House-GAN++: Generative Adversarial Layout Refinement Networks

Nelson Nauata
Simon Fraser University
nnauata@sfu.ca

Sepidehsadat Hosseini
Simon Fraser University
sepidh@sfu.ca

Kai-Hung Chang
Autodesk Research
kai-hung.chang@autodesk.com

Hang Chu
Autodesk Research
hang.chu@autodesk.com

Chin-Yi Cheng
Autodesk Research
chin-yi.cheng@autodesk.com

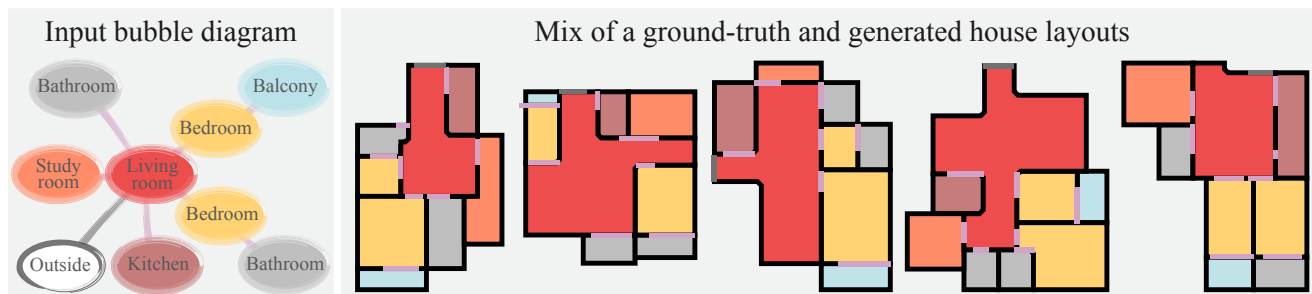Yasutaka Furukawa
Simon Fraser University
furukawa@sfu.ca

Figure 1. The paper makes a breakthrough in the task of automated house layout generation. The right shows the mix of a ground-truth design made by an architect and our generated samples, based on the input bubble-diagram. Can you tell which one is the ground-truth? See the end of the caption for the answer. The paper proposes a novel generative adversarial layout refinement network, whose generator is trained to repeatedly apply and refine the design towards perfection. (The second sample from the right is the ground-truth.)

Abstract

This paper proposes a novel generative adversarial layout refinement network for automated floorplan generation. Our architecture is an integration of a graph-constrained relational GAN and a conditional GAN, where a previously generated layout becomes the next input constraint, enabling iterative refinement. A surprising discovery of our research is that a simple non-iterative training process, dubbed component-wise GT-conditioning, is effective in learning such a generator. The iterative generator also creates a new opportunity in further improving a metric of choice via meta-optimization techniques by controlling when to pass which input constraints during iterative layout refinement. Our qualitative and quantitative evaluation based on the three standard metrics demonstrate that the proposed system makes significant improvements over the current state-of-the-art, even competitive against the ground-truth floorplans, designed by professional architects. We will share our code, model, and data.

1. Introduction

House design is a time-consuming iterative process, requiring multiple rounds of refinements. An architect sketches out a design, evaluates, adjusts, and repeats the cycles until being satisfied with a design within a given time budget. Unfortunately, designing an effective floorplan is possible only by professional architects, where a small fraction of buildings (less than 10% in North America) employ a design professional for custom design due to the cost. Automatic floorplan generation will have tremendous impacts on the trillion-dollar real-estate/construction industries.

Floorplan generation has been an active area of research. Early methods formulated it as iterative optimization [22, 21]. The surge of deep neural networks has made a breakthrough, where state-of-the-art algorithms [24, 7] utilize Generative Adversarial Networks (GANs) [10]. GAN is a one-shot generation process, converting a noise vector into a sample. A conditional GAN could take a design in progress and produce a refined version iteratively [25]. However, there does not exist a database of incomplete floorplans during design iterations, making such a training infeasible.
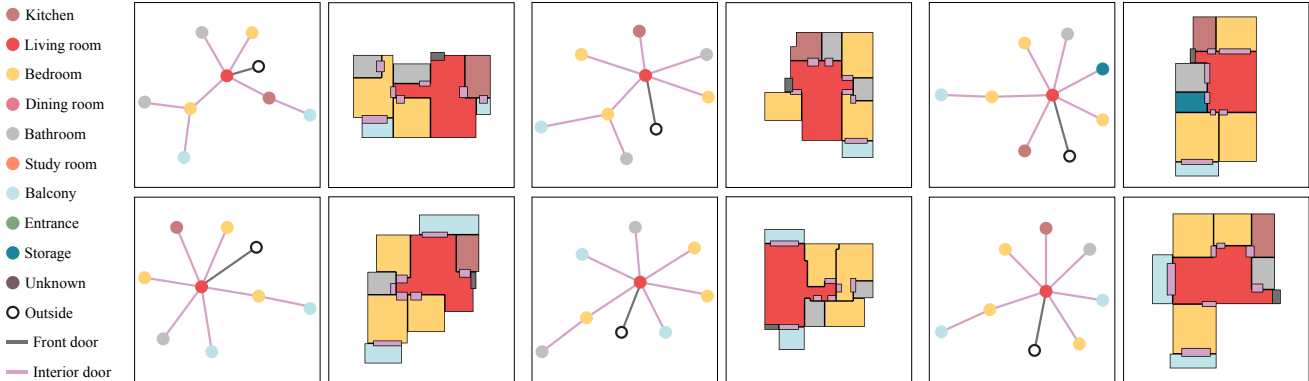
Figure 2. We have parsed the RPLAN dataset [30] to prepare 60k house layouts with the corresponding bubble-diagrams. Our task is to generate a realistic and diverse set of floorplans that are compatible with the input bubble-diagram.

This paper proposes a novel generative adversarial layout refinement network, where the generator is trained to generate a floorplan as a graph of segmentation masks by iteratively refining its design (See Fig. 1). We borrow the problem setup from the state-of-the-art system House-GAN [24], while making a few extensions towards a more challenging production-level task (i.e., handling non-rectangular room shapes, generating doors/entrances, and using a functional graph instead of an adjacency graph).

The technical contribution of this paper is three-fold: an architecture, a training algorithm, and a test-time meta-optimization algorithm. First, the architecture is an integration of a graph-constrained relational GAN (i.e., based on the current state-of-the-art [24]) and a conditional GAN [25], where a previously generated model becomes the next input constraint, enabling iterative layout refinement. Second, a surprising discovery of our research is that a computationally affordable non-iterative training process, dubbed component-wise GT-conditioning, is effective in learning an iterative generator, where a ground-truth segmentation is passed to each component as a condition at a random probability. Third, our framework creates a new opportunity in further optimizing a metric of choice via meta-optimization, such as Bayesian optimization by controlling when to pass which constraints.

We have used the RPLAN dataset [30], which offers 60k vector-graphics floorplans designed by professional architects. Qualitative and quantitative evaluations based on the three standard metrics (i.e., realism, diversity, and compatibility) in the literature demonstrate that the proposed system outperforms the current-state-of-the-art by a large margin. The system is even competitive against the ground-truth floorplans, based on user studies judged by professional architects. We will share our code, model, and data.

2. Related Work

This section reviews generative models for structured data such as objects, 3D buildings, or floorplans.

Traditional methods: Classical algorithms formulate optimization or design hand-crafted rules for structured data generation. Procedural modeling is an early successful approach for 3D building models, where shape-grammars define iterative generation processes [23, 3]. Integer programming is formulated to generate an arrangement of tiles [26]. Optimization has also proven effective for game-level design [20, 12]. For floorplans specifically, Bayesian network is trained to learn distributions of architectural components for stochastic sample generation [22].

Neural approach (one-step generation): With the surge of deep learning, researchers started to train deep neural networks (DNNs) to learn a single-step model generation. Convolutional neural networks are trained to add objects into a room one by one for indoor scene generation [29, 27]. A scene graph generation in addition to the object placement are learned step by step via DNNs [28]. Neural turtle graphics learns RNNs to encode incoming paths and generate outgoing edges at one node [8]. An iterative algorithm uses the RNN modules repeatedly to generate road layouts. While being a reconstruction task as opposed to a generative one, RoadTracer learns a CNN that takes the current partial network then decides on a next action for reconstructing a road network from a satellite image [4].

Neural approach (joint generation): Joint generation of multiple components in an entire structured model is more challenging. Room boundaries as a raster image is estimated from a house foot-print by a CNN, followed by a series of heuristics to generate a vectorized floorplan [30]. Image generation has been an active area of research, where realistic object arrangements are learned via variational auto encoder [15], a differentiable renderer [16], or graph convo-
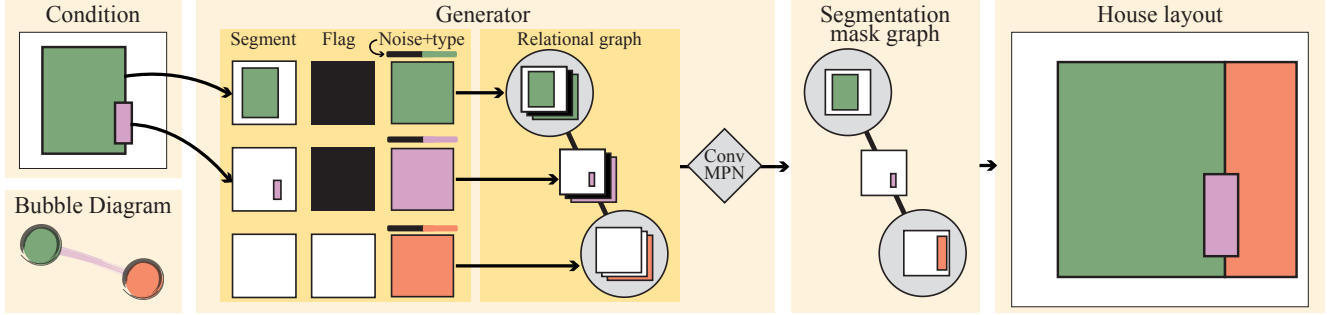
Figure 3. Our architecture is built on top of a relational GAN from the state-of-the-art system [24]. An additional 2D segmentation mask for each room/door can be specified as an input condition, enabling iterative design refinement.

lutional networks [14, 2]. Graph generation is modeled as a sequence of graph editing operations, whose entire process is learned by recurrent neural networks [31] or graph recurrent attention networks [17]. Natural language generation is another related area of research, where an auto-regressive model is trained by masking words from ground-truth sentences in the input [18]. The adversarial training is also employed for the same language task [9] in a seq2seq architecture. This paper takes the idea of masking and adversarial training to the task of 2D layout generation, where our architecture jointly estimates segmentation masks for all architectural components (as opposed to a recurrent formulation) and is capable of iteratively refining a design.

Neural approach (joint generation w/ refinement): On the iterative sample refinement, GANHopper is the closest to ours in spirit [19]. The key difference is that our system produces a structured model (as opposed to a raster image) and the training process is non-sequential.

3. Problem Formulation

In a standard design process, architects incorporate constraints from clients into a sketch called a bubble-diagram, then convert it into a floorplan through design explorations and iterations. We borrow the problem setup from the state-of-the-art system House-GAN [24], while adding a few extensions towards the production-level task.

Input: The input to the system is a bubble-diagram, which is represented as a graph where a node encodes a room with its room type (See Fig. 2) [1]. In the original problem, an edge encoded spatial adjacency of two rooms. In this work, an edge encodes a functional connection ("interior door" or "front door") as in real architect's sketches. An "interior door" is a connection between two rooms, and a "front door" is a connection between a room and the outside area.

---

[1]Room types are: "living room", "kitchen", "bedroom", "balcony", "entrance", "dining room", "study room", "storage", "unkown", or "outside". Note that "outside" is not an actual room and we define for the convenience of defining a "front door" as an edge.

Output: The output of the system is a segmentation mask for each room and door. We also utilize an off-the-shelf floorplan vectorization algorithm [6] to convert the layout into a vector floorplan image, where a room is represented by an axis-aligned closed-polygon, adjacent rooms share the walls with the common line segments, and a door is represented as a line-segment on a wall.

Metrics: Realism, diversity, and compatibility are the three metrics, evaluating the performance as in the prior work [24]. Roughly speaking, realism is an average user rating, diversity is the Fréchet Inception Distance (FID) [13], and the compatibility is the graph edit distance (GED) [1] between the input bubble-diagram and the one constructed from the output layout. To account for the addition of doors, we adjusted the metric definitions slightly, whose details are referred to the supplementary document. We describe the details of the user study in Sect. 5.

4. Technical Innovations

Learning an iterative refinement process is non-trivial due to the lack of databases containing step-by-step design iterations. Our architecture is a straightforward integration of a relational GAN [24] and a conditional GAN [25]. A surprising discovery of our research is that a simple non-iterative training procedure is effective in training such an iterative generator. The iterative refinement capability opens up a new opportunity in further improving the metric of choice by meta-optimizing the refinement scheme at test-time. The section explains the architecture, the training algorithm, and the meta-optimization algorithm.

4.1. Architecture

Following House-GAN [24], our network backbone is a convolutional message passing network (Conv-MPN [32]), whose relational graph structure is defined by the bubble-diagram (See Fig. 3). There are three key differences in our architecture: 1) Edges in addition to nodes carry features for the generation of doors; 2) Each node/edge takes a 2D segmentation mask as an additional input constraint with an
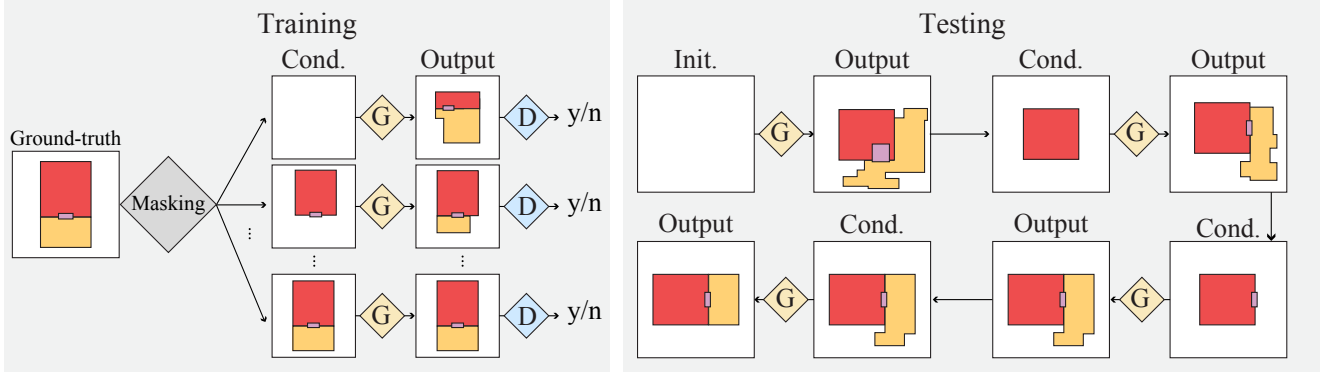
3

Figure 4. During training, we specify a GT segmentation mask for each room/door with a 50% chance. The generator needs to learn the task of inpainting missing components when many GT masks are given, or the task of generating a complete design when few masks are given. During testing, previously generated layouts are passed to the generator as potential input constraints, enabling iterative design refinement.

associated new loss; and 3) Conv-MPN feature pooling [32] is reformulated to allow feature-exchange between nodes and edges. We now explain the three differences in detail.

Edge features: House-GAN has a 10-d one-hot vector, which encodes a room type and initializes a node feature vector. In our work, doors have two types, and we extend the one-hot vector to 12-d over the mix of 10 room types and 2 door types. With the common type vector, door generation from an edge becomes the same as room generation from a node, except the pooling mechanism in the convolutional message passing as detailed in the following paragraph.

Mask condition: House-GAN initializes each node with a noise vector and a room-type, which is transformed into a $8 \times 8 \times 16$ feature volume. Our relational generator takes in an additional $64 \times 64 \times 2$ condition image for each node/edge. The first channel provides the segmentation mask, which we expect the generator to learn to keep unchanged. The second channel becomes 1 for every pixel when the segmentation mask is specified, otherwise 0. We use a 3-layer CNN to convert the condition image to $8 \times 8 \times 16$, which is concatenated to the original feature. When the segmentation mask is specified, we enforce an L1-loss between the condition image $\hat{\mathbf{m}}_i$ and the generated mask $\mathbf{m}_i$. i is the index of a node or an edge. Precisely, let $\mathcal{I}$ denote the set of node/edge indices where the condition masks are specified. The loss function is defined as follows, where L_{org} is the original generator adversarial loss [11] in House-GAN and λ is 1000:

$$L = L_{org} + \lambda \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} |\mathbf{m}_i - \hat{\mathbf{m}}_i|. \quad (1)$$

Conv-MPN pooling: Conv-MPN message passing is de-

fined based on the connectivity of the relational graph [32]:

$$\mathbf{g}_r \leftarrow \text{CNN} \left[\mathbf{g}_r \quad ; \quad \underset{s \in \mathcal{N}(r)}{\text{Pool}} \mathbf{g}_s \quad ; \quad \underset{s \in \overline{\mathcal{N}}(r)}{\text{Pool}} \mathbf{g}_s \right]. \quad (2)$$

$\mathbf{g}_r$ denotes the feature volume for a component r. $\mathcal{N}(r)$ and $\overline{\mathcal{N}}(r)$ denote the neighbors and its complement. We redefine $\mathcal{N}(r)$ to allow feature exchange between rooms and doors. First, a room is a neighbor of its connected rooms and the doors in-between. Second, a door is a neighbor of its incidental two rooms. The rest of the architecture is the same as House-GAN, except that we upsample the feature volume three times instead of twice, where the final segmentation mask is produced in the resolution of 64×64.

4.2. Component-wise GT-conditional training

Our training strategy is similar in spirit to "masking" in NLP [9] (See Fig. 4). In each training step, we pick a ground-truth layout at random, initialize a relational graph with its bubble-diagram, then specify a ground-truth segmentation mask as the input condition for each room/door with a 50% chance. This strategy forces the generator to either create a whole layout when few GT conditions are specified, or simply inpaint the missing components when many GT conditions are given. Surprisingly, this simple training process allows the generator to iteratively improve a design by passing the previously generated layout as the input condition.

4.3. Meta-optimizing iterative refinement scheme

Iterative layout refinement starts by running a generator without input constraints. From the second iteration, we have an option of specifying the previously generated room/edge masks. What is the best strategy? Is it the best to always pass all the constraints? We propose three strategies that control when to pass which constraints based

on: 1) fixed heuristics; 2) static node/edge properties; or 3) dynamic layout information. In the latter two cases, we parameterize the space of strategies and employ a meta-optimization algorithm to seek for good solutions subject to either our diversity or compatibility metrics. In all strategies, we run the generator 10 times to refine a layout.

Fixed heuristics: The first heuristic is to pass the segmentation mask for each room/door at a 50% chance at every iteration ($\text{Ours}_{heur}^{50\%}$). The second heuristic is to always specify all the conditions ($\text{Ours}_{heur}^{100\%}$).

Static scheme (Ours_{static}): Our static scheme is based on the observation that architects start by designing certain spaces first (e.g., a living room). We parameterize the scheme by a 12-d vector $\{V_i\}$, where i is an index over 10 room types and 2 door types. Each element takes a value in the range $[1, 10]$. Suppose V_3 is 4. In this scheme, the previous mask is specified as the input condition for the 3_{rd} room (or door) type after the 4_{th} iteration. FID (i.e., diversity) score is the target metric for the optimization.

Dynamic scheme (Ours_{dyna}): Another intuitive strategy is to control the conditioning based on the compatibility of the current model with the input bubble-diagram. We parameterize the scheme by two 12-d vectors $\{T_i\}$ and $\{U_i\}$, where i is again the index over the room/door type. Suppose T_3 is 4, and U_3 is 7. In this scheme, the previously generated mask is specified after the 4_{th} (resp. 7_{th}) iteration if the current room (door) is compatible (resp. incompatible) for the 3_{rd} room type. FID (i.e., diversity) or a graph edit distance (i.e., compatibility) is the target metric.

5. Implementation Details

We have used PyTorch for implementation and a workstation with dual Xeon CPUs and dual NVIDIA Titan RTX GPUs. We adopt the same training configuration as in House-GAN except for the batch size (1 instead of 32) and the number of iterations (500k instead of 300k). [2] For the meta scheme optimization, we use the Python library Hyperopt [5] with Tree-structured Parzen Estimators. In each round, we compute the target metric on 1,000 bubble-diagrams from the training set, and pick the best scheme after 500 rounds.

Testing framework: To avoid a network from simply copying and pasting layouts, we use the k-fold cross validation from House-GAN [24], while dividing the samples into four groups based on the number of rooms: (5, 6, 7, 8): When generating layouts with 8 rooms, we use samples from the other three groups for training. At test time, we randomly pick a GT layout from the test set, use its bubble-diagram to

---

[2] WGAN-GP [11], ADAM ($b1 = 0.5$, $b2 = 0.999$), the learning rate (0.0001), the number of critics (1), and leaky-ReLUs ($\alpha = 0.1$) for all non-linearities except for the last one with hyperbolic tangent.

initialize a relational graph, and generate a layout sample. We repeat the process 1,000 times to generate 1,000 samples for the evaluation. For the diversity and compatibility evaluations, we compute the mean and the standard deviation of the metrics over five rounds. The same process is used for the layouts with the other room counts.

Realism user study: We have carried out the user study with 10 amateurs (i.e. engineers and graduate students) and 10 professional architects on two layout representations. The first is the raw output of the systems as pixel-wise segmentation masks. The second is a vector-floorplan representation where we have used an off-the-shelf floor-plan vectorization system (Floor-SP [6]) for the conversion of the samples except for the ground-truth. Each room/door is assigned a unique color based on its type as in Fig. 2. A subject is presented a pair of layouts as in Fig. 5, and asked which one is more realistic or a tie. A method scores $(+1)$ for a win, (-1) for a loss, and (0) for a tie. For each pair of methods, we generate 100 pairs to be presented to 10 amateurs and 10 professional architects, where the average score becomes the realism metric. Please see the supplementary document for the full details of the user study.

Competing methods: We compare against the three competing methods with their official implementations: House-GAN [24], Ashual *et al*. [2], and Johnson *et al*. [14]. For House-GAN [24], the room shapes were simplified to rectangles via pre-processing in their work. In our experiments, non-rectangular room shapes are used for fair comparison. For Ashual *et al*. [2] and Johnson *et al*. [14], we convert our bubble-diagram and floorplan data into their scene-graph representations, while limiting to three connection types ("room-room", "room-door", or "none").

6. Experimental Results

Table 1 provides the main results, where we use the proposed approach with a baseline refinement scheme ($\text{Ours}_{heur}^{50\%}$). The table shows that our system outperforms all the other methods in all the metrics with clear margins, which are the greatest for the most challenging task (i.e., column "8", generating layouts of 8 rooms, where layouts of 5, 6, or 7 rooms are used for training). Note that a realism metric is first computed for pairs of methods as described above, then their average is reported for each method.

Qualitative evaluations on realism, diversity, and compatibility are given in Figs. 5, 6, and 7, respectively. The figures show that our layouts are more realistic with better spatial arrangement of rooms and their individual shapes. House-GAN does a reasonable job especially in the diversity evaluation, but doors are often missing and room boundaries have noticeable artifacts. Compatibility is more challenging in our problem, where the functional connectivity is given in the bubble diagram and small doors need to

Table 1. The main quantitative evaluations. Realism is measured by a user study with amateurs and professional architects. Diversity is measured by the FID scores. Compatibility is measured by the graph edit distance. (↑) and (↓) indicate the-higher-the-better and the-lower-the-better metrics, respectively. The cyan, orange, and magenta colors indicate the first, the second, and third best results, respectively.

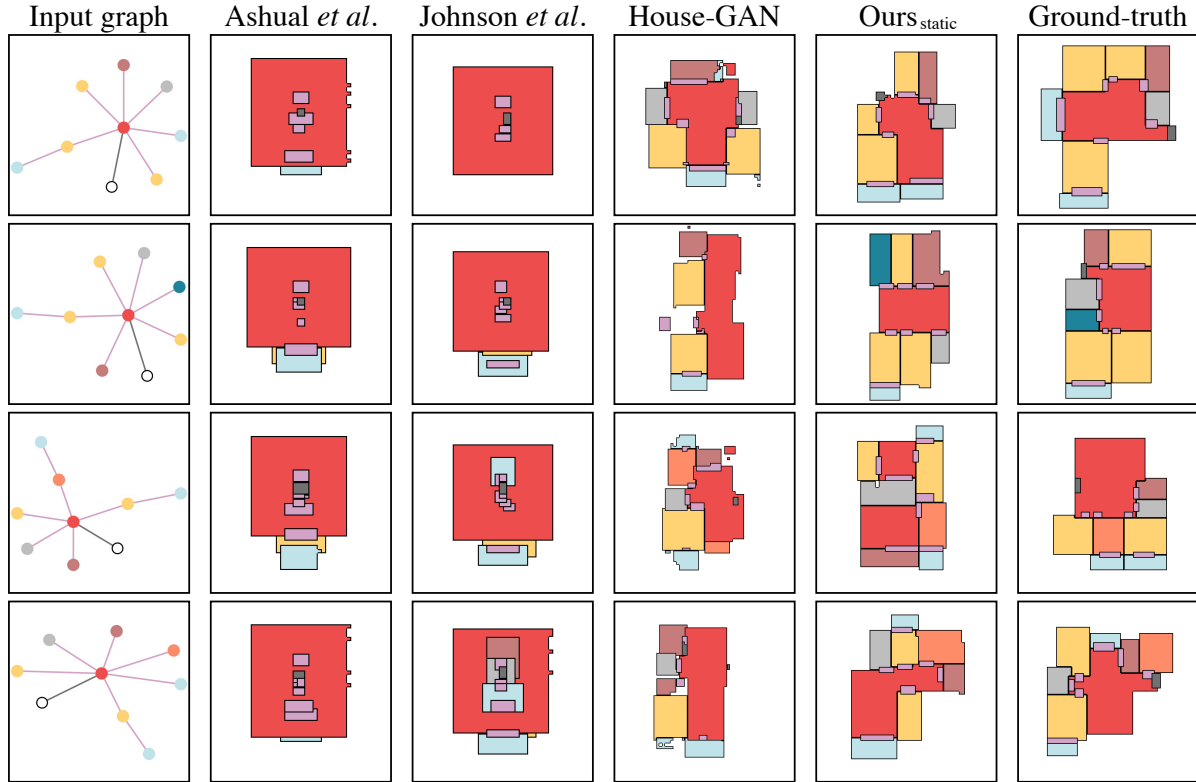| Model	Realism (↑)	Diversity (↓)				Compatibility (↓)			
	8	5	6	7	8	5	6	7	8
Ashual *et al.* [2]	-0.7	120.6±0.5	172.5±0.2	162.1±0.4	183.0±0.4	7.5±0.0	9.2±0.0	10.0±0.0	11.8±0.0
Johnson *et al.* [14]	-0.7	167.2±0.3	168.4±0.4	186.0±0.4	186.0±0.4	7.7±0.0	6.5±0.0	10.2±0.0	11.3±0.1
House-GAN [24]	0.0	37.5±1.1	41.0±0.6	32.9±1.2	66.4±1.7	2.5±0.1	2.4±0.1	3.2±0.0	5.3±0.0
Ours$_{heur}^{50\%}$	0.2	30.4±4.4	37.6±3.0	27.3±4.9	32.9±4.9	1.9±0.3	2.2±0.3	2.4±0.3	3.9±0.5

Figure 5. Realism evaluation. One generated layout is shown for each input bubble diagram.

be placed precisely at the room boundaries. Fig. 8 demonstrates how the iterative refinement improves the quality of a design over time with our method Ours$_{static}$. Both the room spatial arrangement and their shapes make significant improvements over the iterations.

The complete pairwise realism scores are provided in Fig. 10. The left is the results with raw segmentation visualization, where two of our methods (Ours$_{heur}^{50\%}$ and Ours$_{static}$) are compared against the three competing methods and the ground-truth. Our scheme-optimized system (Ours$_{static}$) is the best among the competing methods but is still behind GT with a big margin. In direct comparisons between Ours$_{static}$ and GT, subjects choose "Ours is better" only 6% of the time ("tie" is 20%). The right is the results with the vector-floorplan visualization, comparing against the best competing method House-GAN and the ground-truth. For this evaluation, in addition to Ours$_{static}$, we have prepared another variant Ours$_{static*}$ where only fully compatible samples (roughly 10% in our generations) are used in the evaluation. While GT is still the winner, Ours$_{static*}$ scores -0.18 against GT, where 0.0 is the ultimate score when our results become indistinguishable from GT. Subjects chose "Ours is better", "tie", and "GT is better" at 25%, 32%, and 43% of the time, respectively, indicating that even professional architects struggle to distinguish our results from GT (See Figs. 9 and 10).
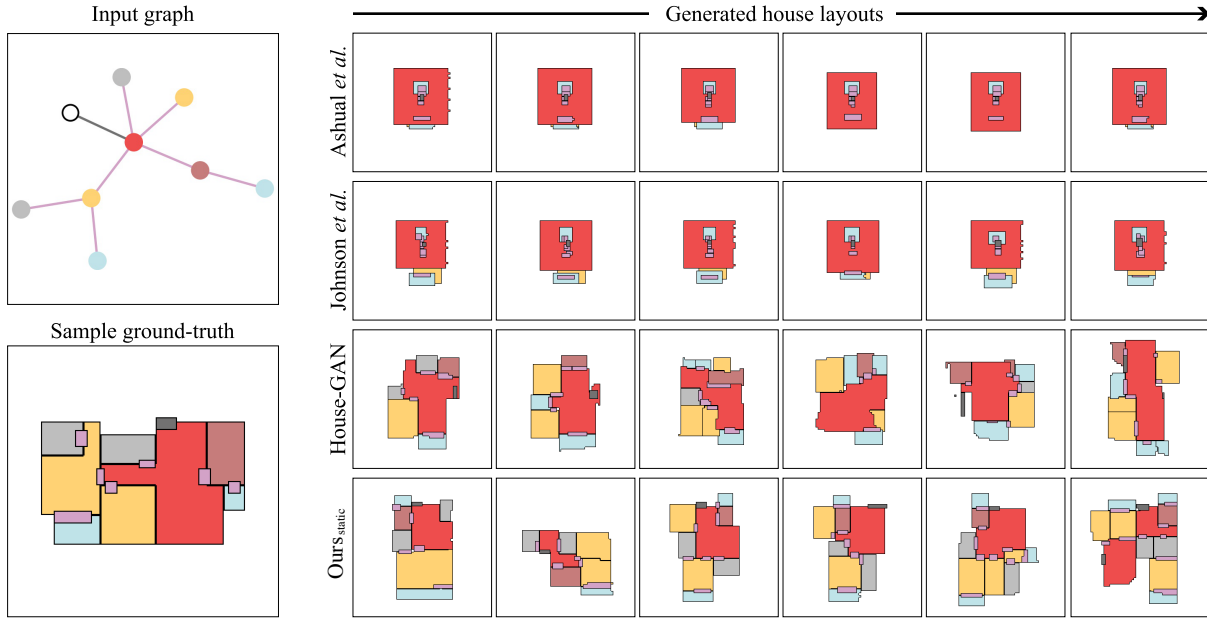
Figure 6. Diversity evaluation. Given an input bubble-diagram, we show six samples generated by each method.

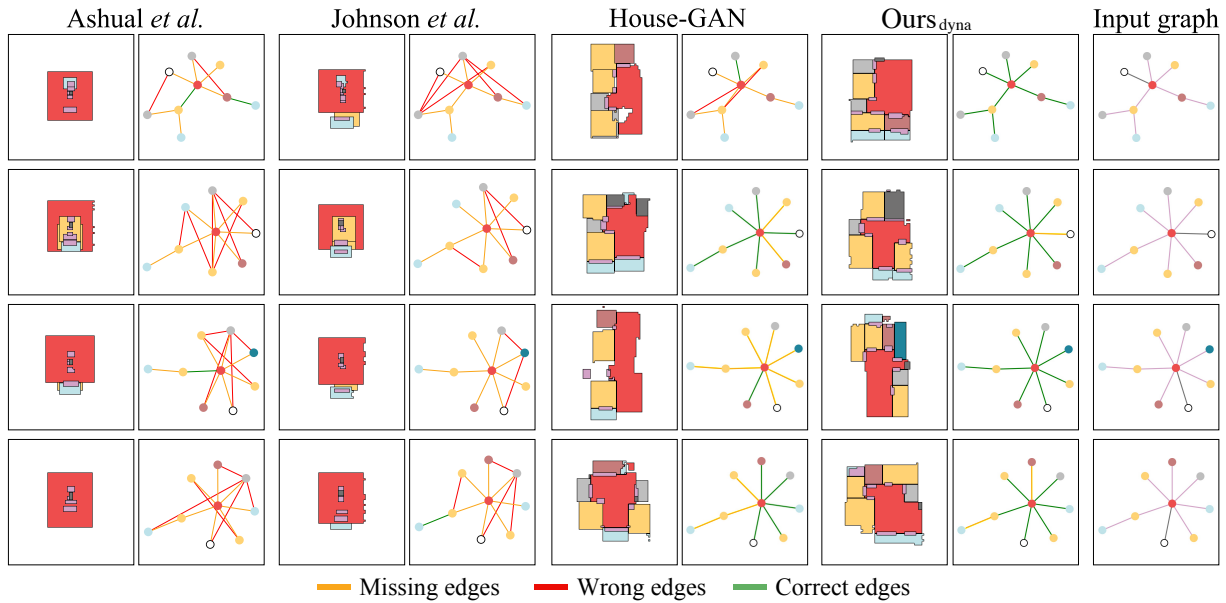Missing edges — Wrong edges — Correct edges

Figure 7. Compatibility evaluation. The figure shows the inconsistency between the input bubble diagram and the ones constructed by the output layouts. The orange, red and green colors indicate if an edge is missing, wrongly predicted, or correctly predicted.

Lastly, Table 2 demonstrates the effectiveness of the refinement scheme optimization, where the diversity or the compatibility is the target metric. The compatibility optimization requires the information of the current design (i.e., dynamic information) and cannot be optimized with Ours$_{static}$. Overall, optimized refinement schemes outperform heuristic schemes consistently. The best compatibility measure is achieved when the compatibility metric is the target as expected. Looking at the scheme parameters, we found that in 9 out of 12 node/edge types, the scheme passes the mask-conditions at earlier iterations when being compatible than when being incompatible, which agrees with our intuition. The best diversity measure is achieved when the diversity is the metric. Looking at the parameters of Ours$_{static}$, we found that the scheme passes the mask-conditions in the order of doors, living room, kitchen, bed-

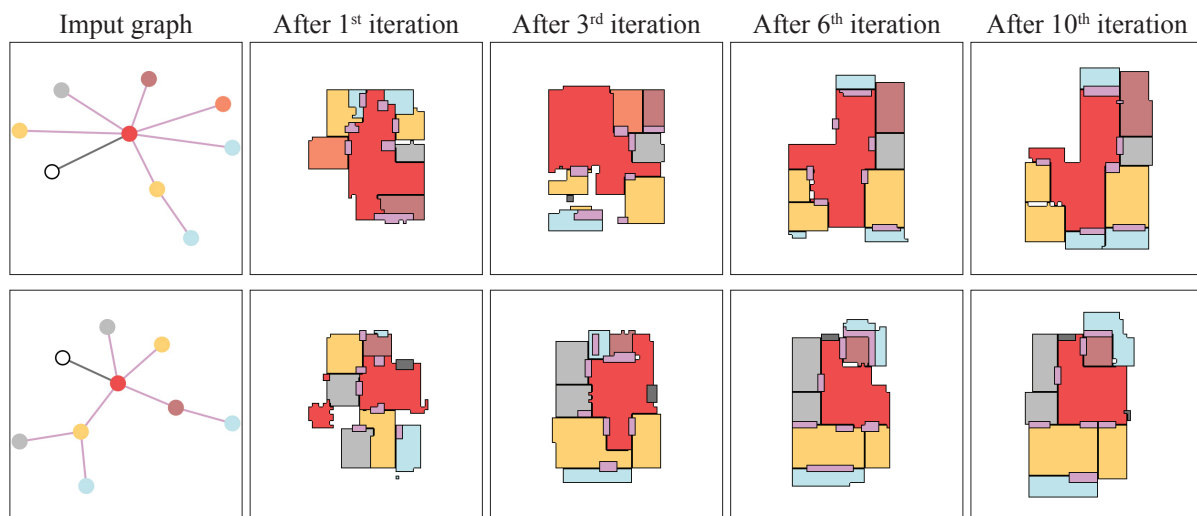| Imput graph | After 1st iteration | After 3rd iteration | After 6th iteration | After 10th iteration |

Figure 8. Iterative layout refinement. Each row shows the input bubble-diagram and the iterative refinement process over the 10 iterations.
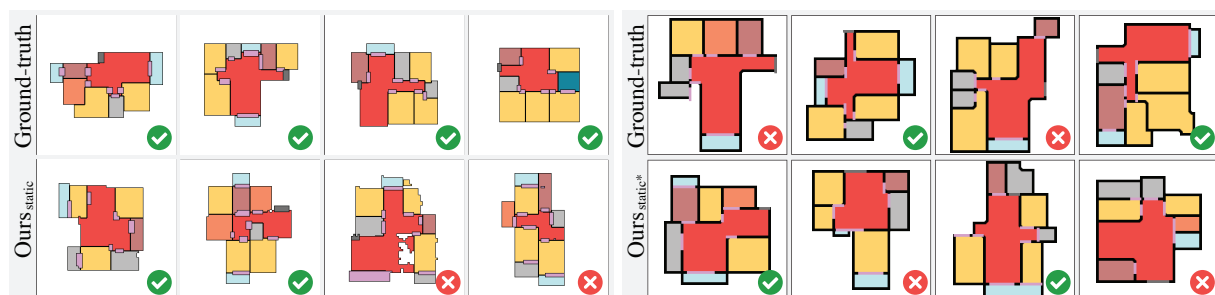

Figure 9. Representative user study results. The left is the raw segmentation visualization and the right is the vector-floorplan visualization. Each column is a pair presented to the subjects. A green mark denotes the preferred design. Two green marks denote a tie.

Table 2. Refinement scheme optimization. The diversity and the compatibility metrics are shown for House-GAN and five variants of our method. Meta optimization is used to optimize the refinement scheme in the last three rows. "Target" column indicates the target metric of the scheme-optimization, where "Divers." and "Compat." indicate FID and the graph edit distance metrics.

Model	Target	Divers. ($\downarrow$)	Compat. ($\downarrow$)
House-GAN	N/A	66.4±1.7	5.3±0.0
$\text{Ours}_{heur}^{50\%}$	N/A	32.9±4.9	3.9±0.5
$\text{Ours}_{heur}^{100\%}$	N/A	31.2±0.6	3.7±0.1
Ours_{static}	Divers.	27.3±1.1	3.2±0.1
Ours_{dyna}	Divers.	24.7±0.8	3.7±0.0
Ours_{dyna}	Compat.	35.7±1.3	2.6±0.0

Figure 10. Realism scores based on the user study for each pair of methods (or GT). The tables are to be read row-by-row: The bottom row shows that the GT receives positive scores against all the other methods. The left is the evaluation with raw segmentation masks, and the right is with the vector-floorplan images.

room, and bathroom. This was counter-intuitive at first, because one designs doors last. This is due to the fact that doors are small and the effects of the L1-loss are minimal. We found that the doors often move when the input condi-

tions are specified. The rest of the room order makes sense, as it is easy to start with rooms at the core (i.e., living room or kitchen).

7. Conclusion

This paper makes a breakthrough in the task of automated house layout generation via a novel generative adversarial layout refinement network, generating vector floorplans often indistinguishable from ground-truth. While we have evaluated the system for an offline layout generation task, computational capability of refining an incomplete design is also effective in incorporating user inputs: Architects can take a design, makes adjustments, and pass back to the system for refinement. We believe that this paper sets a milestone in the literature of automated house layout generation, even with potentials in the space of interactive layout design. We will share our code, model, and data.

References

[1] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. 2015. 3

[2] Oron Ashual and Lior Wolf. Specifying object attributes and relations in interactive scene generation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4561–4569, 2019. 3, 5, 6

[3] Fan Bao, Dong-Ming Yan, Niloy J Mitra, and Peter Wonka. Generating and exploring good building layouts. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013. 2

[4] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4720–4728, 2018. 2

[5] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123, 2013. 5

[6] Jiacheng Chen, Chen Liu, Jiaye Wu, and Yasutaka Furukawa. Floor-sp: Inverse cad for floorplans by sequential room-wise shortest path. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2661–2670, 2019. 3, 5

[7] Qi Chen, Qi Wu, Rui Tang, Yuhan Wang, Shuai Wang, and Mingkui Tan. Intelligent home 3d: Automatic 3d-house design from linguistic descriptions only. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12625–12634, 2020. 1

[8] Hang Chu, Daiqing Li, David Acuna, Amlan Kar, Maria Shugrina, Xinkai Wei, Ming-Yu Liu, Antonio Torralba, and Sanja Fidler. Neural turtle graphics for modeling city road layouts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4522–4530, 2019. 2

[9] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: Better text generation via filling in the␣. *arXiv preprint arXiv:1801.07736*, 2018. 3, 4

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 1

[11] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017. 4, 5

[12] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1–22, 2013. 2

[13] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017. 3

[14] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1219–1228, 2018. 3, 5, 6

[15] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. Layoutvae: Stochastic scene layout generation from a label set. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9895–9904, 2019. 2

[16] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767*, 2019. 2

[17] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4255–4265, 2019. 3

[18] Yi Liao, Xin Jiang, and Qun Liu. Probabilistically masked language model capable of autoregressive generation in arbitrary word order. In *Annual Meeting of the Association for Computational Linguistics*, 2020. 3

[19] Wallace Lira, Johannes Merz, Daniel Ritchie, Daniel Cohen-Or, and Hao Zhang. Ganhopper: Multi-hop gan for unsupervised image-to-image translation. In *ECCV*, 2020. 3

[20] Chongyang Ma, Nicholas Vining, Sylvain Lefebvre, and Alla Sheffer. Game level layout from design specification. In *Computer Graphics Forum*, volume 33, pages 95–104. Wiley Online Library, 2014. 2

[21] Jess Martin. Procedural house generation: A method for dynamically generating floor plans. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 1–2, 2006. 1

[22] Paul Merrell, Eric Schkufza, and Vladlen Koltun. Computer-generated residential building layouts. In *ACM Transactions on Graphics (TOG)*, volume 29, page 181. ACM, 2010. 1, 2

[23] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers*, pages 614–623. 2006. 2

[24] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *ECCV*, 2020. 1, 2, 3, 5, 6

[25] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016. 1, 2, 3

[26] Chi-Han Peng, Yong-Liang Yang, and Peter Wonka. Computing layouts with deformable templates. *ACM Transactions on Graphics (TOG)*, 33(4):1–11, 2014. 2

[27] Daniel Ritchie, Kai Wang, and Yu-an Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6182–6190, 2019. 2

[28] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)*, 38(4):132, 2019. 2

[29] Kai Wang, Manolis Savva, Angel X Chang, and Daniel Ritchie. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018. 2

[30] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG)*, 38(6):1–12, 2019. 2

[31] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018. 3

[32] Fuyang Zhang, Nelson Nauata, and Yasutaka Furukawa. Conv-mpn: Convolutional message passing neural network for structured outdoor architecture reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2798–2807, 2020. 3, 4