

House-GAN: Relational Generative Adversarial Networks for Graph-constrained House Layout Generation

Nelson Nauata¹, Kai-Hung Chang², Chin-Yi Cheng², Greg Mori¹, and Yasutaka Furukawa¹

¹ Simon Fraser University

{nnauata, furukawa}@sfu.ca, mori@cs.sfu.ca

² Autodesk Research

{kai-hung.chang, chin-yi.cheng}@autodesk.com

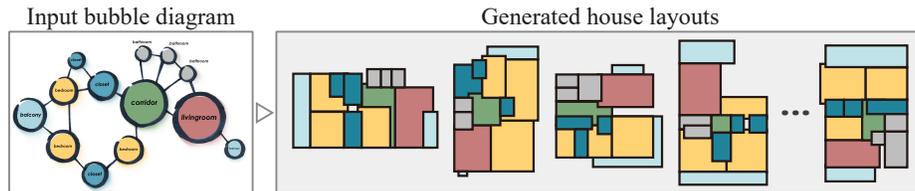


Fig. 1. House-GAN is a novel graph-constrained house layout generator, built upon a relational generative adversarial network. The bubble diagram (graph) is given as an input for automatically generating multiple house layout options.

Abstract. This paper proposes a novel graph-constrained generative adversarial network, whose generator and discriminator are built upon relational architecture. The main idea is to encode the constraint into the graph structure of its relational networks. We have demonstrated the proposed architecture for a new house layout generation problem, whose task is to take an architectural constraint as a graph (i.e., the number and types of rooms with their spatial adjacency) and produce a set of axis-aligned bounding boxes of rooms. We measure the quality of generated house layouts with the three metrics: the realism, the diversity, and the compatibility with the input graph constraint. Our qualitative and quantitative evaluations over 117,000 real floorplan images demonstrate that the proposed approach outperforms existing methods and baselines. We will publicly share all our code and data.

Keywords: GAN, graph-constrained, layout, generation, floorplan

1 Introduction

A house is the most important purchase one might make in life, and we all want to live in a safe, comfortable, and beautiful environment. However, designing

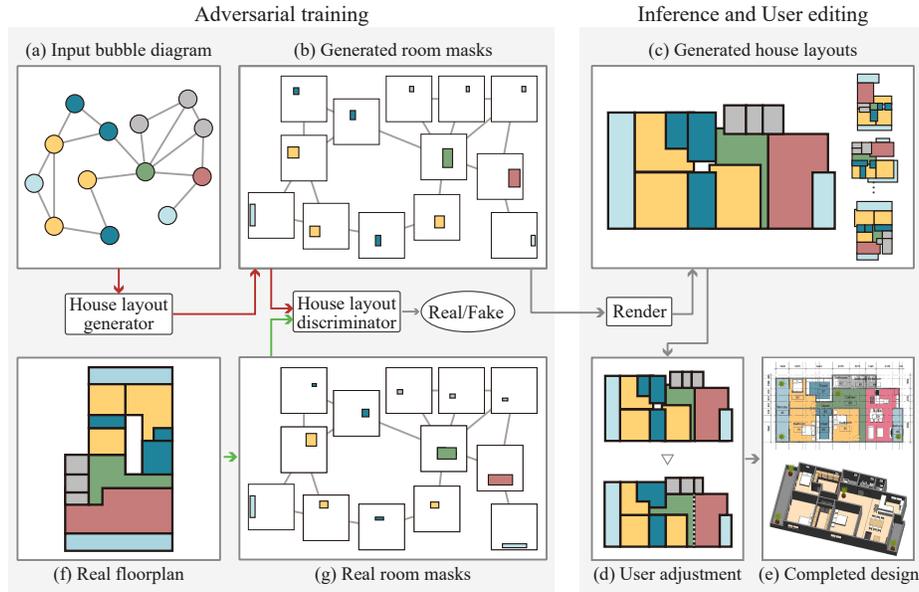


Fig. 2. Floorplan designing workflow with House-GAN. The input to the system is a bubble diagram encoding high-level architectural constraints. House-GAN learns to generate a diverse set of realistic house layouts under the bubble diagram constraint. Architects convert a layout into a real floorplan.

a house that fulfills all the functional requirements with a reasonable budget is challenging. Only a small fraction of the residential building owners have enough budget to employ architects for customized house design.

House design is an expensive and time-consuming iterative process. A standard workflow is to 1) sketch a “bubble diagram” illustrating the number of rooms with their types and connections; 2) produce corresponding floorplans and collect clients feedback; 3) revert to the bubble diagram for refinement, and 4) iterate. Given limited budget and time, architects and their clients often need to compromise on the design quality. Therefore, automated floorplan generation techniques are in critical demand with immense potentials in the architecture, construction, and real-estate industries.

This paper proposes a novel house layout generation problem, whose task is to take a bubble diagram as an input, and generates a diverse set of realistic and compatible house layouts (See Fig. 1). A bubble diagram is represented as a graph where 1) nodes encode rooms with their room types and 2) edges encode their spatial adjacency. A house layout is represented as a set of axis-aligned bounding boxes of rooms (See Fig. 2).

Generative models have seen a breakthrough in Computer Vision with the emergence of generative adversarial networks (GANs) [6], capable of producing realistic human faces [14], street-side images [15]. GAN has also proven effective

for constrained image generation. Image-to-image translation has been an active area of research, where an image is given as a constraint (e.g., generating a zebra image with the same pose from a horse image) [27,11,5]. Realistic scene images are generated given object bounding boxes and placements as the constraint [3].

The house layout generation poses a new challenge: The graph is enforced as a constraint. We present a novel generative model called House-GAN that employs relational generator and discriminator, where the constraint is encoded into the graph structure of their relational neural networks. More specifically, we employ convolutional message passing neural networks (Conv-MPN) [26], which differs from graph convolutional networks (GCNs) [12,3] in that 1) a node represents a room as a feature volume in the design space (as opposed to a 1D latent vector), and 2) convolutions update features in the design space (as opposed to multi-layer perceptron). The architecture enables more effective higher-order reasoning for composing layouts and validating adjacency constraints.

Our qualitative and quantitative evaluations over 117,000 real floorplan images demonstrate that House-GAN is capable of producing more diverse sets of more realistic floorplans that are compatible with the bubble diagram than the other competing methods. We will publicly share all our code and data.

2 Related work

Procedural layout generation: Layout composition has been an active area of research in various domains, including architectural layouts [8,4,20,21], game-level design [9,18] and others. In particular, Peng *et al.* [21] takes a set of deformable room templates and tiles arbitrarily shaped domains subject to the accessibility and aesthetics constraints. Ma *et al.* [18] generates diverse game-level layouts, given a set of 2D polygonal “building blocks” and their connectivity constraints as a graph. These methods are more traditional and based on hand-crafted energy minimization. Our approach exploits powerful data-driven techniques (i.e., deep neural architecture) for more robustness.

Data-driven space planning: Data-driven sequential generative methods have been proposed for indoor scene synthesis by Wuang *et al.* [24] and Ritchie *et al.* [22], indoor plan generation by Wu *et al.* [25], and outdoor scene generation by Jyothi *et al.* [13]. In particular, Wu *et al.* [25] proposes a data-driven method for automatic floorplan generation for residential houses from a building footprint. The method starts from the living-room and sequentially adds rooms via an encoder-decoder network, followed by a final post-processing for the vectorization. Jyothi *et al.* [13] proposes a variational autoencoder (VAE), which iteratively predicts a diverse yet plausible counts and sets of bounding boxes, given a set of object labels as input. Li *et al.* [16] proposes a non-sequential adversarial generative method called LayoutGAN, which has a self-attention mechanism in the generator and a wireframe renderer in the discriminator. These methods produce impressive results but cannot take a graph as an input constraint.

Graph-constrained layout generation: Graph-constrained layout generation has also been a focus of research. Wang *et al.* [23] plans an indoor scene as a relation graph and iteratively inserts a 3D model at each node via convolutional neural network (CNN) guided search. Merrel *et al.* [19] utilizes Bayesian Networks for retrieving candidate bubble diagrams, given high-level conditions such as the number of rooms with room types and approximate square footage. These bubble diagrams are later converted to floorplans using the Metropolis algorithm. Jonhson *et al.* [12] and Ashual *et al.* [3] aim to generate image layouts and synthesize realistic images from input scene-graphs via GCNs. Our innovation is a novel relational generative adversarial network, where the input constraint is encoded into the graph structure of the relational generator and discriminator. The qualitative and quantitative evaluations demonstrate the effectiveness of our approach over all the competing methods.

3 Graph-constrained house layout generation problem

We seek to generate a diverse set of realistic house layouts, compatible with a bubble diagram. The section explains our dataset, metrics, and limitations.

Dataset: LIFULL HOME’s database offers five million real floorplans, from which we retrieved 117,587 [1] and rescaled uniformly to fit inside the 256×256 resolution (See Table 1). The database does not contain bubble diagrams. We used the [floorplan vectorization algorithm](#) [17] to generate the vector-graphics format, which is converted into bubble diagrams. A bubble diagram is a graph, where a node is a room with a room type as its property.³ Two rooms are connected if the Manhattan distance between the bounding boxes is less than 8 pixels. An output house layout is axis-aligned bounding boxes (See Fig. 3).

Metrics: We divide the samples into five groups based on the number of rooms: (1-3, 4-6, 7-9, 10-12, and 13+). To test the generalization capability, we conduct k-fold validation (k=5): When generating layouts in a group, we train a model while excluding samples in the same group so that a method cannot simply memorize. At test time, we randomly pick a house layout and generate X samples. $X = 10$ for measuring the realism and diversity, and $X = 1$ for measuring the compatibility whose evaluation is computationally expensive.

- The realism is measured by an average user rating. We present a generated house layout against a ground-truth or another method. A subject puts one of the four ratings: better (+1), worse (-1), equally-good (+1), or equally-bad (-1).
- The diversity is measured by the FID score [10] with the rasterized layout images. We rasterize a layout by 1) Setting the background to white; 2) Sorting the rooms in the decreasing order of the areas; and 3) Painting each room with a color based on its room type (e.g., orange for a bedroom) as shown in Figure 3.

³ Room types are “living room”, “kitchen”, “bedroom”, “bathroom”, “closet”, “balcony”, “corridor”, “dining room”, “laundry room”, or “unkown”.

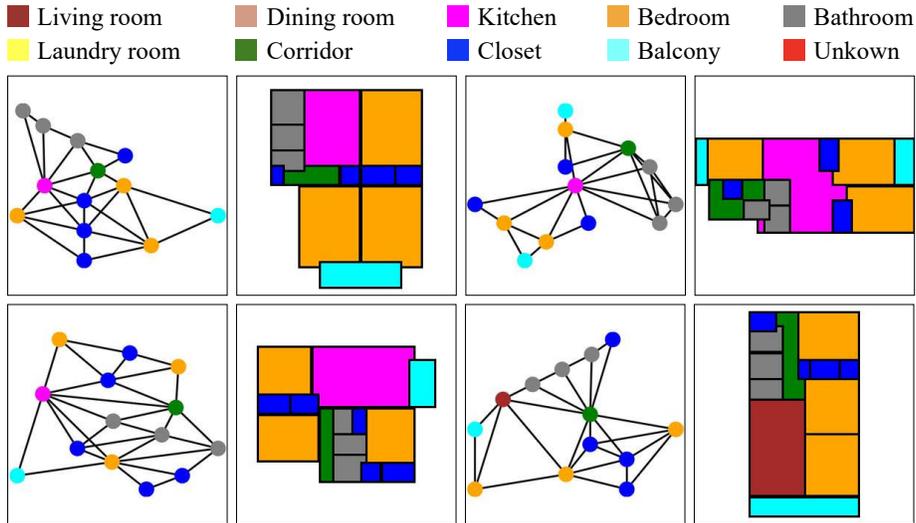


Fig. 3. Sample bubble diagrams and house layouts in our dataset.

Table 1. We divide the samples into five groups based on the room counts (1-3, 4-6, 7-9, 10-12, and 13+). The second row shows the numbers of samples. The remaining rows show the average number of rooms (left) and the average number of edge connections per room (right). The room counts are small for living-rooms, which are often interchangeable with dining-rooms and kitchens in the Japanese real-estate market.

	1-3	4-6	7-9	10-12	13+	All
(# of samples)	7,393	28,170	42,635	30,625	8,764	117,587
Living Room	0.0/1.2	0.0/3.1	0.1/4.8	0.3/5.6	0.3/5.9	0.1/5.1
Kitchen	0.6/1.3	1.0/3.3	1.2/4.5	1.1/5.4	1.3/5.3	1.1/4.4
Bedroom	0.4/1.3	0.8/2.8	1.3/3.5	2.0/3.9	2.8/4.1	1.4/3.6
Bathroom	0.7/1.2	1.6/2.4	2.6/2.9	3.0/3.3	3.4/3.5	2.4/3.0
Closet	0.3/1.2	1.0/2.2	1.6/2.6	2.4/3.1	3.6/3.2	1.7/2.8
Balcony	0.2/0.9	0.6/1.2	0.9/1.5	1.0/1.9	1.3/2.0	0.8/1.6
Corridor	0.1/1.1	0.1/2.6	0.4/3.7	1.0/4.6	1.4/5.0	0.5/4.3
Dining Room	0.0/1.5	0.0/3.0	0.0/3.6	0.0/3.2	0.0/1.9	0.0/2.9
Laundry Room	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0
Unknown	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0

- The compatibility with the bubble diagram is a graph editing distance [2] between the input bubble diagram and the bubble diagram constructed from the output layout in the same way as the GT preparation above.

Assumptions: In contrast to the real design process, we make a few restrictive assumptions to simplify the problem setting: 1) A node property does not have a

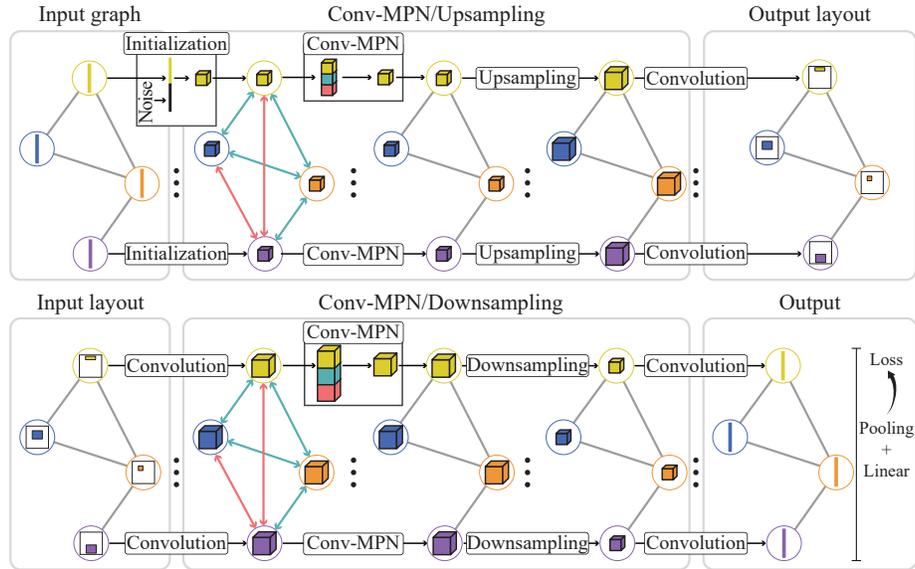


Fig. 4. Relational house layout generator (top) and discriminator (bottom). Conv-MPN is our backbone architecture [26]. The input graph constraint is encoded into the graph structure of their relational networks.

room size; 2) A room shape is always a rectangle; and 3) An edge property (i.e., room adjacency) does not reflect the presence of doors. This is the first research step in tackling the problem, where these extensions are our future work.

4 House-GAN

House-GAN is a relational generative adversarial network. The key specialization is our relational generator and discriminator, where the input graph constraint is encoded into the graph structure of the relational networks. In particular, we employ Conv-MPN [26], which differs from GCNs [3,12] in that a node stores a feature volume and convolutions update features in the design space (as opposed to a 1D latent vector space).

4.1 House layout generator

The generator takes a noise vector per room and a bubble diagram, then generates a house layout as an axis-aligned rectangle per room. The bubble diagram is represented as a graph, where a node represents a room with a room type, and an edge represents the spatial adjacency. More specifically, a rectangle should be generated for each room, and two rooms with an edge must be spatially adjacent (i.e., their Manhattan distance should be less than 8 pixels). We now explain

the three phases of the generation process (See Fig. 4). The full architectural specification is shown in Table 2.

Input graph: Given a bubble diagram, we form Conv-MPN whose relational graph structure is the same as the bubble diagram. We generate a node for each room and initialize with a 128-d noise vector sampled from a normal distribution, concatenated with a 10-d room type vector \vec{t}_r (one-hot encoding). r is a room index. This results in a 138-d vector \vec{g}_r :

$$\vec{g}_r \leftarrow \left\{ \mathbb{N}(0, 1)^{128}; \vec{t}_r \right\}. \quad (1)$$

Conv-MPN stores features as a 3D tensor in the output design space. We apply a shared linear layer for expanding \vec{g}_r into a $(8 \times 8 \times 16)$ feature volume $\mathbf{g}_r^{l=1}$. ($l = 1$) denotes that the feature is for the first Conv-MPN module, which will be upsampled twice to become a $(32 \times 32 \times 16)$ feature volume $\mathbf{g}_r^{l=3}$ later.

Conv-MPN/Upsampling: Conv-MPN module updates a graph of room-wise feature volumes via convolutional message passing [26]. More precisely, we update \mathbf{g}_r^l by 1) concatenating a sum-pooled feature across rooms that are connected in the graph; 2) concatenating a sum-pooled feature across non-connected rooms; and 3) applying a CNN:

$$\mathbf{g}_r^l \leftarrow \text{CNN} \left[\mathbf{g}_r^l \ ; \ \text{Pool}_{s \in \mathbf{N}(r)} \mathbf{g}_s^l \ ; \ \text{Pool}_{s \in \bar{\mathbf{N}}(r)} \mathbf{g}_s^l \right]. \quad (2)$$

$\mathbf{N}(r)$ and $\bar{\mathbf{N}}(r)$ denote sets of rooms that are connected and not-connected, respectively. We upsample features by a factor of 2 using a transposed convolution (kernel=4, stride=2, padding=1), while maintaining the number of channels. The generator has two rounds of Conv-MPN and upsampling, making the final feature volume $\mathbf{g}_r^{l=3}$ of size $(32 \times 32 \times 16)$.

Output layout: A shared three-layer CNN converts a feature volume into a room segmentation mask of size $(32 \times 32 \times 1)$. This graph of segmentation masks will be passed to the discriminator during training. At test time, the room mask (an output of tanh function with the range $[-1, 1]$) is thresholded at 0.0, and we fit the tightest axis-aligned rectangle for each room to generate the house layout.

4.2 House layout discriminator

The discriminator performs a sequence of operations in the reverse order. The input is a graph of room segmentation masks either from the generator (before rectangle fitting) or a real floorplan (1.0 for foreground and -1.0 for background). A segmentation mask is of size $32 \times 32 \times 1$. To associate the room type information, we take a 10-d room type vector, apply a linear layer to expand to 8192-d, then reshape to a $(32 \times 32 \times 8)$ tensor, which is concatenated to the segmentation mask. A shared three-layer CNN converts the feature into a size $(32 \times 32 \times 16)$, followed by two rounds of Conv-MPN and downsampling. We downsample by a

factor of 2 each time by a convolution layer (kernel=3, stride=2, padding=1). Lastly, we use a three layer CNN for converting a room feature into a 128-d vector (\vec{d}_r). We sum-pool over all the room vectors and add a single linear layer to output a scalar \mathbf{d} , classifying ground-truth samples from generated ones.

$$\mathbf{d} \leftarrow \text{Linear}(\text{Pool}_r \vec{d}_r) \quad (3)$$

We use the WGAN-GP [7] loss with gradient penalty set to 10. We compute the gradient penalty as proposed by Gulrajani *et al.*[7]: Linearly and uniformly interpolating room segmentation masks pixel-wise between real samples and generated ones, while fixing the relational graph structure.

5 Implementation Details

We have implemented the proposed architecture in PyTorch and utilized a workstation with dual Xeon CPUs and dual NVIDIA Titan RTX GPUs. Our model adopts WGAN-GP [7] with ADAM optimizer ($b_1 = 0.5$, $b_2 = 0.999$) and is trained for 200k iterations. The learning rates of the generator and the discriminator are 0.0001, respectively. The batch size is 32. We set the number of critics to 1 and use leaky-ReLUs ($\alpha=0.1$) for all non-linearities except for the last one in the generator where we use hyperbolic tangent. We tried but do not use spectral normalization in the convolution layers and a per-room discriminator (before final sum-pooling), which did not lead to significant improvements. See Table 2 for the full architectural specification.

6 Experimental Results

Realism, diversity, and compatibility metrics evaluate the performance of the proposed system against the two baselines and the two competing methods. We first introduce these methods, while referring to the supplementary document for the full architectural specification.

- **CNN-only:** We encode the bubble diagram into a fixed dimensional vector by assuming at most 40 rooms and sorting the rooms based on the room-center x-coordinate in the corresponding floorplan. To be precise, we concatenate a 128-d noise vector, a 10-d room type vector for 40 rooms, and $780 = \binom{40}{2}$ dimensional vector indicating the room connectivity, resulting in a 1308-d vector. We pad zeros for missing rooms. We convert a vector into a feature volume and apply two rounds of upsampling and CNN to produce room masks as a $(32 \times 32 \times 40)$ feature volume. The discriminator takes the room masks, concatenates the room type and connectivity information (i.e. 1308-d vector) represented as a $(32 \times 32 \times 8)$ feature volume and performs an inverse operation of the generator.
- **GCN:** The generator takes a 128-d noise vector concatenated with a 10-d room type vector per room. After 2 rounds of message passing as 1d vectors by

Table 2. House-GAN architectural specification. “s” and “p” denote stride and padding. “x”, “z” and “t” denote the room mask, noise vector, and room type vector. “conv_mpn” layers have the same architecture in all occurrences. Convolution kernels and layer dimensions are specified as $(N_{in} \times N_{out} \times W \times H)$ and $(W \times H \times C)$.

Architecture	Layer	Specification	Output Size
	$concat(z, t)$	N/A	1×138
	$linear_reshape_1$	138×1024	$8 \times 8 \times 16$
	$conv_mpn_1$	$\begin{bmatrix} 16 \times 16 \times 3 \times 3, (s = 1, p = 1) \\ 16 \times 16 \times 3 \times 3, (s = 1, p = 1) \\ 16 \times 16 \times 3 \times 3, (s = 1, p = 1) \end{bmatrix}$	$8 \times 8 \times 16$
House layout generator	$upsample_1$	$16 \times 16 \times 4 \times 4, (s = 2, p = 1)$	$16 \times 16 \times 16$
	$conv_mpn_2$	-	$16 \times 16 \times 16$
	$upsample_2$	$16 \times 16 \times 4 \times 4, (s = 2, p = 1)$	$32 \times 32 \times 16$
	$conv_leaky_relu_1$	$16 \times 256 \times 3 \times 3, (s = 1, p = 1)$	$32 \times 32 \times 256$
	$conv_leaky_relu_2$	$256 \times 128 \times 3 \times 3, (s = 1, p = 1)$	$32 \times 32 \times 128$
	$conv_tanh_1$	$128 \times 1 \times 3 \times 3, (s = 1, p = 1)$	$32 \times 32 \times 1$
	$linear_reshape_1(t)$	10×8192	$32 \times 32 \times 8$
	$concat(t, x)$	N/A	$32 \times 32 \times 9$
	$conv_leaky_relu_1$	$9 \times 16 \times 3 \times 3, (s = 1, p = 1)$	$32 \times 32 \times 16$
	$conv_leaky_relu_2$	$16 \times 16 \times 3 \times 3, (s = 1, p = 1)$	$32 \times 32 \times 16$
	$conv_leaky_relu_3$	$16 \times 16 \times 3 \times 3, (s = 1, p = 1)$	$32 \times 32 \times 16$
	$conv_mpn_1$	-	$32 \times 32 \times 16$
House layout discriminator	$downsample_1$	$16 \times 16 \times 3 \times 3, (s = 2, p = 1)$	$16 \times 16 \times 16$
	$conv_mpn_2$	-	$16 \times 16 \times 16$
	$downsample_2$	$16 \times 16 \times 3 \times 3, (s = 2, p = 1)$	$8 \times 8 \times 16$
	$conv_leaky_relu_1$	$16 \times 256 \times 3 \times 3, (s = 2, p = 1)$	$4 \times 4 \times 256$
	$conv_leaky_relu_2$	$256 \times 128 \times 3 \times 3, (s = 2, p = 1)$	$2 \times 2 \times 128$
	$conv_leaky_relu_3$	$128 \times 128 \times 3 \times 3, (s = 2, p = 1)$	$1 \times 1 \times 128$
	$pool_reshape_linear_1$	128×1	1

GCN, a shared CNN module decodes the vector into a mask. The discriminator merges the room segmentation and type into a feature volume as in House-GAN. A shared CNN encoder converts it into a feature vector, followed by 2 rounds of message passing, sum-pooling, and a linear layer to produce a scalar.

Table 3. The main quantitative evaluations. Realism is measured by a user study with graduate students and professional architects. Diversity is measured by the FID scores. Compatibility is measured by the graph edit distance. (\uparrow) and (\downarrow) indicate the higher-the-better and the lower-the-better metrics, respectively. We compare HouseGAN against two baselines and two competing methods. The cyan, orange, and magenta colors indicate the first, the second, and third best results, respectively.

Model	Realism (\uparrow)	Diversity (\downarrow)					Compatibility (\downarrow)				
	All groups	1-3	4-6	7-9	10-12	13+	1-3	4-6	7-9	10-12	13+
CNN-only	-0.49	13.2	26.6	43.6	54.6	90.0	0.4	3.1	8.1	15.8	34.7
GCN	0.11	18.6	17.0	18.1	22.7	31.5	0.1	0.8	2.3	3.2	3.7
Ashual <i>et al.</i> [3]	-0.61	64.0	92.2	87.6	122.8	149.9	0.2	2.7	6.2	19.2	36.0
Johnson <i>et al.</i> [12]	-0.62	69.8	86.9	80.1	117.5	123.2	0.2	2.6	5.2	17.5	29.3
House-GAN [Ours]	0.15	13.6	9.4	14.4	11.6	20.1	0.1	1.1	2.9	3.9	10.8

• **Ashual *et al.* [3]** and **Johnson *et al.* [12]**: After converting our bubble diagram and floorplan data into their representation, we use their official code to train the models with two minor adaptations: 1) we limit scene-graphs to contain only two types of connections: “adjacent” and “not adjacent”; 2) we provide the rendered bounding boxes filled with their corresponding color during training.

Table 3 shows our main results. As explained in Sect. 3, we divide 117,587 samples into 5 groups based on their room counts. For the generation of layouts in each group, we exclude samples in the same group from the training so that methods cannot simply memorize layouts. HouseGAN outperforms the competing methods and the baselines in all the metrics, except for the compatibility against GCN with a small margin. We now discuss each of the three metrics in more detail with more qualitative and quantitative evaluations.

Realism: We conducted a user study with 12 graduate students and 10 professional architects. Each subject compared 75 pairs of layouts sampled from the five targeted methods and the ground-truth. Table 3 shows that HouseGAN has the best overall user score. Figure 5 shows the direct pairwise comparisons. For each pair of methods, we look at the user scores when the 2 methods were compared, compute their average scores, and take the difference. If subjects always choose “better” for one method, the difference would be 2.0. Therefore, the score difference of 1.0 (e.g., GCN against [12] by students) could mean that the method was rated “better” half the time, and “equally X” half the time. The figure shows that both students and architects rate HouseGAN the most realistic except of course the ground-truth. Figure 6 qualitatively supports the same conclusion. Ashual *et al.* did not produce compelling results, because they rather focus on realistic image generation and need rough locations of objects as input, which are not given in our problem. Johnson *et al.* also failed in our experiments. They produce more realistic results if samples from the same group

	CNN -only	GCN	[3]	[12]	Ours	GT
CNN -only		-1.03	0.33	0.33	-0.87	-1.47
GCN	1.03		1.10	1.00	-0.17	-0.70
[3]	-0.33	-1.10		-0.17	-1.20	-1.83
[12]	-0.33	-1.00	0.17		-1.23	-1.80
Ours	0.87	0.17	1.20	1.23		-0.77
GT	1.47	0.70	1.83	1.80	0.77	

(a) Students

	CNN -only	GCN	[3]	[12]	Ours	GT
CNN -only		-1.16	-0.04	0.28	-0.68	-1.64
GCN	1.16		0.92	1.28	-0.24	-0.76
[3]	0.04	-0.92		0.28	-0.76	-1.56
[12]	-0.28	-1.28	-0.28		-1.12	-1.44
Ours	0.68	0.24	0.76	1.12		-0.64
GT	1.64	0.76	1.56	1.44	0.64	

(b) Architects

Fig. 5. Realism evaluation. The user study score differences for each pair of methods by graduate students (left) and professional architects (right). The tables should be read row-by-row. For example, the bottom row shows the results of GT against the other methods.

are included in the training set, allowing the method to memorize answers. We believe that their network is not capable of generalizing to unseen cases.

Diversity: Diversity is another strength of our approach. For each group, we randomly sample 5,000 bubble diagrams and let each method generate 10 house layout variations. We rasterize the bounding boxes (sorted in a decreasing order of the areas) with the corresponding room colors, and compute the FID score. Ashual *et al.* generates variations by changing the input graph into an equivalent form (e.g., apple-right-orange to orange-left-apple). We implemented this strategy by changing the relation from room1-adjacent-room2 to room2-adjacent-room1. However, the method failed to create interesting variations. Johnson *et al.* also fails in the variation metric. Our observation is that they employ GCN but a noise vector is added after GCN near the end of the network. The system is not capable of generating variations. House-GAN has the best diversity scores except for the smallest group, where there is little diversity and the graph-constraint has little effect. Figure 7 qualitatively demonstrates the diversity of House-GAN, where the other methods tend to collapse into fewer modes.

Compatibility: All the methods perform fairly well on the compatibility metric, where many methods collapse to generating a few examples with high compatibility scores. The real challenge is to ensure compatibility while still keeping variations in the output, which House-GAN is the only method to achieve (See Fig. 8). To further validate the effectiveness of our approach, Table 4 shows the improvements of the compatibility scores as we increase the input constraint information (i.e., room count, room type, and room connectivity). The table demonstrates that House-GAN is able to achieve higher compatibility as we add more graph information. Figure 8 demonstrates another experiment, where we

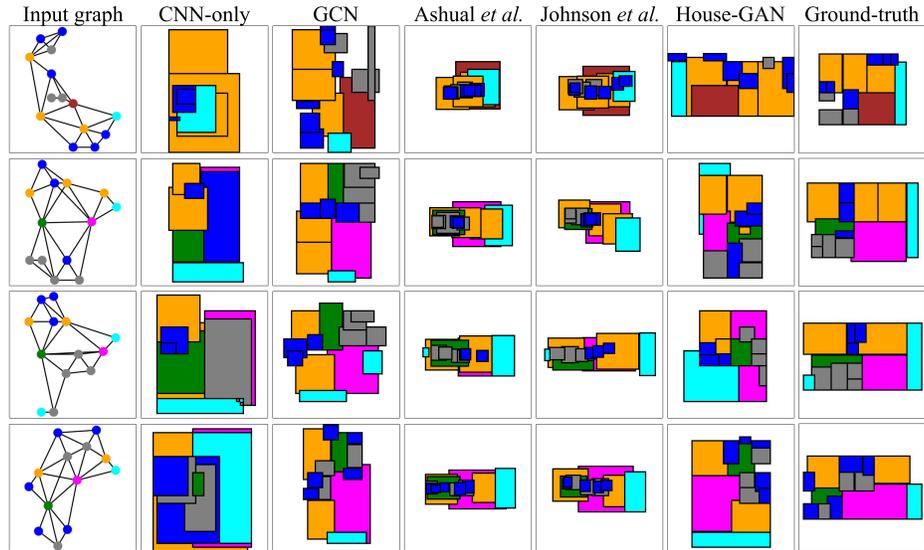


Fig. 6. Realism evaluation. We show one layout sample generated by each method from each input graph. Our approach (House-GAN) produces more realistic layouts whose rooms are well aligned and spatially distributed.

Table 4. Compatibility evaluation: Starting from the proposed approach that takes the graph as the constraint, we drop its information one by one (i.e., room connectivity, room type, and room count). For the top row, where even the room count is not given, it is impossible to form relational networks in House-GAN. Therefore, this baseline is implemented as CNN-only without the room type and connectivity information. The second row (room count only) is implemented as House-GAN while removing the room type information and making the graph fully-connected. Similarly, the third row (room count and type) is implemented as House-GAN while making the graph fully-connected. The last row is House-GAN. We refer to the supplementary document for the full details of the baselines.

Count	Type	Conn.	1-3	4-6	7-9	10-12	13+	
				28.5	28.8	19.0	26.4	32.2
✓			0.6	2.1	4.6	7.3	37.3	
✓	✓		0.4	2.2	4.4	7.5	21.4	
✓	✓	✓	0.1	1.1	2.9	3.9	10.8	

fix the noise vectors and incrementally add room nodes one-by-one. It is interesting to see that House-GAN sometimes changes the layout dramatically to satisfy the connectivity constraint (e.g., from the 4th column to the 5th).

More results and discussion: Figure 9 shows interesting failure and success examples that were compared against the ground-truth in our user study.

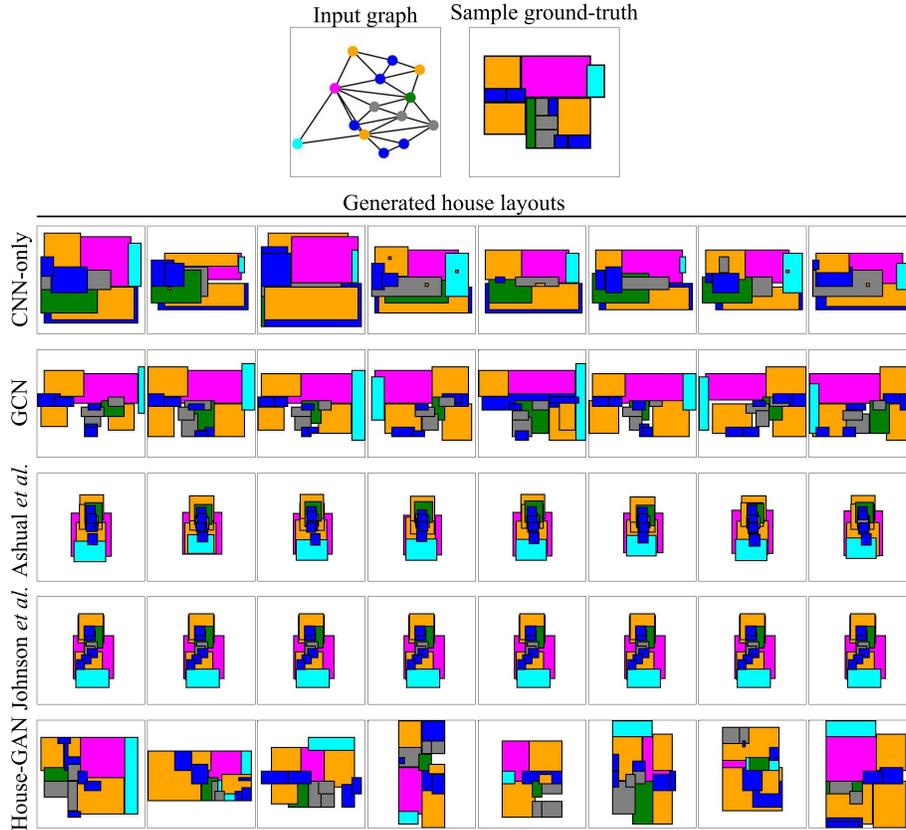


Fig. 7. Diversity evaluation: House layout examples generated from the same bubble diagram. House-GAN (ours) shows the most diversity/variations.

Professional architects rate the three success examples as “equally good” and the three failure cases as “worse” against the ground-truth. For instance, the first failure example looks strange because a balcony is reachable only through bathrooms, and a closet is inside a kitchen. The second failure example looks strange, because a kitchen is separated into two disconnected spaces. Our major failure modes are 1) improper room size or shapes for a given room type (e.g., a bathroom is too big); 2) misalignment of rooms; and 3) inaccessible rooms (e.g., room entry is blocked by closets). Our future work is to incorporate room size information or door annotations to address these issues. Lastly, Figure 10 illustrates the raw output of the room segmentation masks before the rectangle fitting. The rooms are often estimated as rectangular shapes, because rooms are represented as axis-aligned rectangles in our dataset, while the original floor-plan contains non-rectangular rooms. Another future work is the generation of non-rectangular rooms. We refer to supplementary document for more results.

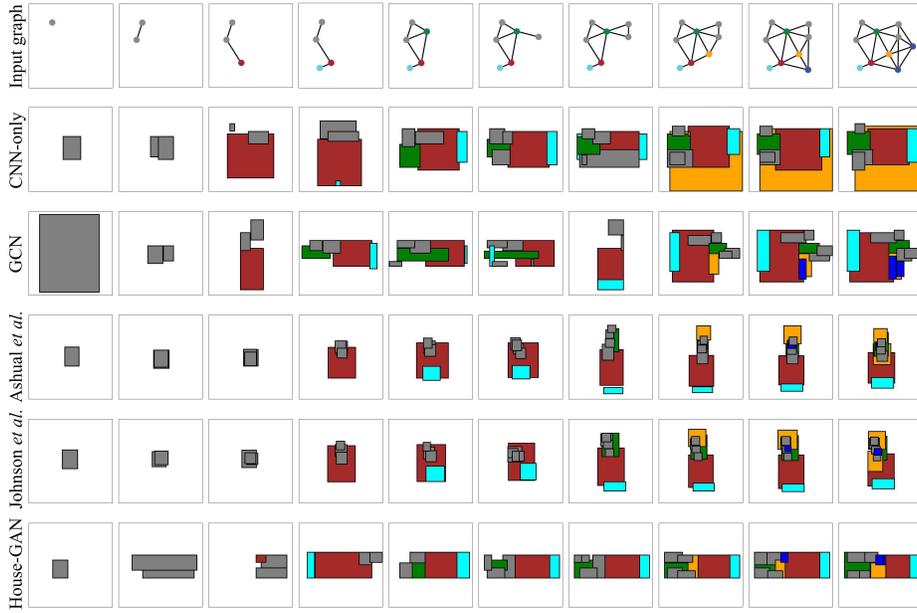


Fig. 8. Compatibility evaluation. We fix the noise vectors and sequentially add room nodes one by one with their incident edges.

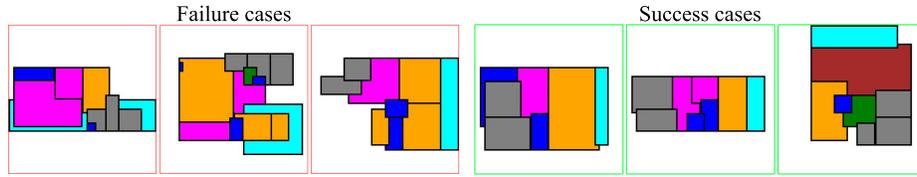


Fig. 9. Failure and success examples by House-GAN from the user study. Architects rate the success examples (right) as “equally good” and the failure examples (left) as “worse” against the ground-truth.



Fig. 10. Raw room segmentation outputs before the rectangle fitting.

7 Conclusion

This paper proposes a house layout generation problem and a graph-constrained relational generative adversarial network as an effective solution. We define three metrics (realism, diversity, and compatibility) and demonstrate the effectiveness of the proposed system over competing methods and baselines. We believe that this paper makes an important step towards computer aided design of house layouts. We will share our code and data.

Acknowledgement: This research is partially supported by NSERC Discovery Grants, NSERC Discovery Grants Accelerator Supplements, and DND/NSERC Discovery Grant Supplement. We would like to thank architects and students for participating in our user study.

References

1. Lifull home’s dataset. <https://www.nii.ac.jp/dsc/idr/lifull>
2. Abu-Aisheh, Z., Raveaux, R., Ramel, J.Y., Martineau, P.: An exact graph edit distance algorithm for solving pattern recognition problems (2015)
3. Ashual, O., Wolf, L.: Specifying object attributes and relations in interactive scene generation. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 4561–4569 (2019)
4. Bao, F., Yan, D.M., Mitra, N.J., Wonka, P.: Generating and exploring good building layouts. *ACM Transactions on Graphics (TOG)* **32**(4), 1–10 (2013)
5. Choi, Y., Choi, M., Kim, M., Ha, J.W., Kim, S., Choo, J.: Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8789–8797 (2018)
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
7. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. In: Advances in neural information processing systems. pp. 5767–5777 (2017)
8. Harada, M., Witkin, A., Baraff, D.: Interactive physically-based manipulation of discrete/continuous models. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. pp. 199–208 (1995)
9. Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A.: Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* **9**(1), 1–22 (2013)
10. 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. In: Advances in Neural Information Processing Systems. pp. 6626–6637 (2017)
11. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1125–1134 (2017)
12. Johnson, J., Gupta, A., Fei-Fei, L.: Image generation from scene graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1219–1228 (2018)
13. 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 International Conference on Computer Vision. pp. 9895–9904 (2019)
14. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of stylegan. *arXiv preprint arXiv:1912.04958* (2019)
15. Kwon, Y.H., Park, M.G.: Predicting future frames using retrospective cycle gan. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1811–1820 (2019)
16. Li, J., Yang, J., Hertzmann, A., Zhang, J., Xu, T.: Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767* (2019)
17. Liu, C., Wu, J., Kohli, P., Furukawa, Y.: Raster-to-vector: Revisiting floorplan transformation. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2195–2203 (2017)
18. Ma, C., Vining, N., Lefebvre, S., Sheffer, A.: Game level layout from design specification. In: *Computer Graphics Forum*. vol. 33, pp. 95–104. Wiley Online Library (2014)

19. Merrell, P., Schkufza, E., Koltun, V.: Computer-generated residential building layouts. In: *ACM Transactions on Graphics (TOG)*. vol. 29, p. 181. ACM (2010)
20. Müller, P., Wonka, P., Haegler, S., Ulmer, A., Van Gool, L.: Procedural modeling of buildings. In: *ACM SIGGRAPH 2006 Papers*, pp. 614–623 (2006)
21. Peng, C.H., Yang, Y.L., Wonka, P.: Computing layouts with deformable templates. *ACM Transactions on Graphics (TOG)* **33**(4), 1–11 (2014)
22. Ritchie, D., Wang, K., Lin, Y.a.: Fast and flexible indoor scene synthesis via deep convolutional generative models. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 6182–6190 (2019)
23. Wang, K., Lin, Y.A., Weissmann, B., Savva, M., Chang, A.X., Ritchie, D.: Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)* **38**(4), 132 (2019)
24. Wang, K., Savva, M., Chang, A.X., Ritchie, D.: Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)* **37**(4), 1–14 (2018)
25. 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)
26. Zhang, F., Nauata, N., Furukawa, Y.: Conv-mpn: Convolutional message passing neural network for structured outdoor architecture reconstruction. *arXiv preprint arXiv:1912.01756* (2019)
27. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: *Proceedings of the IEEE international conference on computer vision*. pp. 2223–2232 (2017)