# End-to-end Graph-constrained Vectorized Floorplan Generation with Panoptic Refinement

Jiachen Liu[1†], Yuan Xue[2†], Jose Duarte[3], Krishnendra Shekhawat[4], Zihan Zhou[5], and Xiaolei Huang[1]

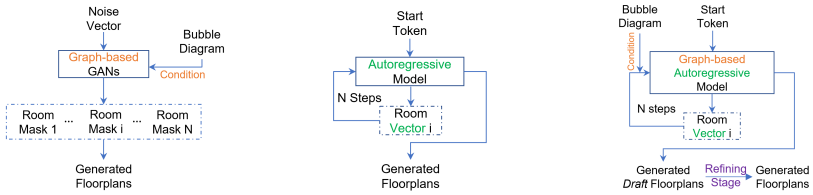
[1] College of Information Sciences and Technology, The Pennsylvania State University
[2] Department of Electrical and Computer Engineering, Johns Hopkins University
[3] College of Arts and Architecture, The Pennsylvania State University
[4] Department of Mathematics, BITS Pilani
[5] Manycore Tech Inc.

**Abstract.** The automatic generation of floorplans given user inputs has great potential in architectural design and has recently been explored in the computer vision community. However, the majority of existing methods synthesize floorplans in the format of rasterized images, which are difficult to edit or customize. In this paper, we aim to synthesize floorplans as sequences of 1-D vectors, which eases user interaction and design customization. To generate high fidelity vectorized floorplans, we propose a novel two-stage framework, including a *draft stage* and a multi-round *refining stage*. In the first stage, we encode the room connectivity graph input by users with a graph convolutional network (GCN), then apply an autoregressive transformer network to generate an initial floorplan sequence. To polish the initial design and generate more visually appealing floorplans, we further propose a novel panoptic refinement network (PRN) composed of a GCN and a transformer network. The PRN takes the initial generated sequence as input and refines the floorplan design while encouraging the correct room connectivity with our proposed geometric loss. We have conducted extensive experiments on a real-world floorplan dataset, and the results show that our method achieves state-of-the-art performance under different settings and evaluation metrics.

## 1 Introduction

House design is essential yet challenging work for professional architects, usually requiring extensive collaboration and multi-round refinement. Floorplan design is a crucial part of house design, which involves designing the room layouts and their connectivities such as walls and doors. With the availability of several large-scale floorplan benchmarks [4,5,24] and advance in generative models such as generative adversarial networks (GANs) [6], generating floorplans automatically has recently attracted the attention and interest of architects as well as computer vision researchers [10,19].

[†] These authors contributed equally to this work.




**(a)** Previous rasterized methods [17,18]. **(b)** Previous vectorized methods [20]. **(c)** Our proposed vectorized method.

Fig. 1: Comparisons between different floorplan generation pipelines.

Motivated by the interactive design process, previous works generated floorplans with different input constraints, such as building boundaries [9,24], room dimensions [22], or bubble diagrams that describe room numbers, room types, and adjacency or connectivity information [17,18]. There is also work which explores floorplan generation from scratch in an unconditional way (*e.g.*, [20]). Since architects usually preset room categories and their connectivities before designing the floorplans, we find the setting by HouseGAN [17] to be the closest to the design practice. We thus follow their setup throughout this paper for interactive floorplan generation.

Current state-of-the-art methods with conditional inputs [17,18] use graph-based GANs to generate room layouts as rasterized images. Although they have shown superior generation realism, we argue that GAN-generated rasterized images have several limitations in representing floorplan layouts. First, it is challenging for GANs to learn geometric layout properties such as axis-aligned walls. Second, some input conditions such as room numbers cannot be explicitly expressed in the rasterized floorplans. Last but not least, rasterized masks are irregular representations, which have restricted the options for users or architects to refine or customize the generated floorplans. In this paper, we propose to represent floorplan layouts as sequences of 1-D vectors describing the room bounding box coordinates, and generate vectorized floorplans in an end-to-end fashion. With a concise and user-friendly representation, generating vectorized floorplans enables efficient inference, more accessible user interactions, and connected space representation.

Mimicking the floorplan design process, we propose a novel two-stage framework conditioned on input bubble diagrams and gradually refine the generated floorplan through multiple steps. Fig. 1 illustrates the difference between our proposed framework and previous methods. In the first stage (*i.e.*, *draft stage*), we apply graph convolutional networks (GCNs) to encode the room connectivity from the bubble diagram, then an initial room sequence is generated with an autoregressive self-attention transformer network following previous layout generation works [7,20]. However, the generated draft result may be unsatisfactory since the autoregressive models cannot access the global information from the entire sequence at each generation timestamp. Thus, in the second stage (*i.e.*, refining stage in Fig. 2), we propose a panoptic refinement network (PRN) to

refine the draft floorplan design for a more visually appealing appearance while keeping the correct room connectivity. Taking the entire floorplan sequence from the draft stage as input, a GCN in the panoptic refinement network first processes the generated sequence to refine the connectivity relationship. Another transformer is further applied to exploit the contextual information of the whole sequence. In addition, a novel geometric loss based on layout overlapping is proposed to encourage the correct layout of the generated floorplan.

The main contributions of our paper can be summarized as follows:

- We propose a novel two-stage framework to generate vectorized floorplans from user inputs. To the best of our knowledge, our work is the first to allow the end-to-end conditional generation of vectorized floorplans.
- In the draft stage, we integrate GCNs into a transformer-based autoregressive model and show how to effectively capture the room connectivity relationships provided in the user inputs.
- In the refining stage, we propose a panoptic refinement network for multi-round refinements of the draft floorplans, while encouraging the correct room connectivity with our proposed geometric loss.
- Extensive experiments on a real-world floorplan dataset show that our method outperforms previous state-of-the-art approaches under most settings.

## 2 Related Work

**Data-driven floorplan generation** Various data-driven approaches have been proposed in recent years for automatic floorplan generation. Mainstream methods primarily focused on generating floorplans as rasterized images. RPLAN [24] created a large-scale annotated dataset from residential buildings and proposed a two-stage approach to generate rooms from a given boundary. Graph2Plan [9] proposed to first generate layout-constrained floorplans as rasterized images via convolutional neural networks (CNNs) and graph neural networks (GNNs), then applied an offline optimization algorithm to align the rooms and transform them into a vectorized floorplan, given the boundary and graph-constrained layout as input. HouseGAN [17] took the bubble diagram of the rooms as user input, and introduced a GAN-based architecture to generate each room. HouseGAN++ [18] was the extension of HouseGAN and the authors proposed an improved GAN-based network, using a GT-conditioning training strategy as well as a list of test-time meta-optimization algorithms.

While previous rasterized floorplan generation methods have achieved promising results, their applicability is often limited by the rasterized representations. In practice, architects mostly work with vectorized representation [25] of floorplans due to its flexibility and geometric compatibility. To leverage vectorized representations, existing methods [18,24] often converted rasterized floorplans into vectorized representations via postprocessing steps [3]. However, the representation power is yet limited by the originally generated rasterized images. To this end, we directly represent the floorplan rooms as vectorized bounding boxes

in our method. More recently, Para *et al.* [20] also utilized vectorized representation to generate floorplans using an autoregressive model and focused more on unconditional generation. More specifically, a transformer was used to predict elements including room coordinates and sizes in an autoregressive manner, followed by another transformer to generate edges such as walls and doors. A post-processing optimization step was adopted to refine the output from the networks to obtain final layouts. Despite using the same floorplan representations, our work has several essential differences from [20]. First, our model enables interaction with graph-based user input, which satisfies the room connectivity constraints. Second, we encode constraints such as doors into the learning objectives of the same network, skipping the need to use a separate model to learn constraints as in [20]. Moreover, unlike [20] which applied linear programming regularization to optimize network output, we propose a panoptic refinement network that refines the draft floorplan while encouraging the correct room connectivity in an end-to-end fashion.

**Autoregressive models for layout generation** Floorplan generation can be interpreted as a special case of layout generation. In the literature, the autoregressive model has been widely used in layout or shape generation tasks [2,7,11,15,16] due to its superior representation ability to model sequential relationships. LayoutVAE [11] proposed a variational auto-encoder (VAE) [13] architecture conditioned on priors such as element numbers and types. LayoutTransformer [7] applied a transformer-based architecture to learn layout attributes autoregressively in an unconditional manner. VTN [2] instantiated the VAE framework with a transformer network and generated layouts in a self-supervised manner. However, there are two potential limitations for transformer-based autoregressive models in layout generation [15]. First, the information flow is unidirectional and immutable in the autoregressive model since the model only has access to previously generated results during the iterative generation. Second, most existing transformers only considered unconditional layout generation without user inputs and thus are not suitable for an interactive generation. To mitigate such limitations, we enable interactive generation by integrating GCNs into the transformer for draft floorplan generation. Unlike existing refinement method for scene layout generation which only refines the output once [26], we resolve the unidirectional generation issue by applying another panoptic refinement network that utilizes the whole vectorized floorplan sequence and perform multi-round refinement for the draft floorplan.

## 3 Methodology

Given a bubble diagram input from the user which encodes the number of rooms, room types, as well as their connections through doors, our proposed framework generates vectorized floorplans in two stages. In the draft stage, we employ a GCN for the layout coordinate embeddings to encode the connectivity relationship between rooms and their connecting doors. The GCN-processed embeddings

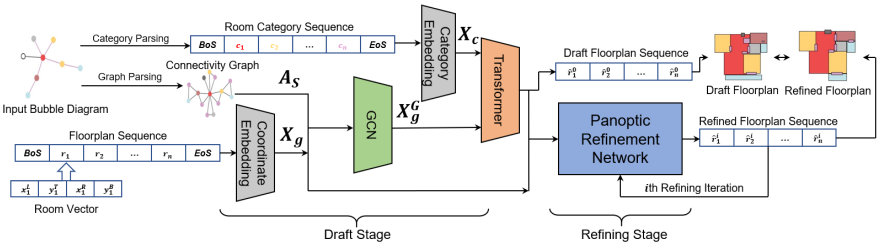



Fig. 2: Overview of our method. User inputs are first processed by a GCN and room elements are generated by a transformer in the *draft stage*. The layouts are further refined by a panoptic refinement network in the *refining stage*.

as well as the category embeddings are then passed into an autoregressive transformer network to generate room layout elements. In the generated floorplan, each room or door is represented by a set of bounding box coordinates. In the refining stage, we apply another transformer network to integrate the global information from the entire floorplan sequence to refine the layouts of the initial floorplan generated in the draft stage. An overview of the proposed framework can be found in Fig. 2. More details are introduced in the remaining part of this section and implementation details can be found in the supplementary materials.

### 3.1 Draft stage with GCN-constrained autoregressive transformer

**Layout representation** Previous layout generation works [7,15] represented each object with 5 attributes $(c, x, y, w, h)$, where $c \in C$ is the category, $(x, y)$ is the center location and $(w, h)$ is the object size. We adopt a similar setting but with the exception of $c$ since in our work, $c$ is given as part of the input conditions encoded in the input bubble diagram therefore there is no need to generate $c$. $c$ can be directly obtained for each room from the input, where doors are treated as special cases of rooms. As illustrated in Fig. 2, we use the geometric representation $(x^L, y^T, x^R, y^B)$ for an object, where $(x^L, y^T)$ is the top-left corner and $(x^R, y^B)$ is the bottom-right corner. We then discretize the continuous $x$- and $y$-coordinates with 8-bit uniform quantization (from 0 to 255) [7,16] for the ease of representation. Following the convention of autoregressive generation [7], we include an additional starting token and an ending token for the layout sequence. We flatten the whole layout with a sequence of 1-D vectors as:

$$S = [\langle BoS \rangle, \boldsymbol{r}_1, \boldsymbol{r}_2, ..., \boldsymbol{r}_N, \langle EoS \rangle], \tag{1}$$

where $N$ is the total number of rooms and doors, $\langle BoS \rangle$ and $\langle EoS \rangle$ are the starting and ending token of the sequence, respectively. Room or door vector is represented by $\boldsymbol{r}_i = [x_i^L, y_i^T, x_i^R, y_i^B]$. Geometric coordinate embedding $X_g$ is learned through a learnable embedding layer which projects the original quantized coordinates to an embedding space.

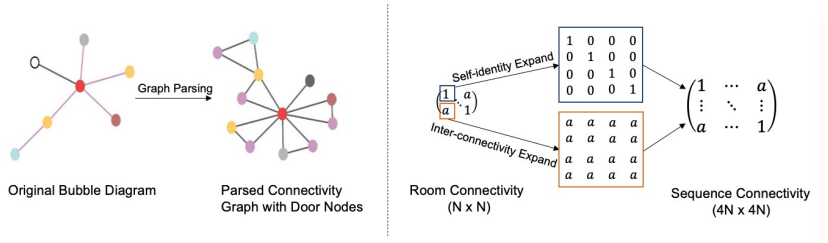



Fig. 3: **Left**: the process of parsing an original bubble diagram to a connectivity graph adapted for GCN. **Right**: the process of transforming the room connectivity matrix $A$ into the sequence connectivity matrix $A_S$.

Also, the room categories corresponding to the objects in the layout sequence are encoded in a separate room category sequence. Following [7], we encode the categories $c$ as discrete values in the range of $[256, 256+N_c]$, where $N_c$ is the number of room categories. Then, a category embedding $X_c$ is also learned through a learnable embedding layer sharing weights with the geometric embedding layer to convert quantized category values to the embedding space.

**Encoding connectivity graph with GCN** To encode the connectivity relationships, we build a connectivity graph $A_S$ which considers the connectivity between rooms via doors from the input bubble diagram. Throughout the paper, we treat all doors as special cases of rooms, as the nodes of the input graph. As illustrated in the left side of Fig. 3, two nodes are parsed as connected by an edge if there are two rooms connected via a shared door or if a room and a door are connected. Inspired by the success of GCN [14] in various tasks, we propose to encode the connectivity graph from the bubble diagram with a GCN. The GCN takes the input geometric embedding $X_g$ as well as a pre-computed sequence connectivity matrix $A_S$ as input. Let $\tilde{A}_S = A_S + I$, where $I$ is the identity matrix indicating that each room is connected to itself, we adopt a learnable layer $W$ to get the updated feature representation $X_g^G$ using the row-normalized connectivity matrix $\hat{A}_S$ as:

$$X_g^G = f(X_g, A_S) = \hat{A}_S X_g W, \quad \hat{A}_S = \tilde{D}^{-\frac{1}{2}} \tilde{A}_S \tilde{D}^{-\frac{1}{2}}, \quad \tilde{D}_{ii} = \sum\nolimits_j \tilde{A}_{S_{ij}}. \tag{2}$$

Now we present how to generate the sequence connectivity matrix $A_S$ as in the right side of Fig. 3. For efficient processing by GCN, we adapt the original room connectivity matrix $A \in N \times N$, where $N$ is the room number, to a sequence connectivity matrix $A_S$ for the vectorized floorplan sequence. We then flatten the quantified $N \times 4$ bounding boxes to a sequence of 1-D vectors with length $4N + 2$, including special tokens $\langle BoS \rangle$ and $\langle EoS \rangle$. For each element $\boldsymbol{r}_i$, we get its connectivity value with another element $\boldsymbol{r}_j$ by querying $A_{i'j'}$ as $A_{S_{ij}} = A_{i'j'}$, where $i'$ and $j'$ are the room indexes of $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$. For $\langle BoS \rangle$ and $\langle EoS \rangle$, we fox their connectivities to other elements as 0. $A_S$ is used as the input of GCN. Note

that during an intermediate step $t$ of the autoregressive generation process, only the generated sequence embeddings $X_{g_{1:t}}$ and a partial graph $A_{S_{1:t}}$ are used as the input of GCN. Thus, the initial floorplan generated in the draft stage may not fully utilize the information of the entire sequence and neglect certain connectivities between rooms. We discuss how to resolve this issue in Sec. 3.2.

**Autoregressive generation with self-attention transformer** Following previous autoregressive generation works [7,16,20], we generate the floorplan sequence through an iterative process. The joint distribution of autoregressive generation can be represented by the chain rule:

$$p(s_{1:4N+2}) = \prod_{i=1}^{4N+2} p(s_i|s_{1:i-1}). \tag{3}$$

A transformer network is applied for autoregressive generation, with a teacher-forcing strategy during training. For the input, we have obtained the GCN-processed geometric embedding $X_g^G$ and the category embedding $X_c$ for each element. To help encode the geometric position of each room inside the floorplan sequence, we also build another embedding layer to learn the positional embeddings $X_p$ [23] for each room element. The input of the transformer is the summation of the three types of embeddings: $X = X_g^G + X_c + X_p$. At the test time, student-forcing is used, and only the starting token is used as input to the transformer network. For the transformer architecture, we apply multi-head self-attention as in LayoutTransformer [7] and use GPT-1 [21] as our backbone transformer network. Note that our model is flexible and also compatible with other transformer models.

### 3.2 Refining stage with panoptic refinement network

A major limitation of transformer-based autoregressive generation is that, for each generation step, only previous states can be taken as input. Thus, the model cannot leverage information from the future states, and may neglect the connectivity between certain elements. For example, in vectorized floorplan generation, it is challenging to properly place a room or door even for human experts, if only limited context information is provided. However, if the initial states for all rooms are given, a transformer model can access the panoptic features and has a better sense of their relative positions with corresponding categories. This motivates us to design a panoptic refining network to improve the floorplan generated in the draft stage by encouraging correct global connectivities.

In the refining stage, we introduce a novel panoptic refinement network (PRN) to refine the initial generations from the autoregressive transformer. The network is also implemented as a GCN and a transformer network with multi-head self-attention modules. Although the network architectures of the two stages are similar, their learning targets are fundamentally different. In the draft stage, our model aims to predict one element at a time given all previously

generated rooms; while in the multi-round refining stage, the PRN aims to learn the refined elements based on the entire sequence.

During training, we use greedy decoding to convert the learned probabilities into the coordinate sequence with discrete values. Note that we do not include $\langle BoS \rangle$ and $\langle EoS \rangle$ in the input of PRN. For the input of the refinement network, since the category embeddings and positional embeddings remain unchanged, we directly get them from the input of the draft stage. Only the geometric embeddings will be updated, which is denoted as $X_{gr}$. The refinement GCN processes $X_{gr}$ and obtain the GCN-updated embeddings $X_{gr}^{G}$. Then the input embeddings $X_r$ for PRN transformer can be represented in a similar manner as the draft stage: $X_r = X_{gr}^{G} + X_c + X_p$.

Finally, the embedding sequence is passed into the PRN transformer to get the refined floorplan sequences. The end-to-end refinement process of PRN can be performed iteratively by taking the refined output from iteration $i$ as the input of the next iteration $i+1$. More analysis is given in Sec. 4.4.

### 3.3 Training strategies and losses

**Hybrid sorting strategy** As shown in previous works [16], ordering is important for training autoregressive models. Layout generation methods [7,16] often sorted the elements by the relative spatial positions. However, this strategy used in unconditional generation cannot be directly applied to category-conditioned generation. Since conditional categories are provided as constraints, a category needs to be strictly associated with a generated element. Thus, we propose a hybrid sorting strategy for a more effective conditional autoregressive generation in both training and testing. To leverage categorical spatial information, we first compute the average spatial position for each room category throughout the whole training set, then sort room categories by their average positions and place two types of doors (*i.e.*, interior and front doors) at the end.

**Reconstruction loss** We apply the standard cross-entropy loss on the generations from both stages to learn a categorical distribution for the quantized coordinates. The loss is averaged by the sequence length $T$ and the refinement iterations $N_r$, which can be represented as:

$$L_{recon} = \frac{1}{T}\sum_{t=1}^{T} L_{ar} + \frac{1}{TN_r}\sum_{t=1}^{T}\sum_{i=1}^{N_r} L_{ref}, \tag{4}$$

where $L_{ar}$ denotes for the loss from autoregressive model and $L_{ref}$ denotes for the loss from panoptic refinement model.

**Geometric loss** Since the reconstruction loss focuses more on single elements, the connectivity between rooms is not explicitly measured by the reconstruction loss. To further encourage correct room layout and connectivity, especially for the refining stage, we introduce a global geometric constraint by calculating the

intersection-over-union (IoU) values between the generated and real layouts. To calculate the IoU values, we need to first transform the predicted probabilities on the quantized coordinates to continuous values. However, direct conversion is not differentiable and thus cannot be trained via back-propagation. To enable end-to-end training, we borrow ideas from the differentiable soft-argmax strategy originally proposed in the stereo matching task [12]. The $i$th element's prediction $x_i$ is then represented as $x_i = \sum_{j=1}^{N} x_j \cdot \sigma(x_j)$, where $\sigma(x_j)$ is the probability of quantized coordinate $x_j$ implemented with the softmax function. Now we can compute a geometric IoU matrix $\mathcal{G}_{gen}$ for the generated sequence and build a geometric loss $L_{geo}$ measuring the $L_1$ distance between $\mathcal{G}_{gen}$ and its groundtruth geometric IoU matrix $\mathcal{G}_{gt}$:

$$L_{geo} = \frac{1}{N_l} \sum_{i_l=1}^{N_l} ||\mathcal{G}_{gen}^{i_l} - \mathcal{G}_{gt}^{i_l}||_1 \ , \tag{5}$$

where $i_l$ and $N_l$ are the index and the total number of pair-wise objects in a floorplan, respectively. $L_{geo}$ is applied in both draft stage and refining stage. Our final loss objective is the combination of two losses without scaling factors.

## 4 Experiments

### 4.1 Dataset and pre-processing

Similar to [18,20], we conducted experiments on RPLAN dataset [24] which provides vector-graphic floorplans from real-world residential buildings. We followed the pre-processing pipeline used in HouseGAN++ with a modification for bounding-box (bbox) vectorized floorplan generation. When we transformed the non-standard polygons into rectangular bounding boxes, we found some cases where frontal doors or interior doors were incorrectly kept inside a room (see Fig. 1 in supplementary materials as an example). To filter out such noisy cases, we computed the overlapping between doors and rooms, then discarded a sample if its frontal doors overlapped with any rooms or any of its interior doors overlapped with any non-living rooms over a threshold $\tau = 50\%$.

### 4.2 Baselines and evaluation metrics

We compared our method with two state-of-the-art floorplan generation methods using the same input setting [17,18] as ours. We also compared with LayoutTransformer [7] which we adopted as the backbone generation network. For fair comparisons, we retrained all baseline methods with their official implementations and applied the same data pre-processing as described in Sec. 4.1.

Following [17,18] which also took bubble diagram as input constraints, we evaluated our method in terms of realism, diversity, and compatibility. For realism, we invited 10 respondents with architecture design expertise (professors and graduate students from the architecture department) and 7 amateurs to

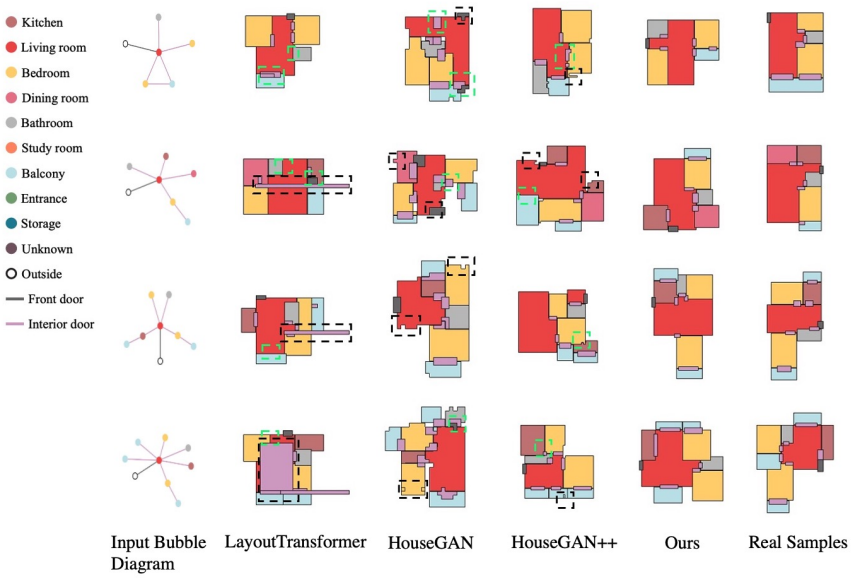



Fig. 4: Sampled floorplan generations from different methods, including real floorplans and their corresponding input bubble diagrams. We use the same color legend as in [18]. Areas with salient connectivity issues and visual artifacts are highlighted with green and **black** dashed boxes, respectively.

give rankings on 100 generation examples based on visual quality and functionality. Our method and two other competing methods [18,7] each generated 100 floorplans and respondents ranked the methods after reviewing each set of 3 generated floorplans that correspond to the same input bubble diagram. A method obtained a score $+1$ for being ranked as $1st$, 0 for $2nd$, and $-1$ for $3rd$. Diversity was computed as the FID score [8] between the generated floorplans from a method and the real floorplans, to assess the diversity of the generated floorplans given the same input. The compatibility [18] validates the connectivity correctness of a generated floorplan by computing the graph edit distance (GED) [1] between the input graph with the reconstructed graph from the generated floorplan. For a generated plan to achieve a high compatibility score, a door must be correctly placed between two rooms (or between a room and the outside area). We generated $1,000$ testing samples on each data split and ran each model for 5 rounds to get the mean and standard deviation of FID and GED scores.

For a comprehensive comparison, we evaluated our methods under both the *separate training* and *mixed training* settings. Under the separate training setting [18], floorplans were split into training and testing sets based on different numbers of rooms. Training was done using any number of rooms except the number of rooms reserved for testing. Comparisons using the separate training are provided in Table 2. In addition to the separate training, we also reported

Table 1: Quantitative comparison between methods using the mixed training. Diversity and compatibility metrics are reported as "mean/std.".

| Model | Visual Realism ↑ | Functional Realism ↑ | Diversity ↓ | Compatibility ↓ |
|---|---|---|---|---|
| LayoutTransformer [7] | -0.58 | -0.57 | **12.3/0.2** | 4.6/0.0 |
| HouseGAN++ [18] | 0.12 | 0.09 | 15.7/0.2 | 3.5/0.0 |
| Ours | **0.46** | **0.48** | 13.8/0.4 | **2.8/0.1** |

Table 2: Quantitative comparison between different methods using the separate training. Integers refer to train on floorplans with other number of rooms and test with the designated number of room.

| Model | Diversity ↓ | | | | Compatibility ↓ | | | |
|---|---|---|---|---|---|---|---|---|
| | 5 | 6 | 7 | 8 | 5 | 6 | 7 | 8 |
| LayoutTransformer [7] | 21.5/0.3 | 27.8/0.5 | 25.6/0.6 | 28.4/0.8 | 2.4/0.1 | 4.0/0.1 | 5.6/0.1 | 6.4/0.1 |
| HouseGAN [17] | 57.6/1.3 | 70.1/1.2 | 61.7/0.9 | 57.7/0.6 | 3.5/1.6 | 4.3/1.7 | 5.2/1.9 | 6.1/1.9 |
| HouseGAN++ [18] | 24.1/0.6 | 17.7/0.7 | 15.0/0.7 | 22.2/0.4 | 1.6/0.0 | 2.5/0.1 | **2.8/0.0** | **3.8/0.0** |
| Ours | **15.4/0.5** | **16.4/0.5** | **14.6/0.4** | **14.1/0.4** | **1.3/0.0** | **1.9/0.0** | 2.9/0.1 | 4.6/0.0 |

metrics calculated using the *mixed training*. Specifically, we mixed floorplans with different numbers of rooms together and split them into training and testing without overlapping in input bubble diagrams. We believe this is a more general setting since floorplans with different number of rooms should all be available for training. Comparisons using mixed training are illustrated in Table 1.

### 4.3 Result Analysis

Table 2 shows the quantitative comparisons among different methods under the separate training setting. Our method outperforms all the previous methods on diversity by a large margin. For compatibility, our model is significantly better than existing approaches on the split containing 5 or 6 rooms, and is comparable with HouseGAN++ [18] on the 7 room split. Under our proposed mixed training setting in Table 1, which we believe is more suitable for realistic scenarios, our method clearly outperforms HouseGAN++ on all metrics. Note that evaluation of visual and functional realism by ten architecture experts and seven amateurs was conducted using floorplans generated in the mixed training setting, and the result on realism comparison is reported in Table 1. One can see that, for realism, our method outperforms [7,18] on both visual quality and functionality by a large margin. Qualitatively, from the samples shown in Fig. 4, one can see that LayoutTransformer [7] often fails to generate doors correctly, especially when the input bubble diagrams are complex. The rooms generated by HouseGAN [17] have salient artifacts due to the limitations of rasterized representation. There are fewer artifacts in floorplans generated by HouseGAN++ [18], but non-straight room boundaries still exist. Another non-negligible drawback of rasterized representation is that several disconnected regions could be generated for one specific room (*e.g.*, the living room of the $3rd$ case by HouseGAN++, Fig. 4). In comparison, our method only generates objects as connected components due to the advantage of vectorized representation. Furthermore, as shown

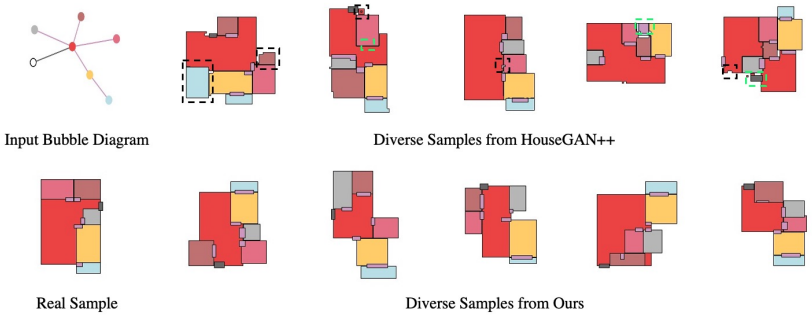



Fig. 5: Diversity comparison between HouseGAN++ and ours. Salient connectivity issues and visual artifacts are highlighted with green and **black** boxes.

Table 3: Ablation study of our proposed method. Base refers to the backbone GPT-1 [21] transformer network. HS, PR, GL represent our proposed hybrid sorting, panoptic refinement, geometric loss, respectively.

**(a)** Our proposed components.

| Components | | | | | Diversity (↓) | Compatibility (↓) |
|---|---|---|---|---|---|---|
| Base | HS | GCN | PR | GL | 6 | 6 |
| ✓ | | | | | 27.8/0.5 | 4.0/0.1 |
| ✓ | ✓ | | | | 29.8/0.6 | 3.6/0.1 |
| ✓ | ✓ | ✓ | | | 14.5/0.2 | 3.7/0.0 |
| ✓ | ✓ | | ✓ | | **14.0/0.3** | 3.4/0.1 |
| ✓ | ✓ | ✓ | ✓ | | 18.3/0.2 | 2.1/0.1 |
| ✓ | ✓ | ✓ | ✓ | ✓ | 15.8/0.5 | **1.9/0.0** |

**(b)** Refinement iterations of PRN.

| # of iteration | Diversity (↓) | Compatibility (↓) |
|---|---|---|
| | 6 | 6 |
| 1-iteration | 15.1/0.8 | 2.7/0.0 |
| 3-iterations | **14.4/0.3** | 2.2/0.0 |
| 5-iterations | 15.8/0.5 | **1.9/0.0** |
| 7-iterations | 16.9/0.4 | 2.1/0.1 |

in Fig. 5, our model better keeps the fidelity and connectivity while generating diverse samples than the state-of-the-art HouseGAN++ [18].

### 4.4 Ablation study

**Proposed components.** We conducted an ablation study on each of our proposed components, including the hybrid sorting (HS) strategy, the use of GCN to encode connectivity graph, the proposed panoptic refinement (PR) network, and the geometric loss (GL). Specifically, we conducted ablation experiments on the 6 room generation split. The base model corresponds to the result of LayoutTransformer [7], which we used as the unconditional backbone network. As shown in Table 3a, applying the hybrid sorting strategy improved the compatibility compared with the base model. Adding GCN or refinement model independently upon the autoregressive model significantly improved the diversity, but had little benefits for compatibility. A possible reason is that, passing a partial graph containing previously generated elements to GCN can smooth the feature for past elements, but is not beneficial to the generation for future elements. Applying the two modules simultaneously greatly improved compatibility

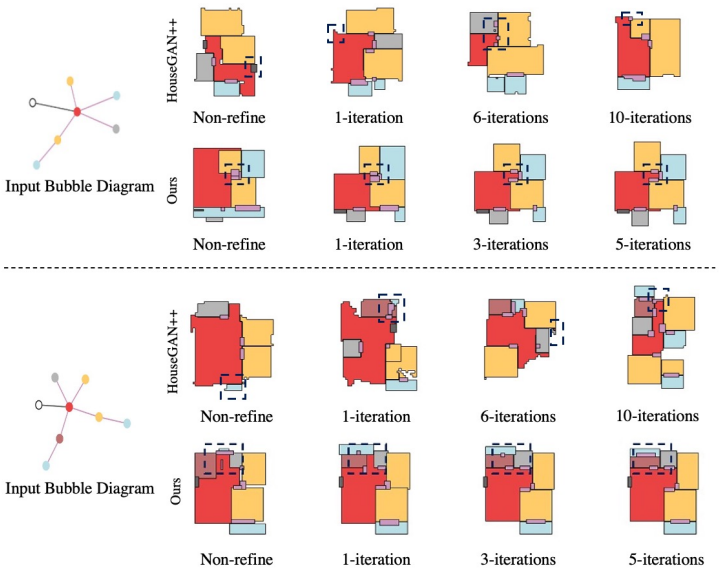



Fig. 6: Floorplan refinement between ours and [18]. Salient differences across different iterations are highlighted with **black** dashed boxes.

since a complete connectivity graph drives GCN to capture the connectivity relationship of the whole sequence, and the refinement transformer can further exploit global contexts. With the proposed geometric loss on the generated vectors, both diversity and compatibility got further improved as the correct layouts and room connectivities were encouraged during the training.

**Refinement iterations** We also demonstrate the effect of applying different numbers of refinement iterations. As shown in Table 3b, there were no significant differences in diversity among various iterations. For compatibility, the best result was achieved when we refined the initial generated sequence for 5 iterations. We have tried to refine more iterations, such as $N_r = 7$, but the experiments showed no further improvement, which may indicate that PRN has adequately refined the draft floorplan. We also show some qualitative visualizations in Fig. 6 to help examine the effect of our multi-round refinements.

## 5 Discussion

HouseGAN++ [18] also proposed to refine floorplans iteratively using heuristics and optimizations. Our proposed PRN refinement network is very different, however, since it is an end-to-end model which refines the draft floorplan while keeping correct layouts. As illustrated in Fig. 6, applying one iteration of PRN refinement (*i.e.*, 1-iteration) significantly improved the generation quality

compared with no refinement (*i.e.*, 'Non-refine'). Refining more iterations consistently made details more accurate, such as reducing overlaps, making neighboring objects better aligned, and placing doors more properly. Compared with the floorplans after multiple refinement iterations by HouseGAN++, the floorplans refined by our PRN were more consistent in layout and inherited correct room connectivities acquired in previous iterations of refinement. Meanwhile, refinements by HouseGAN++ introduced more drastic changes across different iterations, and occasionally regions with desirable properties from previous iterations got lost and disappeared in the later iterations.

Although we have achieved superior performance under most settings, our method still has limitations. First, our model represents rooms as rectangular boxes, which may not always be optimal in practice. We try to mitigate the issue by removing overlapping parts between the living room and other rooms, making the generated floorplan more general. An end-to-end generation with vectorized multi-edge polygons representing rooms could be a future direction. Second, we did not achieve better compatibility performance than HouseGAN++ on the split of 7 or 8 rooms, indicating that our method can be further improved for modeling longer floorplan sequences. More advanced GNNs and transformers may perform better in modeling more complex graphs and longer sequences.

During the user study, besides rankings of methods which we used to calculate realism scores, we also collected feedback in the form of free-form comments from architects. Most of the architects agreed that our method has made substantial progress in automated floorplan generation, and can potentially change the way in which initial floorplan designs are created and visualized. They especially liked that methods like ours can easily generate multiple candidate initial designs. However, they also mentioned that certain domain knowledge should be embedded in the generation process, for example, circulation spaces must be smaller, kitchen and dining must be connected, service blocks like washroom should be at the edge of the building. We plan to explore how to incorporate such domain knowledge into the generation process in our future work.

## 6 Conclusion

In this paper, we propose a novel two-stage vectorized floorplan generation framework. Floorplans can be generated and refined in an end-to-end fashion given an input bubble diagram encoding room number, type and connectivity. Compared to previous state-of-the-art floorplan generation methods which generate rasterized floorplan images, our proposed vectorized floorplan generation method generates more realistic and usable floorplan designs. Endorsed by the architect experts in our user study, we believe our method has great potential in computer-aided floorplan design.

**Acknowledgements.** This work is supported in part by NSF Award #1815491. We appreciate the help from professors and graduate students from College of Arts and Architecture at Penn State with the user study. We also would like to thank Enyan Dai for meaningful discussions on GNN.

## References


1. Abu-Aisheh, Z., Raveaux, R., Ramel, J.Y., Martineau, P.: An exact graph edit distance algorithm for solving pattern recognition problems. In: 4th International Conference on Pattern Recognition Applications and Methods 2015 (2015)
2. Arroyo, D.M., Postels, J., Tombari, F.: Variational transformer networks for layout generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13642–13652 (2021)
3. Chen, J., Liu, C., Wu, J., Furukawa, Y.: Floor-sp: Inverse cad for floorplans by sequential room-wise shortest path. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 2661–2670 (2019)
4. Cruz, S., Hutchcroft, W., Li, Y., Khosravan, N., Boyadzhiev, I., Kang, S.B.: Zillow indoor dataset: Annotated floor plans with 360deg panoramas and 3d room layouts. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2133–2143 (2021)
5. Fan, Z., Zhu, L., Li, H., Chen, X., Zhu, S., Tan, P.: Floorplancad: a large-scale cad drawing dataset for panoptic symbol spotting. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 10128–10137 (2021)
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in neural information processing systems **27** (2014)
7. Gupta, K., Lazarow, J., Achille, A., Davis, L.S., Mahadevan, V., Shrivastava, A.: Layouttransformer: Layout generation and completion with self-attention. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1004–1014 (2021)
8. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017)
9. Hu, R., Huang, Z., Tang, Y., Van Kaick, O., Zhang, H., Huang, H.: Graph2plan: Learning floorplan generation from layout graphs. ACM Transactions on Graphics (TOG) **39**(4), 118–1 (2020)
10. Huang, W., Zheng, H.: Architectural drawings recognition and generation through machine learning (2018)
11. Jyothi, A.A., Durand, T., He, J., Sigal, L., Mori, G.: Layoutvae: Stochastic scene layout generation from a label set. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9895–9904 (2019)
12. Kendall, A., Martirosyan, H., Dasgupta, S., Henry, P., Kennedy, R., Bachrach, A., Bry, A.: End-to-end learning of geometry and context for deep stereo regression. In: Proceedings of the IEEE international conference on computer vision. pp. 66–75 (2017)
13. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
14. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
15. Kong, X., Jiang, L., Chang, H., Zhang, H., Hao, Y., Gong, H., Essa, I.: Blt: Bidirectional layout transformer for controllable layout generation. arXiv preprint arXiv:2112.05112 (2021)
16. Nash, C., Ganin, Y., Eslami, S.A., Battaglia, P.: Polygen: An autoregressive generative model of 3d meshes. In: International Conference on Machine Learning. pp. 7220–7229. PMLR (2020)

17. Nauata, N., Chang, K.H., Cheng, C.Y., Mori, G., Furukawa, Y.: House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In: European Conference on Computer Vision. pp. 162–177. Springer (2020)
18. Nauata, N., Hosseini, S., Chang, K.H., Chu, H., Cheng, C.Y., Furukawa, Y.: House-gan++: Generative adversarial layout refinement networks. arXiv preprint arXiv:2103.02574 (2021)
19. Newton, D.: Generative deep learning in architectural design. Technology— Architecture+ Design **3**(2), 176–189 (2019)
20. Para, W., Guerrero, P., Kelly, T., Guibas, L.J., Wonka, P.: Generative layout modeling using constraint graphs. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6690–6700 (2021)
21. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training (2018)
22. Shekhawat, K., Upasani, N., Bisht, S., Jain, R.N.: A tool for computer-generated dimensioned floorplans based on given adjacencies. Automation in Construction **127**, 103718 (2021)
23. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
24. Wu, W., Fu, X.M., Tang, R., Wang, Y., Qi, Y.H., Liu, L.: Data-driven interior plan generation for residential buildings. ACM Transactions on Graphics (TOG) **38**(6), 1–12 (2019)
25. Xu, L., Xiangli, Y., Rao, A., Zhao, N., Dai, B., Liu, Z., Lin, D.: Blockplanner: City block generation with vectorized graph representation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5077–5086 (2021)
26. Yang, C.F., Fan, W.C., Yang, F.E., Wang, Y.C.F.: Layouttransformer: Scene layout generation with conceptual and spatial diversity. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 3732–3741 (2021)