EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

Floorplan Generation by Alternating Geometry and Semantics Optimization - Supplementary Material

Wenming Wu[1,2] Sizhe Hu[1] Ligang Liu[3] Liping Zheng[4,1] † Xiao-Ming Fu[3]

[1]Hefei University of Technology, China
[2] Anhui Province Key Laboratory of Industry Safety and Emergency Technology (Hefei University of Technology), China
[3]University of Science and Technology of China, China
[4]Anhui University of Technology, China

1. Implementation Details

1.1. Multi-channel image input

Our model takes as input a multi-channel image of resolution 128×128. The specific channels are as follows (Figure 1):

The *Inside mask* is a binary representation that highlights the interior regions of the architectural boundary. Pixels corresponding to the interior of the layout are assigned a value of 1, while all exterior pixels are set to 0.

The *Boundary mask* encodes the structural boundaries of the layout. Pixels representing the exterior walls are assigned a value of 0.5, while the front door region is highlighted with a value of 0.75.

The *Front-door mask* explicitly marks the position of the front door in the layout. A 5×5 square centered on the front door is assigned a value of 1, with all other pixels set to 0.

The *Inside-boundary mask* combines both interior and boundary information into a unified representation. Pixels within the interior are assigned a value of 1, exterior walls are represented with a value of 0.5, and the front door region is highlighted with a value of 0.75.

The *Room-box mask* encodes the categorical information of individual rooms. Each room is represented by a unique label corresponding to its functional category (e.g., bedroom, kitchen), which is assigned to all pixels within the room's bounding box.

The *Room-center mask* is designed to localize the geometric center of each room box. A 10×10 square centered at the geometric center of each room is assigned a value of 1, while all other pixels remain 0.

The *Previous-map mask* contains the semantics map from the last iteration of the optimization process. This mask encodes the categorical and spatial information of the layout from the previous step, serving as iterative context for further refinement. For the first iteration, this mask is omitted, as there is no prior information available.

1.2. Initializing room layouts

We use *BoxGenerator* to initialize room layouts, as shown in Figure 2 and Figure 3. The input to *BoxGenerator* is the given architectural boundary $\mathcal{B}$ and $r_{<i}$, the room boxes already present in the scene. $\mathcal{Q}$ is a learnable attribute vector, which is used to capture the global information of the current scene and serves as an attribute querying vector $\widehat{\mathcal{Q}}$ in the generation phase to create the attributes of the new room box. We adopt an autoregressive approach to generate room boxes.

1.2.1. Feature extraction

We employ a (128×128) multi-channel image to depict the input of $\mathcal{B}$, from which conditional features are extracted. The input includes the following information at each pixel, which defaults to 0:

- *Inside mask*: taking a value of 1 for the interior.
- *Boundary mask*: taking a value of 0.5 for the exterior walls and 0.75 for the front door.
- *Front-door mask*: taking a value of 1 for the front door mask. Using a (5×5) square centered on the front door.

We utilize *ResNet-18* [HZRS16] to extract a 64-dimensional boundary feature from the *Inside mask* and *Boundary mask*, and a 64-dimensional front door feature from the *Front-door mask* via another *ResNet-18* [HZRS16]. These features are concatenated to form the conditional feature. For $r_{<i}$, we obtain the specific attributes for each room box. The room category is encoded into a 64-dimensional feature with an MLP. For central location and room size, we use the positional encoding [VSP*17] to obtain a 64-dimensional feature for each item. These features are then concatenated for the corresponding room box. Before passing to the Transformer encoder, we map the concatenated feature to a 64-dimensional feature as the room feature using an MLP.

1.2.2. Box generation

The conditional feature and all room features of existing room boxes are then concatenated with the learnable attribute vector $\mathcal{Q}$, feeding to the Transformer encoder. We use these features to predict

Figure 1: *Our models accept a* 128×128 *multi-channel image as input, where each channel encodes specific spatial, categorical, or architectural information. These channels include inside mask, boundary mask, Front-door mask, Inside-Boundary mask, Room-box mask, Room-center mask, and Previous-map mask. For visualization, the pixel values in these channels are scaled.*

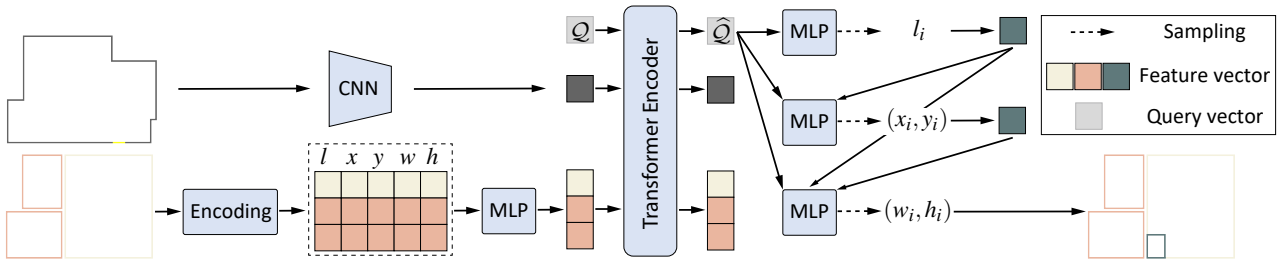


Figure 2: *Network architecture of BoxGenerator. We extract the conditional feature from the architectural boundary* $\mathcal{B}$ *and the room features from the predicted room boxes. These features are concatenated with a learnable attribute vector* $\mathcal{Q}$ *and passed to a Transformer encoder. The encoder predicts an attribute vector* $\widehat{\mathcal{Q}}$ *that contains contextual information of the room layout, used to autoregressively sample new room box attributes. The attribute vector* $\widehat{\mathcal{Q}}$ *progressively produces these attributes.*

a new room box, implementing a multi-head attention Transformer. The Transformer encoder outputs an attribute vector $\widehat{\mathcal{Q}}$, which is used to sample attributes of the new room box. We employ a room box generator to produce attributes of the new room box with the attribute vector $\widehat{\mathcal{Q}}$. To enable autoregressive prediction of attributes, we condition the prediction of a particular attribute on the previously predicted attributes: room category first, followed by central location and room size. $\mathcal{P}(r_i \mid \mathcal{B}, r_{<i}) = \mathcal{P}(l_i, x_i, y_i, w_i, h_i \mid \mathcal{B}, r_{<i}) = \mathcal{P}(l_i \mid \mathcal{B}, r_{<i})\mathcal{P}((x_i, y_i) \mid \mathcal{B}, r_{<i}, l_i)\mathcal{P}((w_i, h_i) \mid \mathcal{B}, r_{<i}, l_i, x_i, y_i)$. We model attributes of room boxes with a mixture of ten logistics distributions, which are used to sample specific attributes of room boxes.

1.3. Network architecture

1.3.1. BoxGenerator

The core network architecture of *BoxGenerator* is the Transformer, and the network configuration of the *BoxGenerator*'s Transformer is shown in Figure 4. We adopt an autoregressive approach to generate room boxes. A generation process of the fixed generation ordering is shown in Figure 3. Some *BoxGenerator*'s training hyperparameters are also shown.

1.3.2. Semantics network

We adopt a modified version of *ASPP* [CPK*17] for the semantics network, as shown in Figure 5. Some hyperparameter settings for training semantics networks are shown in Figure 6.

1.3.3. Geometry network

We follow *Deformable DETR* [ZSL*20] to design the geometry network. Some hyperparameter settings for training geometry networks are shown in Figure 7.

1.4. Constrained generation

1.4.1. Free generation

Our method can generate floorplans without user input, relying solely on the priors it has learned. This free generation is particularly suited for conceptual design, providing innovative design solutions. In the implementation, we need no input. From the results, it is evident that the free generation allows for a high degree of freedom. This approach can generate floorplans with boundaries that may not have appeared in the dataset, yet maintain logical and functional room layouts.

1.4.2. Box-constrained generation

Users can define the floorplan domain as a box, essentially serving as the generated floorplan's bounding box. The box-constrained generation requires that the generated floorplans lie within a given box (i.e., the bounding box of generated floorplans). This approach is beneficial when considering rough building sites or spatial limitations without specific requirements of architectural boundaries, ensuring that the generated floorplans adapt to particular geographical or spatial conditions. In the implementation, we employ a (128×128) single-channel image to depict the input of the building box, which is *Box mask*: taking a value of 1 for the building

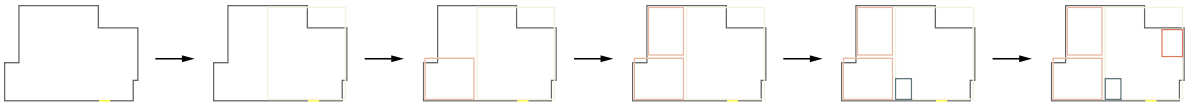


Figure 3: *Given the architectural boundary as input, BoxGenerator progressively generates room boxes in a specific order.*

Transformer	Value
n_layers	8
n_heads	16
query_dimensions	64
value_dimensions	64
feed_forward_dimensions	1024

(a) Network architecture of Transformer

Training	Value
Batch size	64
Epoch	500
Optimizer	Adam
Learning rate	1e-4
Weight decay	1e-2

(b) Hyperparameters

Figure 4: *Hyperparameter settings for BoxGenerator. (a) The network architecture of BoxGenerator is based on a Transformer framework. (b) Some key hyperparameters for training this model. The learning rate is reduced by a factor of 0.1 at specific epochs: the 299th, 349th, 399th, and 449th.*

Network	Output shape
Input	[n, 128, 128]
Resnet.conv1	[64, 64, 64]
Resnet.bn1 Resnet. relu	[64, 64, 64]
Resnet. maxpool	[64, 32, 32]
Resnet. layer1	[64, 32, 32]
Resnet. layer2	[128, 16, 16]
DeepLab-ASPP dilation= [6, 12, 18, 24]	[64, 16, 16]
convT	[#class, 128, 128]

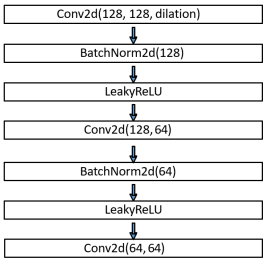


Figure 5: *Architecture of the semantics network. The modified ASPP block of DeepLab-ASPP is shown on the right. This design enables the network to capture precise contextual information at multiple scales, improving its ability to distinguish between different semantic objects in an image.*

box and 0 for others. Box-constrained generation requires that the generated floorplan not exceed the specified building box. From the results, it is apparent that box-constrained generation imposes certain limitations on the generation process. It provides a middle ground in floorplan design, allowing for creating floorplans with unique boundaries not previously seen in the dataset while ensuring that the resulting floorplans have practical and reasonable sizes. For the input of a floorplan box, we use *ResNet-18* [HZRS16] to extract the 64-dimensional constraint feature from the box image and feed it to *BoxGenerator*, as shown in Figure 8. We represent the floorplan box as a (128×128) image for encoding the semantics and geometry networks.

1.4.3. Graph-constrained generation

We represent the relationships between rooms of a floorplan using a graph: the graph nodes represent rooms, while the edges denote the adjacency or connectivity between rooms. Graph-constrained

Training	Value
Batch size	8
Epoch	100
Optimizer	Adam
Learning rate	1e-4
Weight decay	1e-4

Figure 6: *Hyperparameter settings of the semantics network. A key parameter in this configuration is the learning rate, which is strategically adjusted to optimize the network's performance over time. Specifically, the learning rate is reduced by a factor of 0.1 every 30 epochs.*

Training	Value
Batch size	32
Epoch	300
Other settings are the same as *Deformable DETR*	

Figure 7: *Hyperparameter settings of the geometry network. Other settings are the same as Deformable DETR [ZSL*20].*

generation offers a more flexible and versatile approach to guide floorplan generation. This approach allows enhanced generation control over the floorplan by manipulating the nodes and edges of a graph. By specifying, adding, or removing nodes, the generated floorplans can be directly influenced by the types and quantities of nodes. Altering the edges between rooms affects how rooms are arranged or connected, allowing the overall layout topology to be changed. There are two types of graph constraints: the topological graph and the bubble diagram. The difference lies in the attributes of the nodes and edges. In the topological graph, nodes have only one attribute, the room category label. Edges denote the connectivity between rooms, as used in the *House-GAN* series [NCC*20; NHC*21] and *HouseDiffusion* [SHF23]. In contrast, nodes in a bubble diagram not only include the room category label but also encompass attributes like the room's central location and size. Edges denote the adjacency between rooms, which includes not only room pairs that are connected by doors, but also spatially adjacent room pairs, as used in *Graph2Plan* [HHT*20] and *WallPlan* [SWL*22]. The topological-graph-constrained generation is similar to the box-constrained generation, which can provide a certain degree of generative control. The bubble-diagram-constrained generation provides significantly more generative con-

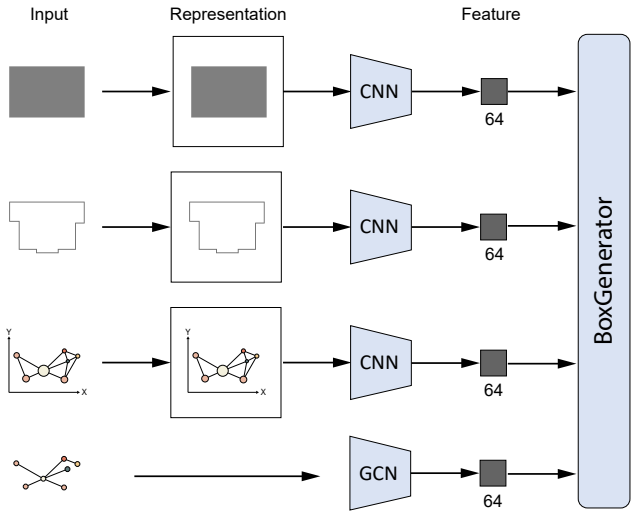


Figure 8: *Extracting constraint features for BoxGenerator. For the input of the floorplan box, architectural boundary, or bubble diagram, we use ResNet-18 [HZRS16] to extract the constraint feature from the representation of the constraint image. For the topological graph, we use GCNConv [KW16] for graph encoding.*

trol, which is attributed to the image-based encoding of the bubble diagram. Although no boundary constraints are imposed, the generated room layouts and architectural boundaries are similar in almost all results. As shown in Figure 8, for the topological graph implementation, we use *GCNConv* [KW16] for graph encoding, obtaining a 64-dimensional graph feature and feeding it to *BoxGenerator*. For the bubble diagram, we refer to the approach of *WallPlan* [SWL*22], converting the bubble diagram into an image for our image-based encoding. We use *ResNet-18* [HZRS16] to extract the 64-dimensional constraint feature from the bubble diagram image and feed it to *BoxGenerator*. For the semantics network and geometry network, we directly encode the topological graph and represent the bubble diagram as a (128×128) image for encoding.

1.4.4. Boundary-constrained generation

Users input the boundary of a floorplan as the design constraint, and the final generated floorplan must completely match these specified boundaries. Boundary constraints can be further divided into two types: with and without a front door. The front door constraint requires the living room in the generated floorplan to be connected to the front door. In the methodology sections, we have discussed the boundary-constraint generation with a front door in detail. Removing related information from the multi-channel image of inputs is sufficient for the boundary-constraint generation without a front door. The boundary-constrained generation with a front door provides stronger control over the generation without a front door, especially for the generation of living rooms. For the input of an architectural boundary, we use *ResNet-18* [HZRS16] to extract the 64-dimensional constraint feature from the boundary image and feed it to *BoxGenerator*, as shown in Figure 8. If the input boundary has a front door, we utilize *ResNet-18* [HZRS16] to extract a 64-dimensional boundary feature from the boundary image and a 64-dimensional front door feature from the front door image. These features are then concatenated to form the conditional feature for *BoxGenerator*. For the semantics and geometry networks, we represent the architectural boundary as a (128×128) image for encoding.

1.4.5. Hybrid-constrained generation

The aforementioned constraints can be combined to form the hybrid-constrained generation, such as floorplan generation based on the architectural boundary, plus the bubble diagram. Hybrid-constrained generation combines different constraints, allowing for more personalized and customized designs. From the results, it is clear that Hybrid-constrained generation exhibits a stronger capability for control over the generation process, especially when bubble diagram constraints are combined with other types of constraints. This similarity suggests that the design space is significantly smaller. For the input of hybrid constraints, we just need to extract the constraint features for each type of constraint separately and then concatenate them together for feeding to *BoxGenerator*.

2. Experiments

2.1. Statistics comparison

We further analyze room area statistics by comparing the area distributions of generated and real floorplans for *bedroom*, *kitchen*, *bathroom*, *balcony*, and *storage*. Following prior work, we visualize the absolute difference between the corresponding frequency distributions (Difference of Frequencies), where lower values indicate better alignment with the ground truth. As shown in Figure 9, our method consistently exhibits smaller distribution gaps than both *RPLAN* and *ATISS*-based initialization, demonstrating improved consistency with real floorplan statistics.

2.2. Perceptual study

To evaluate the realism of the generated floorplans, we also conduct perceptual studies to compare the floorplans generated by our method with the ground-truth floorplans in the dataset and the generated floorplans by state-of-the-art methods. Each set of perceptual studies consists of 15 comparison tasks, where each pair of floorplans is generated by our method and the competitor. In each task, participants are asked to compare a pair of floorplans and select the one that is better than the other or "*Not sure*", based on consideration of the aesthetics, functionality, and connectivity of the floorplans. We recruited 20 undergraduate students who had undergone interior design training to complete all of the perception studies. Therefore, for each group of the perceptual study, there are 300 tasks, with 15 tasks for each of 20 participants. We show example tasks in Figure 10. Each task contains the same question: "*Which one do you think is better for you?*" In each task, we show two floorplans generated by different methods and order them randomly. We provide participants with three options: "*Choosing left*", "*Choosing right*", and "*Not sure*".

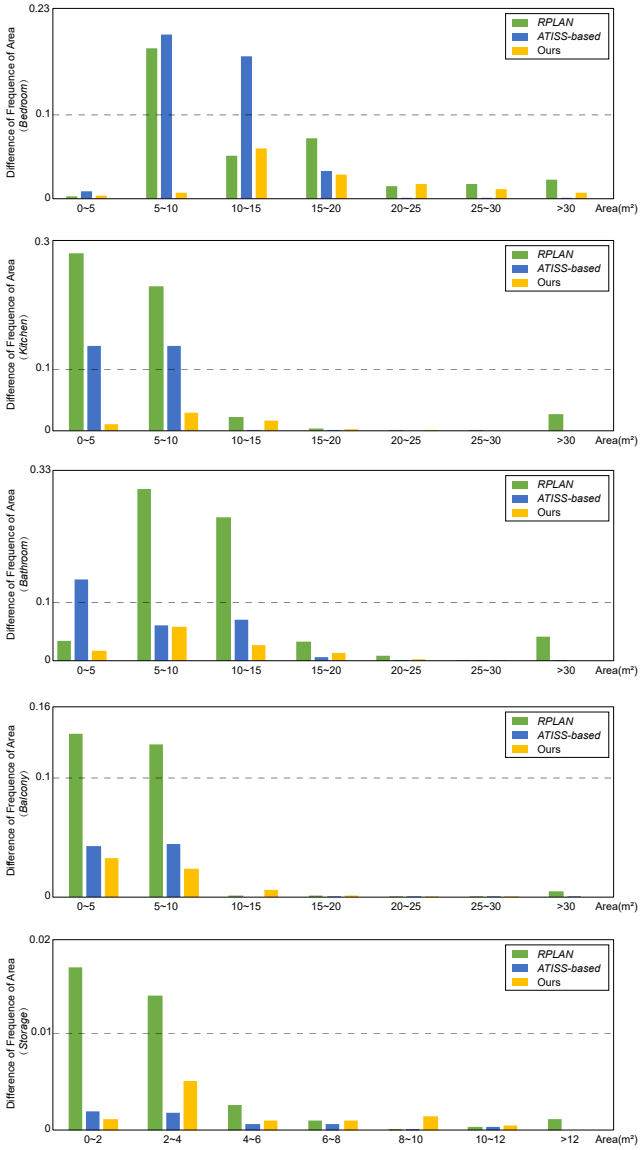


Figure 9: *Difference of frequencies. We visualize the absolute difference between the area distributions of generated and real floorplans for* bedroom, kitchen, bathroom, balcony, *and* storage. *Lower is better; our method consistently exhibits smaller distribution gaps than* RPLAN *and* ATISS*-based initialization.*

3. Abaltion study

We have performed several ablation studies to verify the effectiveness of our framework design.

3.1. *Unordered*

For the *Unordered*, the order of room boxes in the training dataset is randomized during the training of the *BoxGenerator*. This involves shuffling the sequence of room boxes for each floorplan in the training dataset, disrupting any inherent ordering by size or frequency.

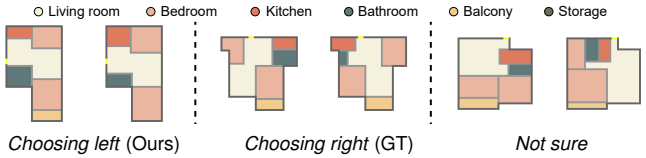


Figure 10: *Several example tasks in the perceptual study when comparing the floorplans generated by our method to the ground-truth floorplans.*

Consequently, the *BoxGenerator* learns to predict room box attributes (e.g., category, position, size) without leveraging structured contextual information. During inference, room boxes are generated in a random order, leading to initial layouts that often lack coherence, exhibit overlapping rooms, and fail to prioritize larger or more frequent room types effectively. For inference, we use the same termination conditions.

3.2. *OrderedArea*

For the *OrderedArea*, the generation of room boxes in the initialization phase is ordered exclusively based on the room area. Room boxes are sorted in descending order of their area sizes, prioritizing the generation of larger rooms before smaller ones. This approach simplifies the ordering strategy by focusing only on room size, without considering other factors such as room occurrence frequency. As a result, the generation process may lack the nuanced guidance provided by a more comprehensive ordering strategy, potentially leading to less functional or coherent initial layouts. For inference, we use the same termination conditions.

3.3. *NoCoarseOpt*

For the *NoCoarseOpt*, the coarse semantics optimization is entirely omitted, while keeping the fine semantics optimization. Instead, to optimize the initial room layout, we train a network specifically for layout-to-layout optimization. This network processes a 128×128 multi-channel image input, similar to the geometry networks, but excludes the *Previous-map mask*. Based on *Deformable DETR* [ZSL*20], this network extracts a global room layout from the initial room layout. To construct the semantics map fed to the fine semantics optimization, we directly generate the *Room-box mask* from the room layout. The fine semantics optimization is then employed to refine this semantics map. For inference, we use the same termination conditions.

3.4. *NoFineOpt*

For the *NoFineOpt*, the fine semantics optimization is omitted entirely, while keeping the coarse semantics optimization. Instead, we leverage the layout-to-layout optimization network described earlier and directly refine the layout generated by the coarse semantics optimization. For inference, we use the same termination conditions.

3.5. *NoSemanticsOpt*

For the *NoSemanticsOpt*, the entire refinement process is carried out through the box-to-box optimization network, bypassing any semantics optimization. This approach eliminates both the coarse and fine semantics optimization, relying solely on the layout-to-layout optimization to refine the initial room layout. The layout-to-layout optimization, as described previously, directly adjusts the room layout iteratively, focusing exclusively on geometric aspects such as room sizes, positions, and overlaps. For inference, we use the same termination conditions.

3.6. *NoGeometryOpt*

For the *NoGeometryOpt*, the geometry optimization is entirely removed from our framework. This uses the *BoxGenerator* to initialize the room layout, followed by the *Coarse Semantics Optimization* and *Fine Semantics Optimization* to refine the semantics map. The initial room layout generated by the *BoxGenerator* is directly fed into the semantics optimization process without further refinement through geometry optimization. The coarse semantics optimization generates an initial semantics map from the input constraints and room layout. The fine semantics optimization further refines the semantics map to enhance pixel-level semantic accuracy and room category labeling. For inference, we use the same termination conditions.

3.7. *BoxCoarseOpt*

In the *BoxCoarseOpt*, the method uses only the *BoxGenerator* followed by the *Coarse Semantics Optimization*, omitting the fine semantics and geometry optimization steps. The initial room layout is generated by the *BoxGenerator*, and the coarse semantics optimization is applied to refine the semantics map at a high level without further iterative improvements. Specifically, the *BoxGenerator* predicts the room layout, including room categories, positions, and sizes. The *Coarse Semantics Optimization* then processes this room layout along with the input constraints to generate an initial semantics map. This step ensures that the semantics map roughly aligns with the input constraints and room layout, providing category labeling for each room region.

3.8. *BoxFineOpt*

In the *BoxFineOpt*, the method uses only the *BoxGenerator* followed by the *Fine Semantics Optimization*, omitting the coarse semantics optimization and geometry optimization steps. The initial room layout is generated by the *BoxGenerator*, and the fine semantics optimization directly refines the semantics map based on this initial layout without any coarse initialization. To construct the semantics map used in the fine semantics optimization, we directly generate the *Room-box mask* from the initial room layout. This mask assigns the room category value for each room box and combines them to form the initial semantics map. For the first iteration of the fine semantics optimization, the initial semantics map is constructed solely from this *Room-box mask*. Subsequent iterations of the fine semantics optimization refine the semantics map using both the input constraints and the updated semantics map from the previous iteration. For inference, we use the same termination conditions as in the complete method.

3.9. *SAOpt*

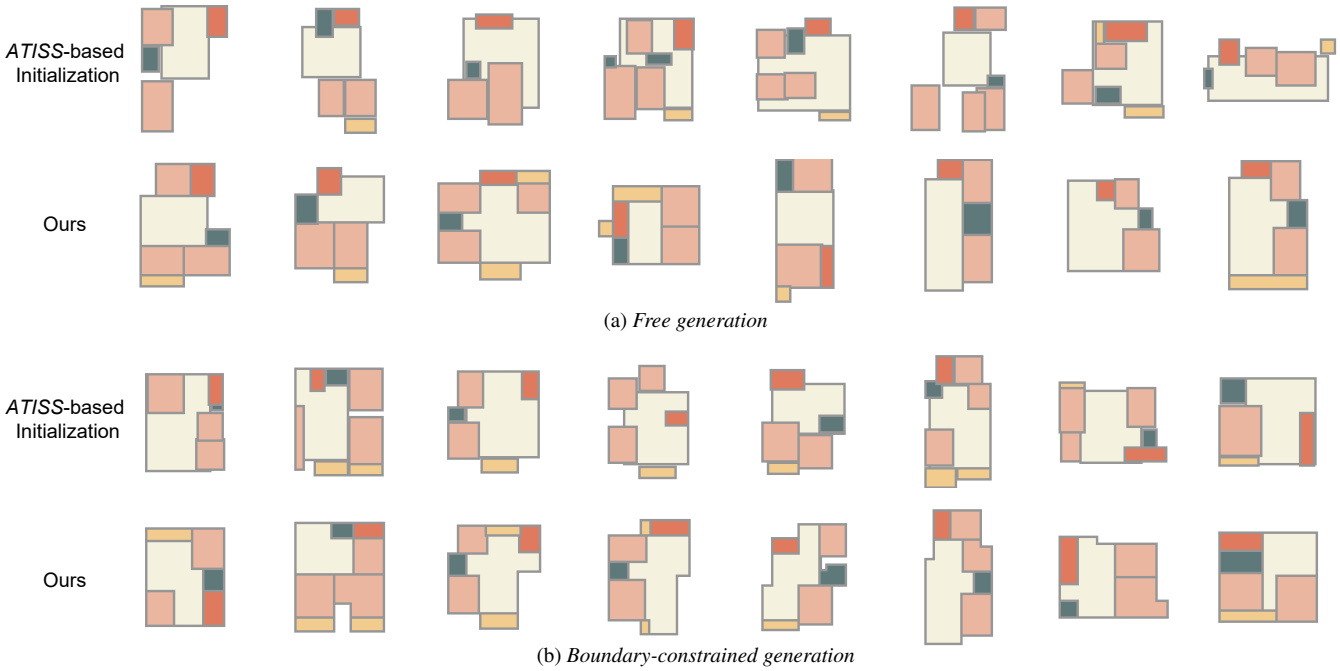
For the *SAOpt*, the geometry optimization network is replaced by a simple simulated annealing (SA) algorithm. Based on the semantics map refined by the coarse and fine semantics optimization, the simulated annealing algorithm directly optimizes the geometry of each room. Specifically, for each room mask extracted from the semantics map, the initial bounding rectangle is set to the mask's bounding box. The simulated annealing algorithm then iteratively refines room boxes by maximizing the Intersection over Union (IoU) between the room mask and the room box. The optimization process involves randomly moving each rectangle edge by ±1 pixel at each step, with an initial temperature of "10", a cooling rate of "0.99", and termination conditions of either reaching "1000" iterations or a temperature below "1e-3". This setup simplifies the refinement process by focusing purely on fitting room geometries to their semantic regions, but it lacks the global optimization capabilities provided by the geometry network. For inference, we use the same termination conditions as in the complete method.

3.10. *ATISS*-based initialization

We implement an *ATISS*-based initialization by directly adopting the layouts generated by *ATISS*-style box generation, without applying any subsequent geometry or semantics refinement. Specifically, the room layouts are produced in a single forward pass and directly rendered as the final result, following the standard ATISS generation protocol. As illustrated in Figure 11, this initialization often suffers from evident semantic and structural inconsistencies, such as unreasonable room proportions, fragmented functional regions, and poor alignment between geometry and room semantics. These issues are difficult to resolve without further refinement. In contrast, our method significantly improves both semantic coherence and global structural rationality through iterative geometry–semantics optimization, demonstrating that the performance gains stem from the proposed refinement framework rather than the *ATISS*-based initialization itself.

4. More results

Our floorplan generation framework can enable a wide range of applications, offering extensive flexibility and practicality. In this part, we will discuss a few practical applications of our method, moving beyond the initial focus on architectural boundaries to include more complex and valuable design constraints. Our method is very scalable for different design constraints. Table 1 gives the navigation for more results. From the results, it can be seen that the hybrid-constrained generation has more control over the generation process, especially when bubble diagram constraints are combined with other types of constraints. As the number of constraints increases, the design space is significantly reduced. As shown in Figure 24, the generation floorplans tends to be the same with the input of a boundary, a front door, and a bubble diagram.



(a) *Free generation*

(b) *Boundary-constrained generation*

Figure 11: *Ablation study on* ATISS*-based initialization versus our full framework under free generation (top) and boundary-constrained generation (bottom). Without iterative refinement,* ATISS*-based initialization exhibits semantic and structural artifacts, while our method consistently yields more regular, coherent, and constraint-satisfying floorplans.*

Table 1: *Navigation for more results.*

Constraints	Figure
A gallery of constrained generation	Figure 12
Free generation	Figure 13
Topological graph	Figure 14
Bubble diagram	Figure 15
Box	Figure 16
Box & Topological graph	Figure 17
Box & Bubble diagram	Figure 18
Boundary	Figure 19
Boundary & Topological graph	Figure 20
Boundary & Bubble diagram	Figure 21
Boundary & Front door	Figure 22
Boundary & Front door & Topological graph	Figure 23
Boundary & Front door & Bubble diagram	Figure 24

References

[CPK*17] CHEN, LIANG-CHIEH, PAPANDREOU, GEORGE, KOKKINOS, IASONAS, et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), 834–848 2.

[HHT*20] HU, RUIZHEN, HUANG, ZEYU, TANG, YUHAN, et al. "Graph2plan: Learning floorplan generation from layout graphs". *ACM Transactions on Graphics (TOG)* 39.4 (2020), 118–1 3.

[HZRS16] HE, KAIMING, ZHANG, XIANGYU, REN, SHAOQING, and SUN, JIAN. "Deep residual learning for image recognition". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778 1, 3, 4.

[KW16] KIPF, THOMAS N and WELLING, MAX. "Semi-supervised classification with graph convolutional networks". *arXiv preprint arXiv:1609.02907* (2016) 4.

[NCC*20] NAUATA, NELSON, CHANG, KAI-HUNG, CHENG, CHIN-YI, et al. "House-gan: Relational generative adversarial networks for graph-constrained house layout generation". *European Conference on Computer Vision*. Springer. 2020, 162–177 3.

[NHC*21] NAUATA, NELSON, HOSSEINI, SEPIDEHSADAT, CHANG, KAI-HUNG, et al. "House-GAN++: Generative Adversarial Layout Refinement Network towards Intelligent Computational Agent for Professional Architects". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, 13632–13641 3.

[SHF23] SHABANI, MOHAMMAD AMIN, HOSSEINI, SEPIDEHSADAT, and FURUKAWA, YASUTAKA. "Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, 5466–5475 3.

[SWL*22] SUN, JIAHUI, WU, WENMING, LIU, LIGANG, et al. "Wallplan: synthesizing floorplans by learning to generate wall graphs". *ACM Transactions on Graphics (TOG)* 41.4 (2022), 1–14 3, 4.

[VSP*17] VASWANI, ASHISH, SHAZEER, NOAM, PARMAR, NIKI, et al. "Attention is all you need". *Advances in neural information processing systems* 30 (2017) 1.

[ZSL*20] ZHU, XIZHOU, SU, WEIJIE, LU, LEWEI, et al. "Deformable detr: Deformable transformers for end-to-end object detection". *arXiv preprint arXiv:2010.04159* (2020) 2, 3, 5.

Figure 12: *A gallery of floorplans generated by our method.* ***Left****: Given a floorplan sampled from the dataset (Highlighted in blue), we extract the floorplan's building box, the topological graph, the architectural boundary with a front door, and the bubble diagram as the design constraints, respectively.* ***Right****: Our method generates high-quality vectorized floorplans based on these constraints, relying on a learning-based refinement process rather than heuristic, threshold-based post-processing or complex optimization. From top to bottom, we show generated floorplans of the free generation, box-constrained generation, topological-graph-constrained generation, boundary-constrained generation, and bubble-diagram-constrained generation, respectively.*

Figure 13: *A gallery of floorplans by the free generation. Our method can generate a large range of floorplans without any inputs.*

Figure 14: *A gallery of floorplans by the topological-graph-constrained generation. Our method can generate floorplans with the input of a topological graph.*

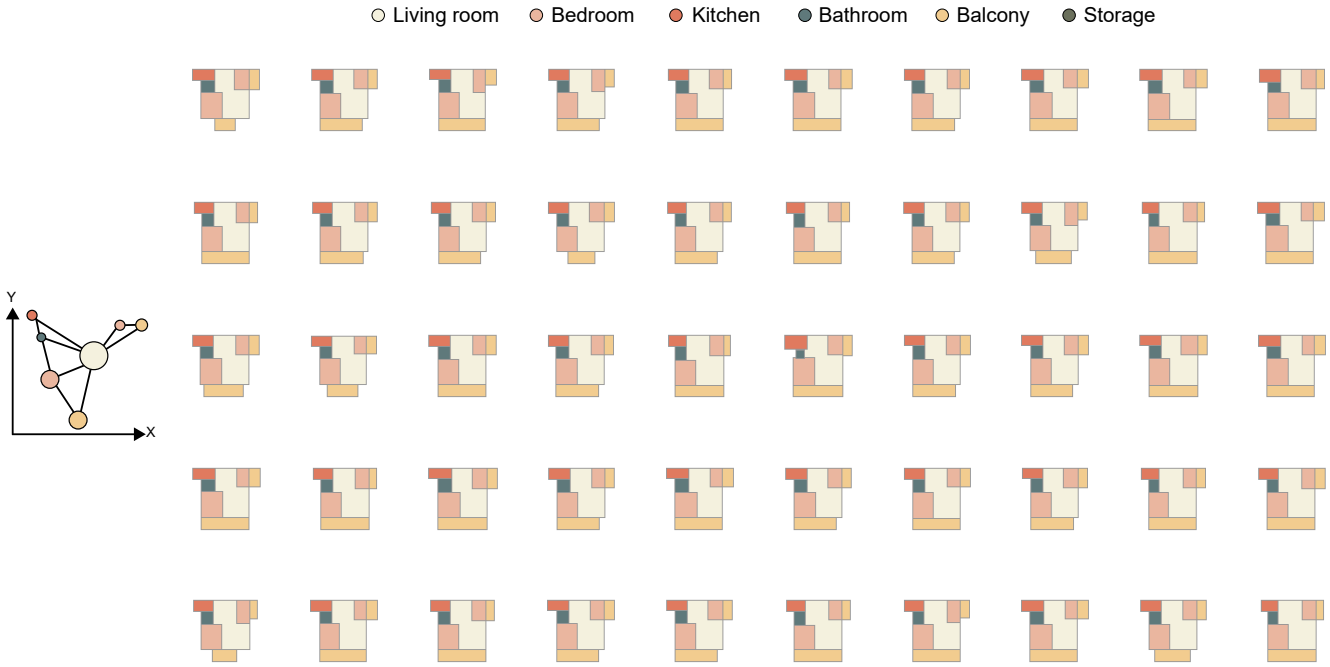


Figure 15: *A gallery of floorplans by the bubble-diagram-constrained generation. Our method can generate floorplans with the input of a bubble diagram.*

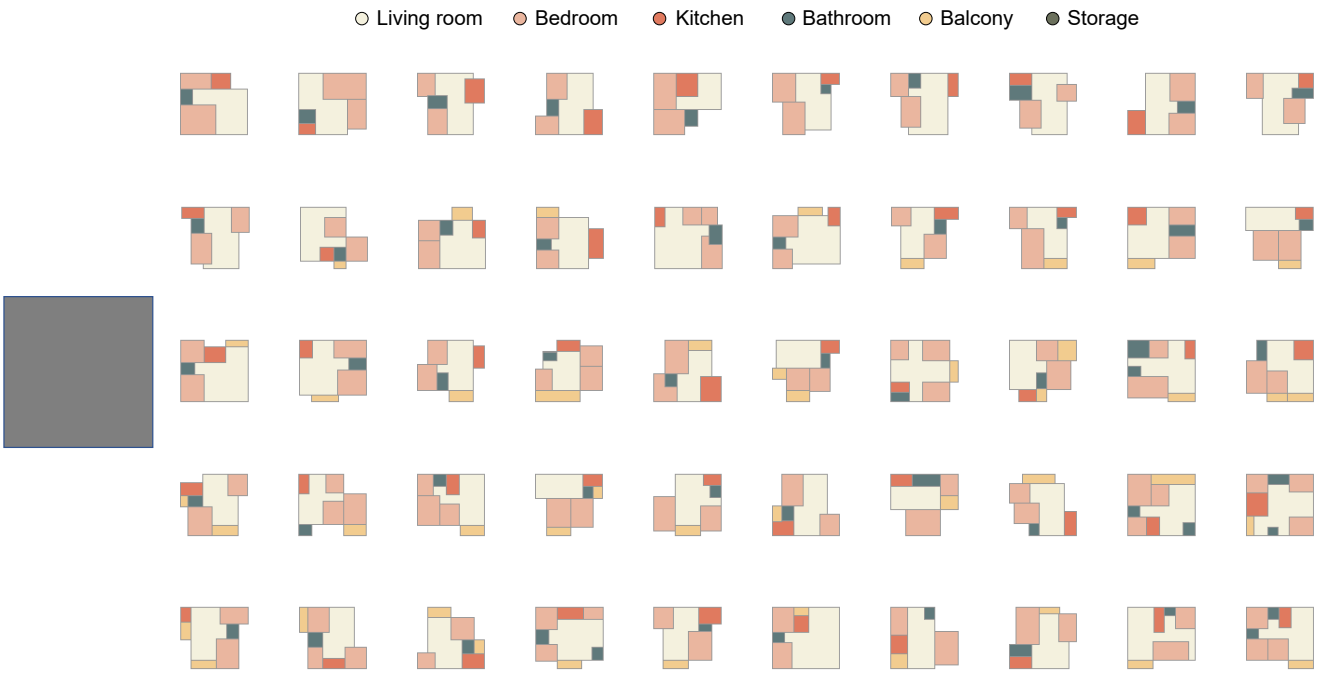


Figure 16: *A gallery of floorplans by the box-constrained generation. Our method can generate a large range of floorplans with the input of a box.*

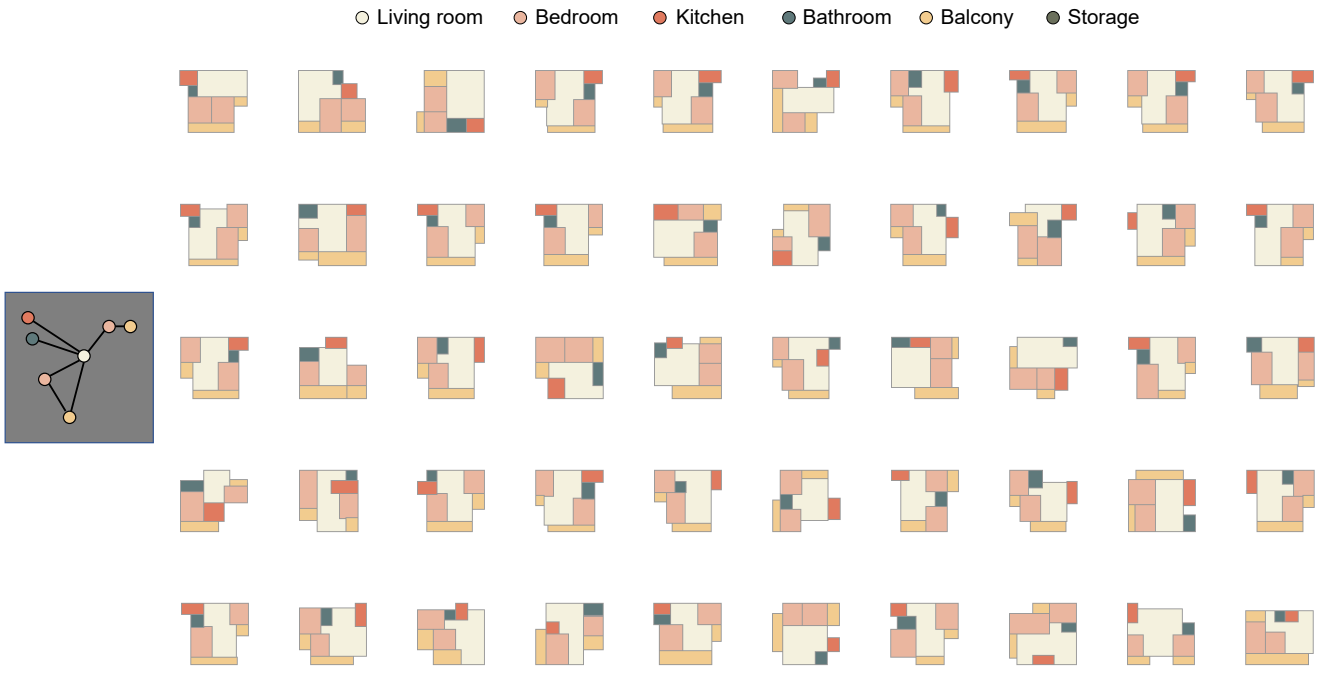


Figure 17: *A gallery of floorplans by the hybrid-constrained generation. Our method can generate a large range of floorplans with the input of a box and a topological graph.*

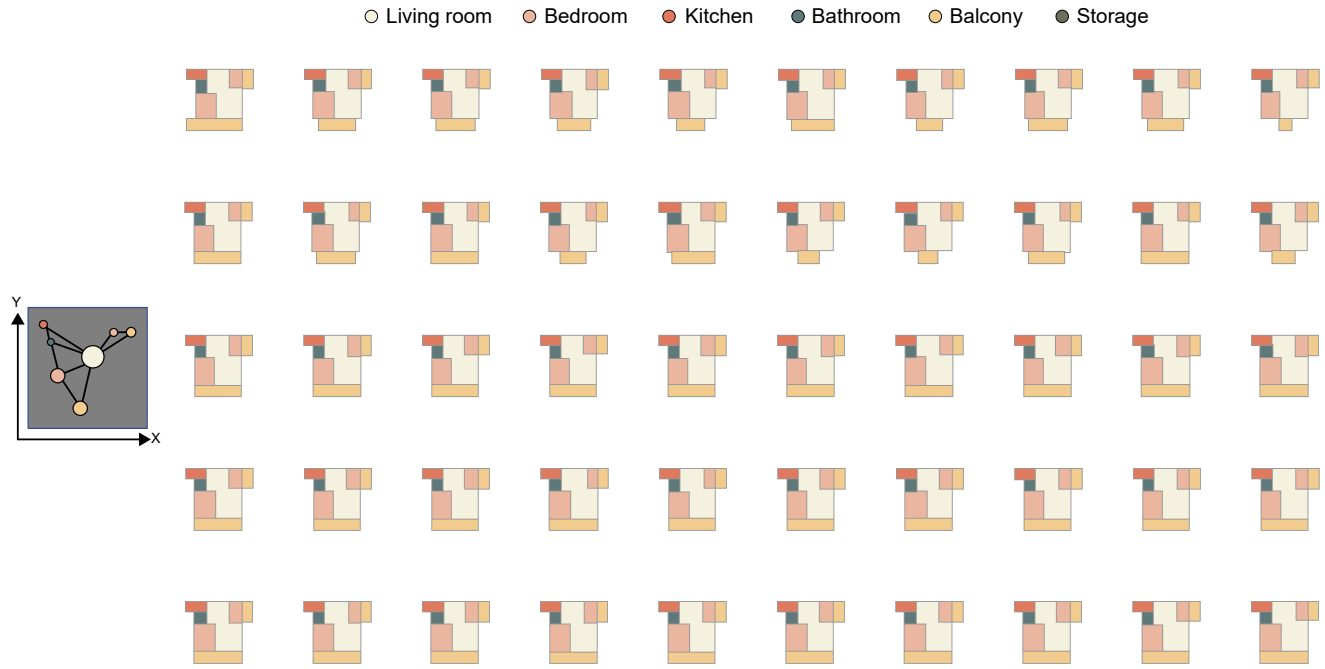


Figure 18: *A gallery of floorplans by the hybrid-constrained generation. Our method can generate a large range of floorplans with the input of a box and a bubble diagram.*

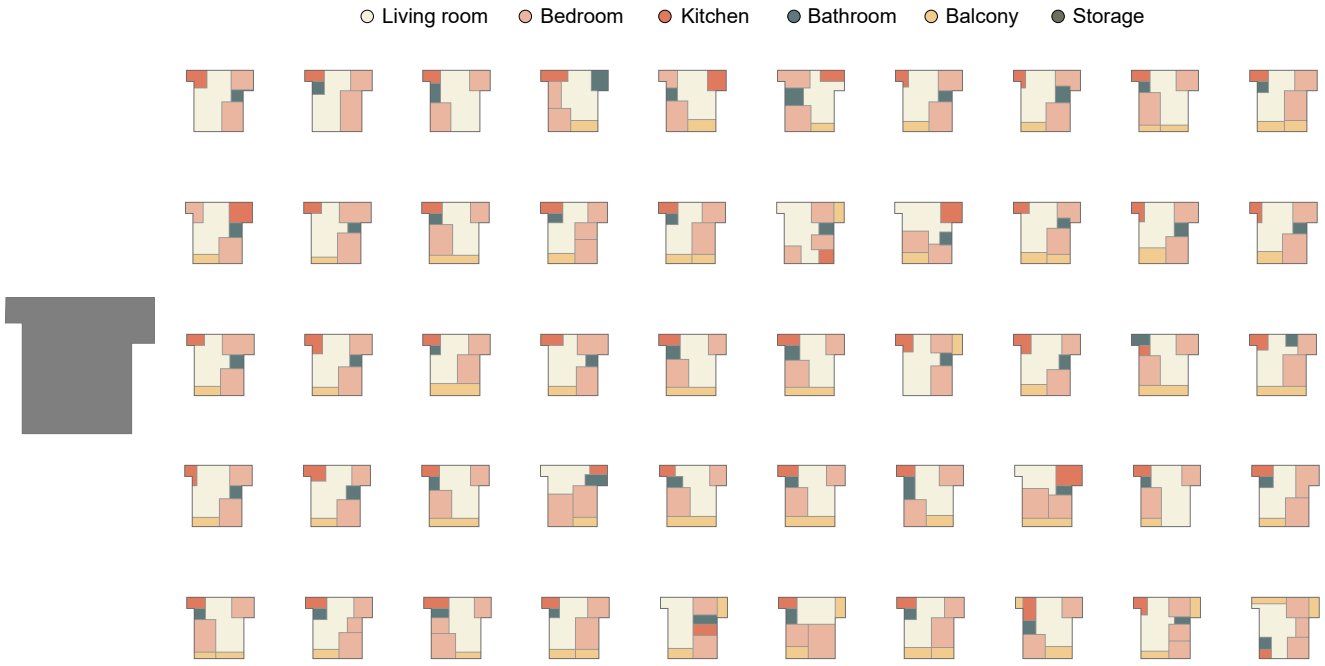


Figure 19: *A gallery of floorplans by the boundary-constrained generation. Our method can generate a large range of floorplans with the input of a boundary.*

Figure 20: *A gallery of floorplans by the hybrid-constrained generation. Our method can generate a large range of floorplans with the input of a boundary and a topological graph.*

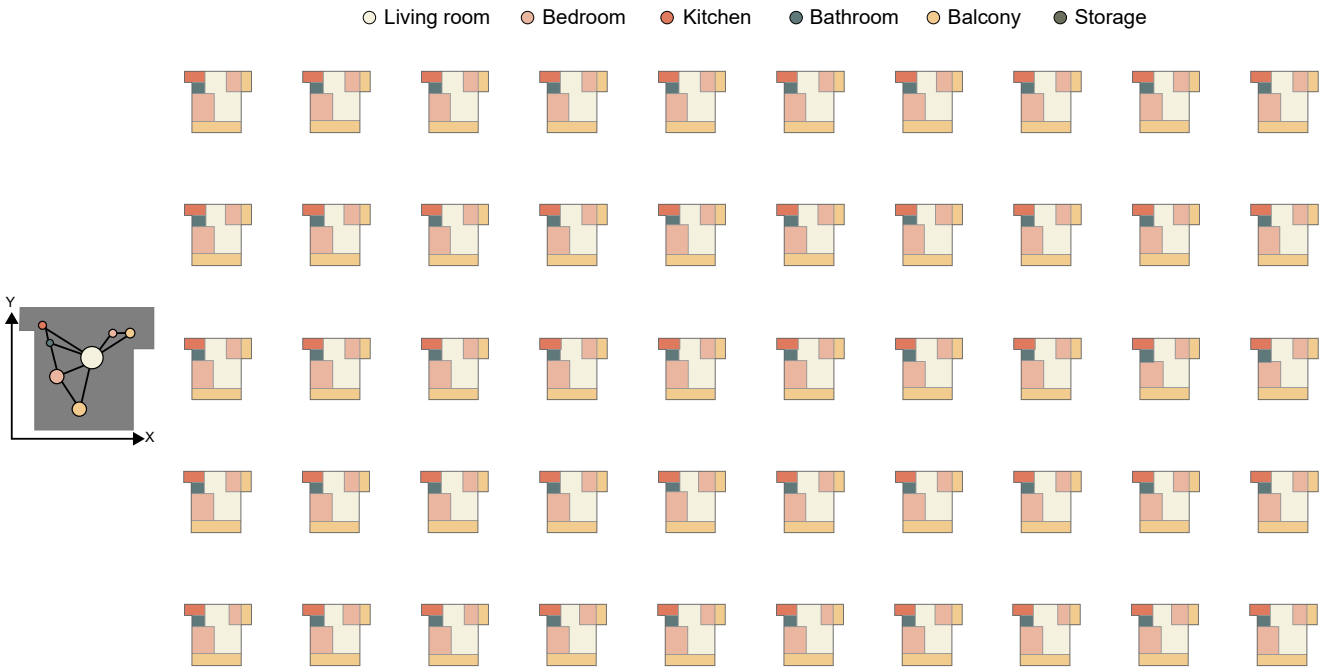


Figure 21: *A gallery of floorplans by the hybrid-constrained generation. Our method can generate a large range of floorplans with the input of a boundary and a bubble diagram.*

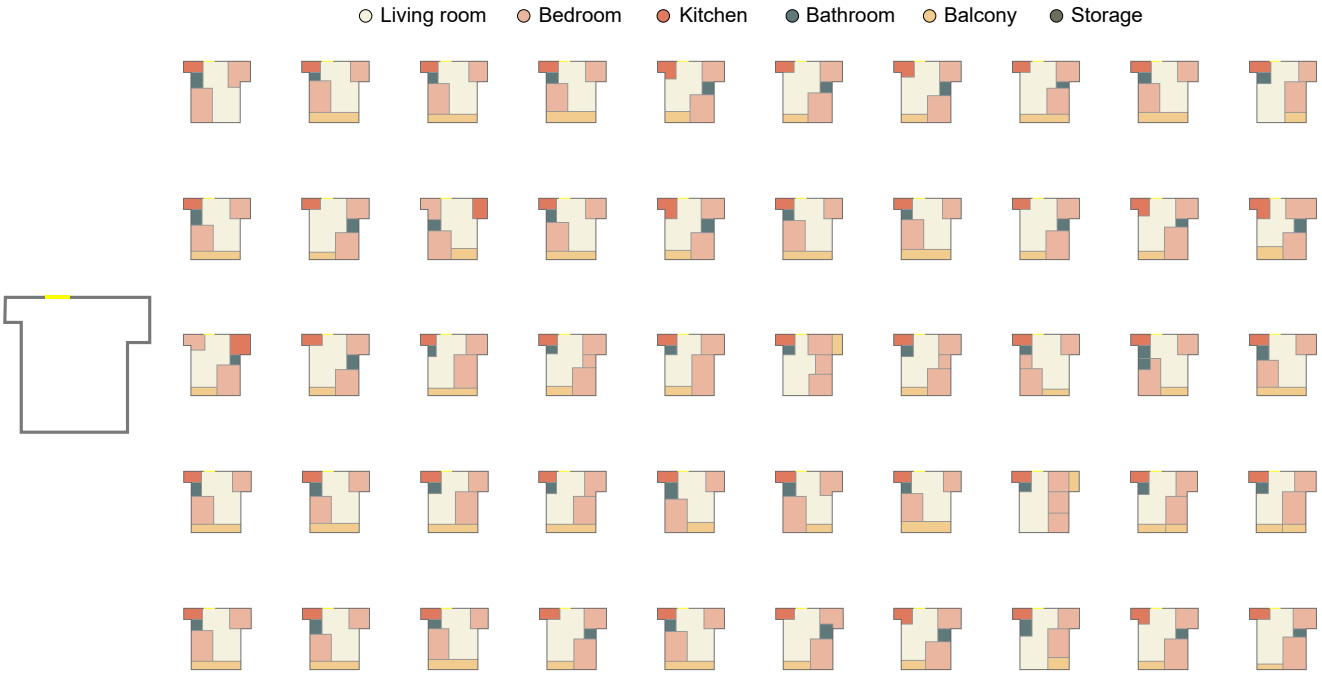


Figure 22: *A gallery of floorplans by the boundary-constrained generation. Our method can generate a large range of floorplans with the input of a boundary and a front door.*

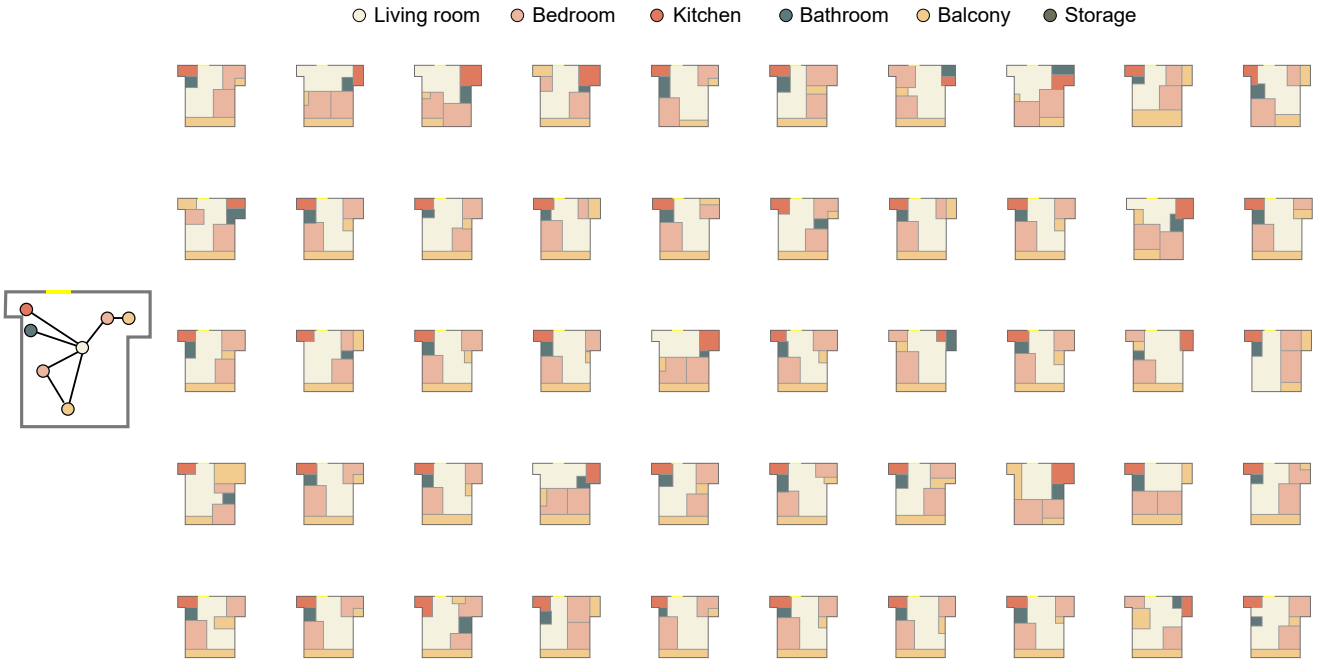


Figure 23: *A gallery of floorplans by the hybrid-constrained generation. Our method can generate a large range of floorplans with the input of a boundary, a front door, and a topological graph.*

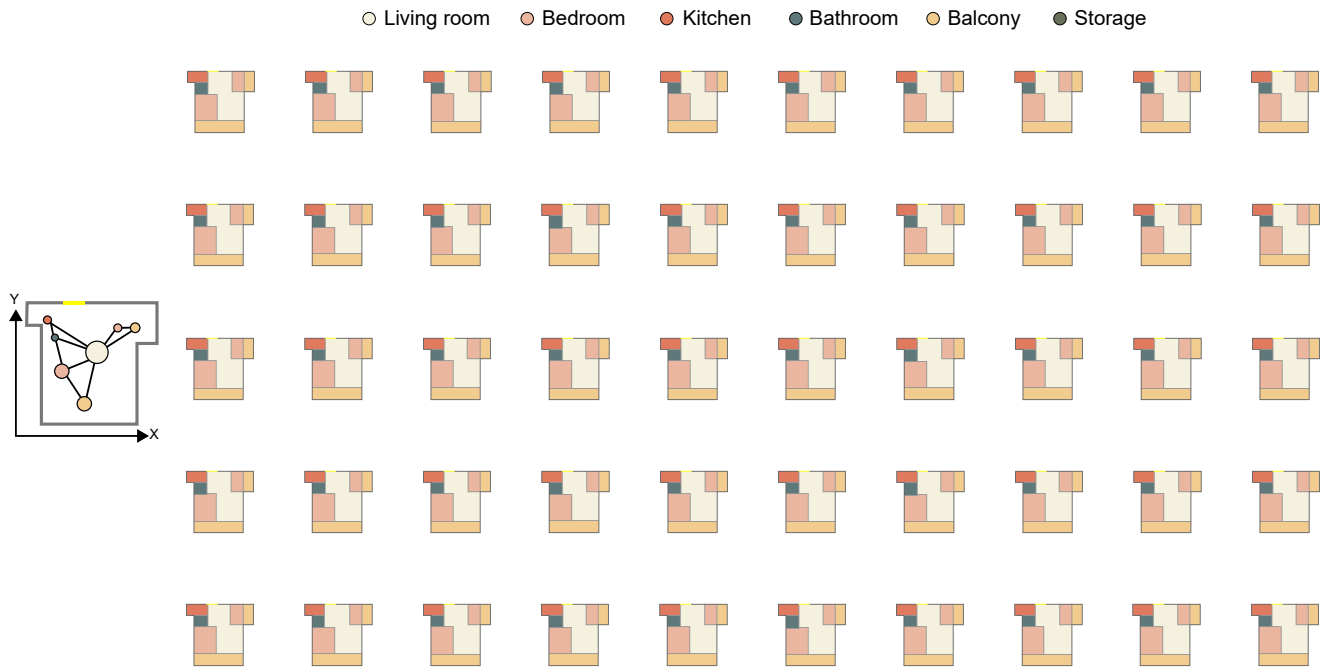


Figure 24: *A gallery of floorplans by the hybrid-constrained generation. Our method can generate a large range of floorplans with the input of a boundary, a front door, and a bubble diagram.*