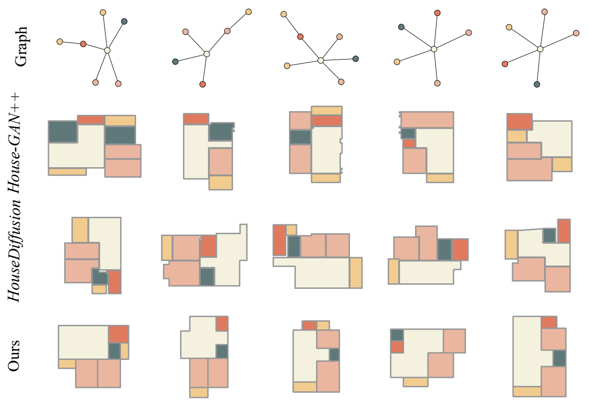


Figure 8: Qualitative evaluation of floorplan generation based on the topological graph constraint.

area, avoiding unreasonable room sizes. *Ours-II*, using ground-truth segmentations, outperforms *Ours-I*, providing results that more closely resemble the ground truth. This shows that more accurate segmentations lead to better results.

Graph-constrained generation. We compare the results of graph-constrained generation with *Graph2Plan* [12] and *WallPlan* [25] (Figure 7). Note that the graph-constrained generation requires the boundary to be used jointly as input. The room areas in our results are more consistent with the ground truth, particularly for smaller rooms. Our method ensures that no rooms are omitted, whereas *Graph2Plan* and *WallPlan* occasionally miss certain room categories.

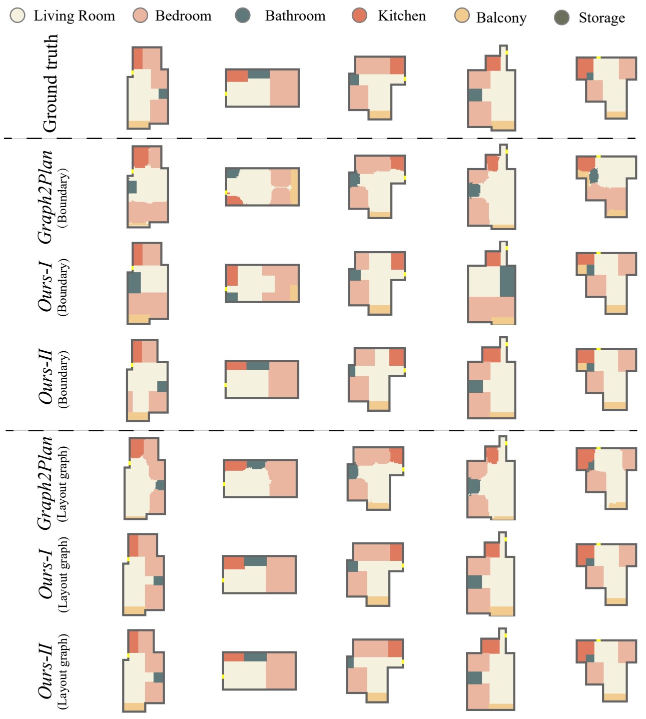


Figure 9: The intermediate room maps generated by different methods. Our results exceed the baseline method.

For instance, in the first column, *Ours-I* produces results that are nearly identical to the ground truth, while *Graph2Plan* shows noticeable deviations in the bathroom and kitchen, and *WallPlan* generates incorrect room types. Overall, our method produces more semantically and geometrically plausible floorplans, with room areas closer to the ground truth. Furthermore, *Ours-I* produces results comparable to *Ours-II*, which uses the ground-truth patches, indicating that the layout graph offers sufficient structural and semantic guidance for both patch division and labeling. We also compare the floorplans generated by our method with those produced by *House-GAN++* [19] and *HouseDiffusion* [23]. Both methods generate floorplans solely from a topological graph, without requiring the boundary as input. Additionally, the topological graph used in these methods does not contain information about room positions or sizes. Our method can also generate floorplans using only the topological graph as a constraint, enabled by a simple strategy. Specifically, we adopt a retrieval-based approach similar to *WallPlan* [25]: Given a topological graph, we retrieve a matching layout graph from a predefined dataset. The retrieved layout graph is then used to guide the constrained floorplan generation. Figure 8 shows that the quality of floorplans generated by our method is much higher than that of *House-GAN++* and *HouseDiffusion.*

More qualitative evaluation. We also perform a qualitative evaluation of the intermediate results. Figure 9 shows the intermediate room maps produced by our method and *Graph2Plan*. Our generated semantic maps do not have jagged boundaries or outliers. Our method, based on patch division, ensures horizontal or vertical

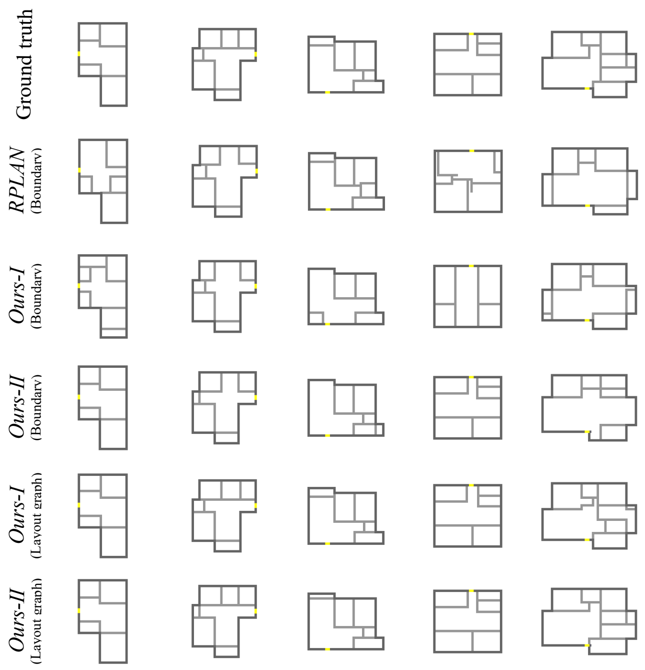


Figure 10: The intermediate wall maps produced by different methods. Our results exceed the baseline method.

Method	MMD (↓)	FID (↓)	KID (↓)	mACC (↑)	mIoU (↑)
RPLAN (Boundary)	3.498	4.975	51.53	0.6563	0.7007
Graph2Plan (Boundary)	3.507	1.758	7.466	0.6282	0.6817
WallPlan (Boundary)	3.525	1.91	10.01	0.6654	0.7118
Ours-I (Boundary)	3.463	1.6079	4.662	0.698	0.7378
Ours-II (Boundary)	3.076	1.209	4.749	0.8465	0.8706
Graph2Plan (Layout graph)	3.419	1.027	3.119	0.8649	0.9303
WallPlan (Layout graph)	3.363	1.078	4.943	0.9157	0.9577
Ours-I (Layout graph)	2.215	0.4387	1.430	0.9441	0.9938
Ours-II (Layout graph)	2.040	0.5472	2.772	0.9685	0.9952

Table 1: The quantitative evaluation of floorplans generated by different methods. KID and MMD are scaled by 10^4.

Method	FID (↓)	KID (↓)	MMD (x10^4) (↓)
House-GAN++	73.5187	0.0842	4.1716
HouseDiffusion	11.7128	0.0115	**4.1694**
Ours	**9.0288**	**0.0102**	8.8742

Table 2: The quantitative evaluation of floorplan generation under the topological graph constraint.

boundaries with tightly arranged patches, resulting in no gaps or distorted boundaries. We compare the intermediate wall maps with *RPLAN*, as shown in Figure 10. Our walls are always closed, while *RPLAN* sometimes generates unclosed segments, which can affect subsequent results. Comparing *Ours-I* (Boundary) with *Ours-I* (Layout graph), the latter, with ground truth segmentations, generates results very close to the ground truth. Additionally, *Ours-II* (Layout graph), using ground truth segmentations, outperforms *Ours-I* (Layout graph) in more complex wall scenarios.

Method	MMD (↓)	FID (↓)	KID (↓)	mACC (↑)	mIoU (↑)	RI (↑)
Graph2Plan (Boundary)	3.5094	53.3386	716.8	0.6554	0.6554	0.2318
Ours-I (Boundary)	3.468	1.0554	1.665	0.6998	0.7003	0.4137
Ours-II (Boundary)	3.242	0.8436	2.768	0.8338	0.8388	0.6421
Graph2Plan (Layout graph)	3.506	52.6304	708.1	0.8368	0.8368	0.4074
Ours-I (Layout graph)	2.601	0.394	1.05	0.9566	0.9585	0.8726
Ours-II (Layout graph)	2.712	0.5788	2.633	0.9579	0.9636	0.8796

Table 3: The evaluation of intermediate room maps generated by different methods. KID and MMD are scaled by 10^4.

Method	MMD (↓)	FID (↓)	KID (↓)
RPLAN (Boundary)	3.49	2.456	19.25
Ours-I (Boundary)	3.457	1.3041	4.24
Ours-II (Boundary)	3.039	0.9589	4.28
Ours-I (Layout graph)	2.306	0.36	5.724
Ours-II (Layout graph)	3.545	0.4291	1.394

Table 4: The evaluation of intermediate wall maps generated by different methods. KID and MMD are scaled by 10^4.

4.2 Quantitative evaluation

We compare our method with *RPLAN*, *Graph2Plan*, and *WallPlan*. The results for the boundary-based and graph-based generation are shown in Table 1, and our method achieves the best performance across all metrics. Regarding generation quality metrics, *Ours-I* performs best in FID (Fréchet Inception Distance) [9] and KID (Kernel Inception Distance) [2], achieving the lowest scores, indicating the smallest gap between generated samples and the real distribution. This means our method generates more realistic and credible results. Additionally, inspired by point cloud evaluation metrics [31], we also calculate MMD (Maximum Mean Discrepancy). Our method shows the lowest deviation in MMD, further confirming high consistency between generated and real data distributions. Our method performs best in IoU (Intersection over Union) and ACC (Accuracy), reflecting high precision and consistency in target region segmentation. Moreover, *Ours-II* (Boundary) and *Ours-II* (Layout graph) show improvements across all metrics when using ground truth segmentations, especially in mACC, demonstrating the significant impact of high-quality segmentations on performance. Table 2 presents the quantitative evaluation of floorplan generation under the topological graph constraint. The results demonstrate that our method significantly outperforms *House-GAN++* and *HouseDiffusion* in terms of generation quality.

The results of intermediate room maps generated by different methods are shown in Table 3. *Graph2Plan* may produce jagged pixels between rooms, which do not occur in our results. To evaluate this, we use RI [30], the product of the average IoU of matched rooms and F1-score, to evaluate room integrity. *Ours I* performs well across all metrics, particularly in RI, where our results are significantly better. The intermediate results from *Graph2Plan* contain many outlier pixels, reducing IoU/TP values and increasing false positives, which affects RI.

The results of intermediate wall maps generated by different methods are shown in Table 4. *Ours-I* (Layout graph) outperforms *Ours-II* (Layout graph) in FID and KID, showing that our model can generate wall results similar to those produced with ground truth segmentations when given bubble diagram inputs. At the

Method	MMD (↓)	FID (↓)	KID (↓)	mACC (↑)	mIoU (↑)
Regression model	3.069	1.012	5.925	**0.9441**	0.9218
Ours	**2.215**	**0.4387**	**1.430**	**0.9441**	**0.9938**

Table 5: Ablation study for the division model under layout graph constraints. KID and MMD are scaled by 10^4.

Method	mAcc (↑)	RI (↑)
w/o division model (Using boundary lines)	0.5795	0.3020
w/o division model (Using regular patches)	0.5673	0.1725
w/o Transformer (Using GAT)	0.9140	0.7324
w/o progressive training	0.9163	0.7602
Ours	**0.9273**	**0.8322**

Table 6: Ablation study under the boundary constraint.

same time, *Ours-I* (Boundary) achieves the second-best KID value, demonstrating that the wall details generated by our model based on the boundary are close to the real ground truth. This indicates that our segmentation line and patch classification algorithms work well together to produce walls closely matching the ground truth.

4.3 Ablation study

Our model consists of two main components: a division model for patch generation and a labeling model for floorplan segmentation. We conduct ablation studies on each component individually. The ablation experiments demonstrate the effectiveness of our method, highlighting the contributions of each component.

For the division model, our default method employs a classification network to generate floorplan patches. As a baseline, we implement a regression-based variant that directly predicts the coordinates of horizontal and vertical scanning lines. The comparison results under the layout graph constraint are shown in Table 5. To further validate the effectiveness of our division model, we perform additional ablation experiments with two simplified alternatives: (1) w/o division model (Using boundary lines): patches are generated by extending and intersecting the boundary lines directly, without any learned division. (2) w/o division model (Using regular patches): the design space is evenly divided into a regular grid without considering input constraints or semantic structure. The comparison results are shown in Table 6.

For the labeling model, we adopt a Transformer-based architecture as the backbone. A progressive training strategy is applied, in which a proportion of patch labels is randomly retained as input, and this proportion is gradually reduced to zero throughout training. We conduct ablation experiments under the boundary constraint to evaluate the contributions of different components in the labeling model. Specifically, we replace the Transformer with a Graph Attention Network (GAT) [22] to examine the impact of the attention mechanism (w/o Transformer (Using GAT)). In addition, we evaluate the effect of removing the progressive training strategy by directly training the model without partial label supervision (w/o progressive training). The results are presented in Table 6. Note that we calculate the mAcc of patches.

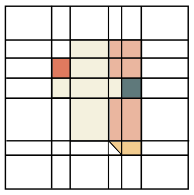
Method	Avg. Time (s)	Std. mACC	Std. mIoU	Std. RI
RPLAN (Boundary)	4.00	0.2027	0.2110	0.2529
Graph2Plan (Boundary)	0.40	0.1930	0.2049	0.2358
WallPlan (Boundary)	0.74	0.1840	0.1908	**0.2161**
Ours (Boundary)	**0.12**	**0.1481**	**0.1746**	0.2487

Table 7: Computational analysis of different methods under the boundary constraint.

4.4 Computational analysis

We further compare the computational efficiency of different methods under the boundary constraint. The average execution time for generating a single floorplan is summarized in Table 7. *RPLAN* is the slowest, taking approximately 4.00 seconds per sample due to its two-stage generation process and raster-to-vector conversion. *Graph2Plan* is more efficient, with an average runtime of 0.40 seconds, while *WallPlan* takes about 0.74 seconds. In contrast, our method is significantly faster, requiring only 0.12 seconds to generate a complete floorplan. For fairness, all methods are evaluated under the same hardware environment using an NVIDIA GeForce RTX 3090 GPU. Beyond efficiency, we also assess generation stability by computing the standard deviation of evaluation metrics across multiple runs. Our method consistently demonstrates high stability with low variance in performance. In contrast, *RPLAN*, *Graph2Plan*, and *WallPlan* exhibit greater variability, largely due to their reliance on non-deterministic post-processing steps, such as raster-to-vector conversion or heuristic refinement.

5 Conclusion



In this paper, we proposed *FloorplanSBS*, a novel framework for the automatic generation of vector floorplans based on patch-based segmentation. This method allows for the direct synthesis of vector floorplans without requiring post-processing. Extensive experiments demonstrate that *FloorplanSBS* outperforms existing methods in generating high-quality floorplans. Some limitations remain and point to promising future directions. Our current patch division is relatively rigid, using axis-aligned patches that may struggle to capture complex or curved structures. Our framework can be extended to non-axis-aligned floorplan generation by adopting a hybrid grid with different patch shapes, such as rectangles and triangles. For curved floorplans, we first generate straight-line floorplans and then transform selected segments into smooth curves, such as Bézier curves or arcs. While our method performs well across various constraints and metrics, it has some limitations. In rare cases, the division model may fail to produce ideal scanning lines, especially with irregular or narrow boundaries, leading to fragmented patches and minor semantic errors. Although our method avoids post-processing, semantically implausible layouts may still occur, such as placing incompatible rooms adjacent. This stems from the local nature of patch-level predictions and the lack of explicit design constraints. Future work may incorporate architectural priors or constraint-aware training to further enhance layout realism.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive suggestions and comments. This work is supported by the National Natural Science Foundation of China (Grant No. 62372152) and the Open Project Program of the State Key Laboratory of CAD&CG (Grant No. A2412), Zhejiang University.

References

[1] Arash Bahrehmand, Thomas Batard, Ricardo Marques, Alun Evans, and Josep Blat. 2017. Optimizing layout using spatial quality metrics and user preferences. *Graphical models* 93 (2017), 25–38.

[2] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. 2018. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401* (2018).

[3] Sumit Bisht, Krishnendra Shekhawat, Nitant Upasani, Rahil N Jain, Riddhesh Jayesh Tiwaskar, and Chinmay Hebbar. 2022. Transforming an adjacency graph into dimensioned floorplan layouts. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 5–22.

[4] Stanislas Chaillou. 2020. ArchiGAN: Artificial Intelligence x Architecture. In *Architectural Intelligence*. Springer, 117–127.

[5] Qi Chen, Qi Wu, Rui Tang, Yuhan Wang, Shuai Wang, and Mingkui Tan. 2020. Intelligent home 3d: Automatic 3d-house design from linguistic descriptions only. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12625–12634.

[6] Alexey Dosovitskiy. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems* 30 (2017).

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems* 33 (2020), 6840–6851.

[11] Shibo Hong, Xuhong Zhang, Tianyu Du, Sheng Cheng, Xun Wang, and Jianwei Yin. 2024. Cons2Plan: Vector Floorplan Generation from Various Conditions via a Learning Framework based on Conditional Diffusion Models. In *Proceedings of the 32nd ACM International Conference on Multimedia*. 3248–3256.

[12] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. 2020. Graph2plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 118–1.

[13] Sizhe Hu, Wenming Wu, Yuntao Wang, Benzhu Xu, and Liping Zheng. 2024. GSDiff: Synthesizing Vector Floorplans via Geometry-enhanced Structural Graph Generation. *arXiv preprint arXiv:2408.16258* (2024).

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Han Liu, Yong-Liang Yang, Sawsan AlHalawani, and Niloy J Mitra. 2013. Constraint-aware interior layout exploration for pre-cast concrete-based buildings. *The Visual Computer* 29, 6 (2013), 663–673.

[16] Ziniu Luo and Weixin Huang. 2022. FloorplanGAN: Vector residential floorplan adversarial generation. *Automation in Construction* 142 (2022), 104470.

[17] Paul Merrell, Eric Schkufza, and Vladlen Koltun. 2010. Computer-generated residential building layouts. In *ACM SIGGRAPH Asia 2010 papers*. 1–12.

[18] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. 2020. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *European Conference on Computer Vision*. Springer, 162–177.

[19] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. 2021. 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*. 13632–13641.

[20] Wamiq Para, Paul Guerrero, Tom Kelly, Leonidas J Guibas, and Peter Wonka. 2021. Generative layout modeling using constraint graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6690–6700.

[21] Julian F Rosser, Gavin Smith, and Jeremy G Morley. 2017. Data-driven estimation of building interior plans. *International Journal of Geographical Information Science* 31, 8 (2017), 1652–1674.

[22] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[23] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. 2023. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5466–5475.

[24] Krishnendra Shekhawat, Nitant Upasani, Sumit Bisht, and Rahil N Jain. 2021. A tool for computer-generated dimensioned floorplans based on given adjacencies. *Automation in Construction* 127 (2021), 103718.

[25] Jiahui Sun, Wenming Wu, Ligang Liu, Wenjie Min, Gaofeng Zhang, and Liping Zheng. 2022. Wallplan: synthesizing floorplans by learning to generate wall graphs. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–14.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[27] Xiao-Yu Wang and Kang Zhang. 2020. Generating layout designs from high-level specifications. *Automation in Construction* 119 (2020), 103288.

[28] Wenming Wu, Lubin Fan, Ligang Liu, and Peter Wonka. 2018. MIQP-based Layout Design for Building Interiors. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 511–521.

[29] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. 2019. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–12.

[30] Bingchen Yang, Haiyong Jiang, Hao Pan, and Jun Xiao. 2023. Vectorfloorseg: Two-stream graph attention network for vectorized roughcast floorplan segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1358–1367.

[31] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. 2019. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4541–4550.

[32] Pengyu Zeng, Wen Gao, Jun Yin, Pengjian Xu, and Shuai Lu. 2024. Residential floor plans: Multi-conditional automatic generation using diffusion models. *Automation in Construction* 162 (2024), 105374.