DOI: 10.1111/cgf.14984
Pacific Graphics 2023
R. Chaine, Z. Deng, and M. H. Kim
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 42 (2023), Number 7

BubbleFormer: Bubble Diagram Generation via Dual Transformer Models

Jiahui Sun Liping Zheng Gaofeng Zhang Wenming Wu †

Hefei University of Technology, China

† Corresponding author. E-mail: wwming@hfut.edu.cn

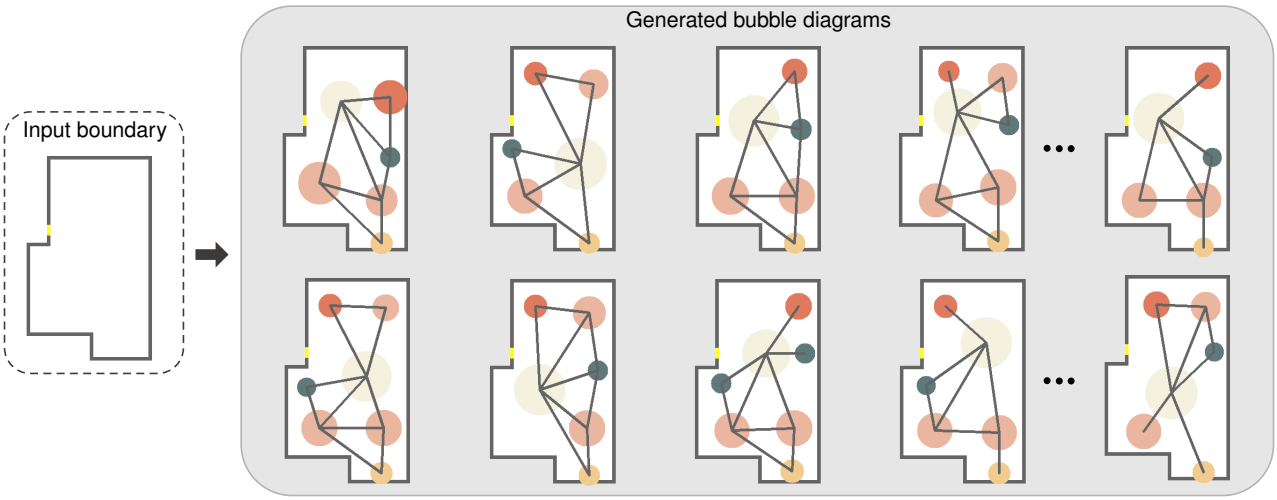


Figure 1: *We propose BubbleFormer, a novel generative model for bubble diagram generation. BubbleFormer takes as input a layout boundary and outputs a large number of reasonable bubble diagrams constrained to the given boundary. These generated bubble diagrams can further be used as input to existing state-of-the-art generative models to produce high-quality layout plans.*

Abstract

Bubble diagrams serve as a crucial tool in the field of architectural planning and graphic design. With the surge of Artificial Intelligence Generated Content (AIGC), there has been a continuous emergence of research and development efforts focused on utilizing bubble diagrams for layout design and generation. However, there is a lack of research efforts focused on bubble diagram generation. In this paper, we propose a novel generative model, BubbleFormer, for generating diverse and plausible bubble diagrams. BubbleFormer consists of two improved Transformer networks: NodeFormer and EdgeFormer. These networks generate nodes and edges of the bubble diagram, respectively. To enhance the generation diversity, a VAE module is incorporated into BubbleFormer, allowing for the sampling and generation of numerous high-quality bubble diagrams. BubbleFormer is trained end-to-end and evaluated through qualitative and quantitative experiments. The results demonstrate that BubbleFormer can generate convincing and diverse bubble diagrams, which in turn drive downstream tasks to produce high-quality layout plans. The model also shows generalization capabilities in other layout generation tasks and outperforms state-of-the-art techniques in terms of quality and diversity. In previous work, bubble diagrams as input are provided by users, and as a result, our bubble diagram generative model fills a significant gap in automated layout generation driven by bubble diagrams, thereby enabling an end-to-end layout design and generation. Code for this paper is at https://github.com/cgjiahui/BubbleFormer.

Keywords: Graph generation; Bubble diagram; Deep generative modeling

CCS Concepts

• **Computing methodologies** → **Shape modeling; Computer vision;**

1. Introduction

A bubble diagram is a graphical representation that uses circles or bubbles to depict spaces, activities, or functions in a layout plan, which is extensively utilized in architectural planning and graphic design. In the computer graphics and vision community, the generative model of graph-to-plan has drawn recent interests, as it is an effective tool for exploring and visualizing spatial relationships and circulation patterns before developing more detailed design solutions. Recently, there have been significant advancements in deep learning for content generation and computer-aided design. Starting with bubble diagrams, state-of-the-art techniques help designers to explore possible solutions for various kinds of space planning, such as floor plans [HHT*20, SHF22], indoor scenes [ZWK19, GSM*23], and graphic layouts [LJE*20].

Traditionally, bubble diagrams are obtained through hand-drawn sketches or computer-aided design software. There is a lack of research efforts focused on bubble diagram generation in the fields of computer graphics and computer vision. Existing graph-driven layout generative models produce plausible plans either with user-specified bubble diagrams [NCC*20, NHC*21] or with synthesized bubble diagrams using heuristic methods [HHT*20]. State-of-the-art techniques do not actually generate creative bubble diagrams that embody the true design intent. More importantly, there is a lack of generation diversity in bubble diagrams which significantly constrains the exploration space of designers.

To circumvent the need for users to provide bubble diagrams, as well as enhance the diversity of bubble diagrams, we aim to develop a generative model for bubble diagram generation. However, the challenges of achieving this goal are twofold. Firstly, it is non-trivial to generate bubble diagrams that are both reasonable and exhibit a certain degree of diversity. Secondly, it is necessary to generate layouts under different constraints to meet the varying requirements of layout generation tasks.

In this paper, we propose BubbleFormer, a novel generative model dedicated to bubble diagram generation. BubbleFormer can directly produce elements in a layout plan, obtain connections between elements, and generate diverse plausible bubble diagrams, while the generated bubble diagrams can serve as the input for downstream layout generation tasks. Dual transformer models are implemented in BubbleFormer which consists of two improved Transformer models: NodeFormer and EdgeFormer. The inputs (e.g., building boundaries) are fed to NodeFormer and EdgeFormer to generate elements and connections of the bubble diagram, respectively. The generated elements and connections are matched and combined to obtain the final bubble diagram. NodeFormer produces element features that are also fed into EdgeFormer. This enables EdgeFormer to better understand and incorporate the element features, allowing it to predict connections between elements. In addition, we aim to provide users with a variety of selectable solutions. Therefore, a VAE (Variational AutoEnceoder) is embedded into BubbleFormer. The goal is to learn the latent space of bubble diagrams, allowing us to sample and generate a large number of high-quality bubble diagrams.

BubbleFormer can be trained and inferred end-to-end. Qualitative and quantitative evaluations demonstrate that BubbleFormer has the capability to generate convincing bubble diagrams, which directly drive downstream tasks of layout generation to produce high-quality layout plans. BubbleFormer can also generate valid bubble diagrams in other layout generation tasks, such as indoor scene synthesis and document layout generation, proving the generalization of our method. In addition, it is shown that BubbleFormer outperforms state-of-the-art techniques in qualitative and quantitative experiments. To the best of our knowledge, BubbleFormer stands as the pioneering generative model that employs an end-to-end learning approach for generating bubble diagrams, effectively filling the longstanding critical gap in automated layout generation driven by bubble diagrams.

2. Related work

Bubble diagrams establish a connection between the designer's intent and solution, leading to a great line of literature about layout design. However, there is relatively little work dedicated to directly generating bubble diagrams for layout design. As follows, we first discuss the application of bubble diagrams in architectural design and indoor scene synthesis, and then introduce some graph generation approaches more closely related to our work.

2.1. Bubble diagram in architectural design

Bubble diagrams are widely utilized in various architectural designs. The role of handcrafted bubble diagrams in the field of architecture design has been extensively discussed in [DG01]. [NRS13] propose a design methodology along with a set of tools for configuring architectural layouts using bubble diagrams. [RMMB22] employ a bybird mehtod of deep learning and agent-based modeling to generate architectural floorplans. [Der12] utilize bubble diagrams to assist the space planning and design of hotels, representing different activity zones and their spatial relationships.

Recently, learning-based approaches are proposed to enable architectural floorplan generation by mapping bubble diagrams to floorplans, achieving state-of-the-art results [CWT*20, HHT*20, NHC*21, PGK*21, SWL*22, SHF22]. However, these works do not actually generate bubble diagrams, which are predetermined or predefined in most state-of-the-art techniques. Graph2Plan [HHT*20] obtains bubble diagrams by retrieving real-world floorplans with bubble diagrams from a dataset. [PGK*21] claim to generate layouts using constraint graphs, while it only generates a set of layout constraints instead of explicit bubble diagrams. In contrast, we aim to directly generate bubble diagrams for architectural design tasks.

2.2. Bubble diagram for indoor scene synthesis

Bubble diagrams have found direct applications in the field of indoor scene modeling. Most of these works have adopted bubble diagrams to represent the indoor scene layouts. [CMS*15, MPF*18, HQZ*18]. [LPX*19] consider the indoor scene layout as a hierarchical graph and employ a VAE module for generation, resulting in the indoor scene synthesis. SceneGraphNet [ZWK19] organizes indoor scene objects into a graph and utilizes message-passing to model the relationships between objects. PlanIT [WLW*19] represents furniture layout using a bubble-diagram-like graph structure

and completes indoor scene generation by constructing a scene relationship graph. [SCW*19] train a multi-layer perceptron to predict a segment affinity graph that describes the indoor scene layout and internal relationships. [LZWT20] learn to generate the indoor scene layout using a scene graph as input, which can be viewed as the topological graph of a bubble diagram. More recently, [CZW*23] propose a graph variational autoencoder with a structured prior for generating the layout of indoor scenes. These works highlight the diverse applications and approaches in utilizing bubble diagrams for indoor scene synthesis. However, similar to architectural design, bubble diagrams as input for indoor scene synthesis relies on strong heuristics for defining relations among indoor objects. Our model can directly create bubble diagrams for modeling indoor scenes.

2.3. Graph generation

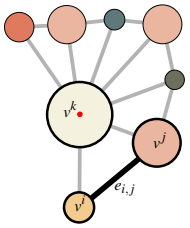
Since bubble diagrams can be abstractly represented as graphs, some generative models of graphs are closely related to our work. [LVD*18] focuse on the topology generation of the graph and proposes a method that generates graphs using a graph neural network in an autoregressive manner. However, for bubble diagrams, spatial information is also very critical for the task of layout design. [YYR*18] employ a deep autoregressive model to decompose graph generation into sequentially generating a sequence of nodes and edges. [LLS*19] produce graphs by generating one block of nodes and associated edges at a time, allowing for a trade-off between sample quality and efficiency by adjusting the block size. MolecularRNN [PSOI19] presents a graph recurrent generative model for the generation of diverse and realistic molecular graphs, trained on a database of molecules. [SWL*22] introduce a new representation of floor plans called wall graph and propose a well-designed coupled structure network for wall graph generation. Graph generation tasks are also utilized for scene understanding and reconstruction. [YLL*18] propose Graph R-CNN for scene graph generation, effectively leveraging object-relationship regularities through object set extraction and relationship measurement. Conv-MPN [ZNF20] introduces a message-passing neural architecture with convolutional neural networks to generate a graph to reconstruct the outdoor building. It is worthwhile that the general reference to a graph will default to a topology structure while the bubble diagram is a specific form of a graph, we emphasize the spatial attributes of its elements, such as position and size. This is also the key distinction between our model and existing graph generation approaches.

While bubble diagrams play a crucial role in layout design, there is still a lack of automated generation methodologies specifically for bubble diagram generation, and that's exactly what we're aiming for in this work.

3. Overview

An overview of our bubble diagram generation framework is given in Figure 2. As follows, we discuss our graphical representation of bubble diagrams, the generation problem, and analysis the challenges to solving this generation problem. Finally, our method for bubble diagram generation is introduced.

Representation. A bubble diagram is a graphical representation that uses circles or bubbles to depict layout elements in a layout plan while using connections to emphasize the topological relations between elements. The bubble contains various information about the category, location, and size of elements, and so on. Therefore, the bubble diagram can be represented as a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, where $\boldsymbol{V} = \{v^k\}_{k=0}^{M}$ denotes a node set and $\boldsymbol{E} = \{e_{i,j} = (v^i, v^j) \mid v^i, v^j \in \boldsymbol{V}\}$ indicates an edge set. Each node $v^k = (c_k, p_k, s_k)$ is a multi-dimensional vector, depicting the element category, central location, and space size, respectively.



Dataset. To address the issue of training data, we extract a large-scale bubble diagram dataset from RPLAN [WFT*19], obtaining more than 60,000 bubble diagrams of real-word floorplans. Specifically, we first use rooms in the floorplan as nodes, extracting information including room category, central location, and room area. Then the edges of the bubble diagram are extracted according to the connectivity between rooms. To our knowledge, this is also the first bubble diagram dataset.

Problem. We aim to develop a generative model capable of producing diverse bubble diagrams while adhering to given constraints. The generated bubble diagrams can offer designers a wide range of solutions. In this paper, we illustrate our method with the bubble diagram generation for floor plans given the building boundary as the generation constraint, as shown in Figure 2. Our method takes a building boundary $\boldsymbol{B} \in \mathbb{R}^{128\times128}$ as input. Given $\boldsymbol{B}$, we aim to design a generative model for $\mathcal{P}(\boldsymbol{G}) = \mathcal{P}(\boldsymbol{V}, \boldsymbol{E} \mid \boldsymbol{B})$.

Challenge. Achieving this goal is non-trivial. On the one hand, generating a diverse range of reasonable bubble diagrams adhering to the given constraints is not straightforward for a generative model, not to mention for an end-to-end framework. On the other hand, topological diversity in bubble diagrams could lead to irrationality, which can be a critical flaw in bubble diagram generation. Therefore, it is essential to generate the topology of bubble diagrams with a careful balance between diversity and rationality.

Methodology. Inspired by DETR [CMS*20] and HOTR [KLK*21], we formulate the bubble diagram generation problem as node generation and edge prediction. The process of generating bubble diagrams from a building boundary involves two steps: generating nodes of the bubble diagram and obtaining edges between nodes. They are implemented by a node Transformer called NodeFormer and an edge Transformer named EdgeFormer. NodeFormer generates nodes for the bubble diagram. It takes the embedding features of the building boundary as input and generates the individual nodes of the bubble diagram. EdgeFormer determines the connections between the generated nodes. This takes the node features generated by NodeFormer and the embedding features of the building boundary as input and predicts the appropriate edges between nodes. Therefore, our generative model BubbleFormer can be represented as

$$\mathcal{P}(\boldsymbol{G}) = \mathcal{P}(\boldsymbol{E} \mid \boldsymbol{V}, \boldsymbol{B}) \mathcal{P}(\boldsymbol{V} \mid \boldsymbol{B}) \tag{1}$$

$$\mathcal{P}(\boldsymbol{E} \mid \boldsymbol{V}, \boldsymbol{B}) = EdgeFormer(Embedding(\boldsymbol{V}), \boldsymbol{B}) \tag{2}$$

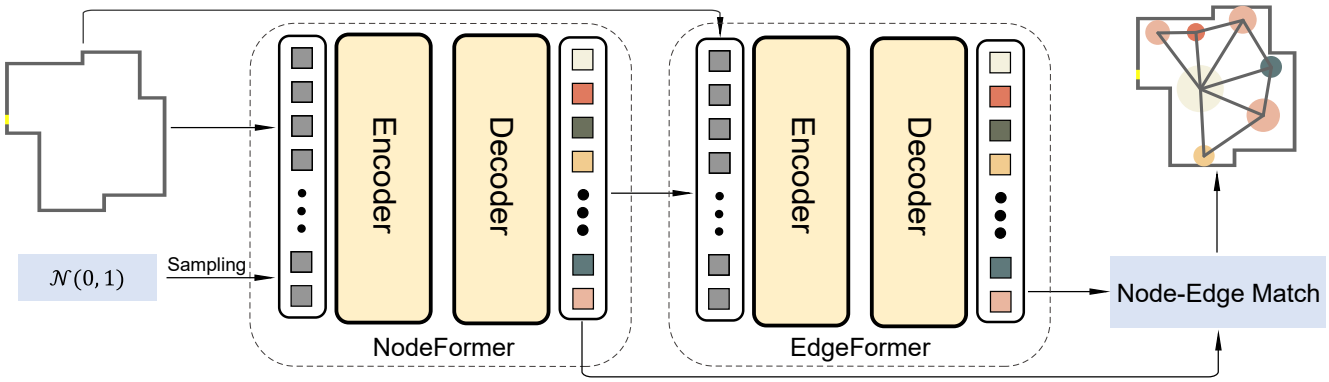


Figure 2: *Architecture of BubbleFormer. The network takes as input a building boundary as well as a sampled noise map from a standard normal distribution. The core module of BubbleFormer is dual Transformer models: NodeFormer and EdgeFormer, which are responsible for generating nodes and edges of the bubble diagram, respectively. The generated nodes and connections are matched and combined to get the final bubble diagram.*

$$\mathcal{P}(\boldsymbol{V} \mid \boldsymbol{B}) = NodeFormer(\boldsymbol{B}) \tag{3}$$

Here $Embedding(\boldsymbol{V})$ is the node features generated by NodeFormer.

As shown in Figure 2, we propose dual Transformer models for bubble diagram generation. The node features outputted by NodeFormer are not only used for node generation but also serve as the inputs to EdgeFormer, providing guidance for connection prediction. Given a specific building boundary, the design of bubble diagrms might not be unique. To this end, a VAE module is embedded into NodeFormer to learn the latent space of bubble diagrams, allowing us to sample and generate a large number of high-quality bubble diagrams. We employ the Hungarian matching algorithm to obtain unique matches between generated nodes and predicted edges, enabling end-to-end training and inference. More details about the methodology can be found in Section 4.

4. Method

Given the boundary of a layout, the node transformer NodeFormer and the edge transformer EdgeFormer of BubbleFormer respectively produce the node set and edge set of the bubble diagram. These two sets are then matched and combined to construct the final bubble diagram.

4.1. Node Generation

Our method to bubble diagram generation is inspired by DETR, a state-of-the-art work in object detection. The Transformer model [VSP*17] is known for its ability to capture long-range dependencies and has been successful in various natural language processing tasks. Therefore, a Transformer model NodeFormer is employed to generate the node set of the bubble diagram. With a given input boundary, there may exist multiple reasonable bubble diagrams. We aim to provide users with a variety of selectable solutions. Therefore, a VAE module NodeVAE is embedded into NodeFormer. By combining NodeVAE and NodeFormer, the model can produce diverse sets of bubble diagram nodes by sampling different latent variables. This provides designers with multiple ideas or solutions for the bubble diagram.

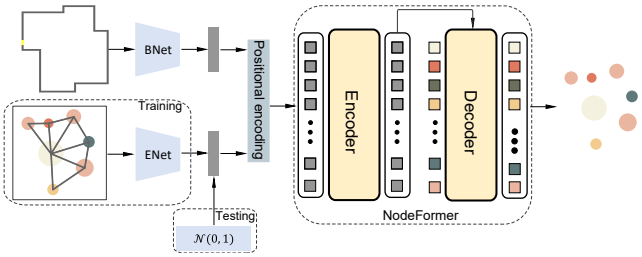


Figure 3: *Node generation. A VAE module is embedded into NodeFormer. In the stage of training, we embed the given boundary with BNet and the ground truth bubble diagram with ENet. These features are fed to NodeFormer after positional encoding. Note that here positional encoding refers to the regular processing of inputs by adding positional information in the Transformer network, which helps to better encode inputs and does not require additional location information from the dataset. The output of NodeFormer can be used to reconstruct the nodes of the bubble diagram. In testing, we directly use a noise map sampled from a standard normal distribution.*

NodeVAE. Given a specific boundary B, there exists a distribution of possible node sets of bubble diagrams. Thus, we propose to learn their distributions $\mathcal{P}(\boldsymbol{V} \mid \boldsymbol{B})$ from the real bubble diagrams. We introduce NodeVAE which learns the distribution of input bubble diagrams, as shown in Figure 3. The key idea is to map the real bubble diagram $\boldsymbol{G}$ to a latent space using an encoder $ENet$, and then mapping a latent variable z in the latent space back to reconstruct the original node set $\boldsymbol{V}$ of the real bubble diagram by a decoder, and here we adopt NodeFormer. In addition, we design an embedding module $BNet$ for the input boundary, which is designed based on a traditional convolutional neural network ResNet [HZRS16].

By feeding *BNet* with $\boldsymbol{B}$, it outputs an embedded boundary feature map γ. Specifically, we represent the input boundary as a three-channel image that stores necessary information about the layout space, which includes the boundary grayscale mask, inside grayscale mask, and the front door grayscale mask. We use specific integers (e.g., 1 to indicate the presence and 0 for absence) to represent different semantics. After passing through *BNet*, we compress the input boundary feature map to a size of 30×30.

Then, $\{\mu, \Sigma\} = ENet(\boldsymbol{G})$ where $\mu \in \mathbb{R}^{32\times32}$ and $\Sigma \in \mathbb{R}^{32\times32}$ are the mean and covariance of a Gaussian distribution. Furthermore, a latent variable z is sampled from $\mathcal{N}(\mu, \Sigma)$ using the reparameterization trick [KW13]. Given z and γ, we reconstruct $\boldsymbol{V}$ by $\boldsymbol{V} = NodeFormer(z, \gamma)$. We employ a standard VAE loss for training. Similar to *BNet*, *ENet* is also designed based on the ResNet, which is sensitive to spatial positions and maps the real bubble diagram to a latent space that approximates a standard normal distribution. *ENet* takes a three-channel grayscale image as input, which stores essential information of the bubble diagram, including the node mask, connect mask, and connect-living-room mask. Specific integers are used to represent specific semantics, similar to *CNet*. During inference, for a new boundary $\boldsymbol{B}$, we sample z from $\mathcal{N}(\mathbf{0}, \mathbf{1})$ and predict $\boldsymbol{V}$ by $\boldsymbol{V} = NodeFormer(z, BNet(\boldsymbol{B}))$.

NodeFormer. The encoding of *BNet* and *ENet* will be fed to the decoder network, NodeFormer. NodeFormer generates node representation for each query, which will be used to produce each node's attributes. NodeFormer adopts the Transformer architecture modified from HOTR [KLK*21] as shown in the supplementary material. NodeFormer adopts non-autoregressive parallel decoding, generating predictions for all nodes in the bubble diagram simultaneously, similar to the previous work DETR [CMS*20] and HOTR. To this end, we employ a non-autoregressive approach to generate nodes of the bubble diagram, significantly improving the generation efficiency of our method. Specifically, we set the number of queries of NodeFormer to 8, which is enough for our task. The spatial feed-forward neural network FNN_s of NodeFormer infers the central location and size of nodes (denoted as v_i^s), while the semantic feed-forward neural network FNN_c infers the node categories (denoted as v_i^c). For the node feature α_i generated from NodeFormer, both the spatial and semantic attributes are obtained through two fully connected layers:

$$v_i^s = FNN_s(\alpha_i) \in \mathbb{R}^3 \tag{4}$$

$$v_i^c = FNN_c(\alpha_i) \in \mathbb{R}^7 \tag{5}$$

During training, we generate all nodes in an end-to-end manner. Therefore, we face a similar problem to DETR, that is, the match between the generated and real nodes. Therefore, we modify Hungarian matching algorithm [Kuh55], which is used in target detection, to make it suitable for our generation task. We employ Hungarian matching algorithm to find the node pairs that minimize the matching loss by considering the similarity between the predicted nodes' locations and categories and the real node set. We then compute the error loss for the matched node pairs. The real node set is denoted as y, and $\hat{y}$ represents the generated node set. For each of the N predicted nodes, we compute the permutation of N combinations $\mathfrak{S}_N$ and calculate the permutation $\sigma \in \mathfrak{S}_N$ that has the lowest matching loss.

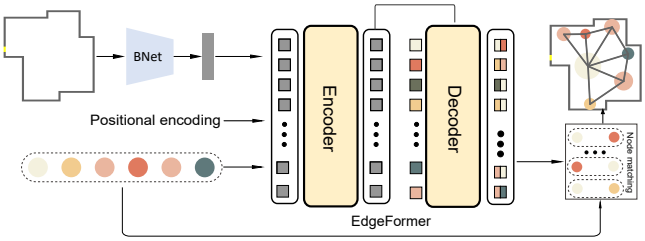


Figure 4: *Edge generation. We do not add a VAE module in EdgeFormer. Besides embedding the input boundary by BNet, we also incorporate the generated embeddings of nodes by NodeFormer as inputs. These features are fed to EdgeFormer after positional encoding. We employ the Hungarian matching algorithm to obtain unique matches between the generated nodes and edges.*

$$\hat{\sigma} = \underset{\sigma\in\mathfrak{S}_N}{\arg\min} \sum_{i}^{N} \mathcal{L}_{\text{match}}\left(y_i, \hat{y}_{\sigma(i)}\right) \tag{6}$$

Here $y = \{(v_i^s, v_i^c), i = 0, 1, ...\}$ and $\mathcal{L}_{\text{match}}\left(y_i, \hat{y}_{\sigma(i)}\right)$ computes the matching loss for a given index i with the real nodes, which takes into account both the room spatial and semantic attributes similarities with GT. We borrow the $\mathcal{L}_{match}$ calculation from DETR. With the index σ obtained after Hungarian matching algorithm, we define the matching loss as:

$$\mathcal{L}_{\text{match}} = -\log \hat{p}_{\sigma(i)}\left(v_i^c\right) + \mathcal{L}_{\text{box}\{v_i^c \neq \varnothing\}}\left(v_i^s, \hat{v}_{\sigma(i)}^s\right) \tag{7}$$

With the index σ, we can obtain the room type probability of $\log \hat{p}_{\sigma(i)}\left(v_i^c\right)$ for calculating loss of cross entropy for binary classification and the predicted box as $\left(v_i^s, \hat{v}_{\sigma(i)}^s\right)$ for calculating the loss of bounding box coordinates $\mathcal{L}_{\text{box}}$. More details can refer to DETR. For the error loss, we used weighted cross-entropy loss for node types and L1 loss for node locations and sizes.

4.2. Edge generation

Edge generation is to produce a coherent topological structure for bubble diagrams. We translate the edge generation problem into edge prediction.

Edge prediction. We have mentioned that it is challenging to achieve variable nodes and stable connection in an end-to-end framework. Topological diversity in bubble diagrams could lead to irrationality. Therefore, we do not add a VAE module in EdgeFormer. This decision is based on the observation that the diversity of the bubble diagram is primarily manifested through the distribution of nodes. Once the spatial positions of the nodes are determined, the arrangement of edges in the bubble diagram is predominantly governed by the node positions. As the plausible solution space for nodes is relatively limited, we aim to minimize the impact of sampled feature maps from the prior distribution on edge generation, thereby reducing the diversity of edge generation while emphasizing its coherence and plausibility. In addition, the sam-

pling space and the sampled hidden vectors of the two VAEs may not coincide, which may generate crashing results.

EdgeFormer. Instead of directly using the generated nodes as input, we incorporate the generated embeddings of nodes by NodeFormer as inputs to the EdgeFormer. This enables EdgeFormer to capture the generated nodes by NodeFormer and generate edges conditioned on them. Simultaneously, EdgeFormer independently encodes the input layout boundary using *BNet*, which generates feature maps that capture boundary information. Taking inspiration from HOTR, we use node pointers in a predicted edge, each specifically indicating a particular node. This approach proves effective in avoiding redundant predictions for nodes that participate in multiple connections, as opposed to making separate predictions for the attributes of each connection. To predict edges, we aim to generate a triplet $(idx1, idx2, e)$, where $idx1$ and $idx2$ are two indices of the nodes generated by NodeFormer. We also introduce a presence flag e for each triple to prevent the prediction of redundant and illogical connection groups. We perform similarity queries between the node pointers and each node feature. The node with the highest score in the query is selected as the indicated node. Similar to NodeFormer, EdgeFormer also adopts the Transformer architecture modified from HOTR, with non-autoregressive parallel decoding, generating predictions for all edges in the bubble diagram simultaneously. Specifically, we utilize the connection feed-forward neural network FNN_{idx1} and FNN_{idx2} to generate the ith connection vectors v_i^{idx1} and v_i^{idx2} from the edge feature β_i outputted by EdgeFormer. This can be expressed as:

$$v_i^{idx1} = FNN_{idx1}(\beta_i) \in \mathbb{R}^d \tag{8}$$

$$v_i^{idx2} = FNN_{idx2}(\beta_i) \in \mathbb{R}^d \tag{9}$$

Here d represents the embedding length of the EdgeFormer and we set $d = 192$. Finally, two output node indices c_i^{idx1} and c_i^{idx2} can be obtained based on the similarity of the generated connection vector and the node feature:

$$c_i^{idx1} = \text{argmax}\left(\text{sim}\left(v_i^{idx1}, \alpha_j\right)\right) \tag{10}$$

$$c_i^{idx2} = \text{argmax}\left(\text{sim}\left(v_i^{idx2}, \alpha_j\right)\right) \tag{11}$$

Here α_j is the node feature generated by NodeFormer and the similarity sim can be calculated by matrix multiplication $\text{sim}(u,v) = u^\top v/\|u\|\|v\|$.

The objective of BubbleFormer is to generate node groups in the form of (N_{idx1}, N_{idx2}, e). The output of the EdgeFormer aims to generate edges in the form of $(idx1, idx2, e)$. These two pointers point to different nodes $\boldsymbol{V}$ generated by NodeFormer. During training, we are provided with the real node group $(N_{idx1}^{gt}, N_{idx2}^{gt}, e^{gt})$. Therefore, in this phase, our goal is to match the generated edge triplets and GT edge triplets, and then the corresponding loss is calculated based on these correspondences, as in HOTR. Specifically, we apply the Hungarian matching to establish one-to-one correspondences between between the GT edges $(idx1^{gt}, idx2^{gt}, e^{gt})$ and the generated edges $(idx1, idx2, e)$. The algorithmic process is shown below:

(a) The pointers of generated nodes $\boldsymbol{V}$ by NodeFormer to GT nodes $\boldsymbol{V}^{gt}$ and the index of the two nodes in the GT edges $\boldsymbol{E}^{gt}$ to the $\boldsymbol{V}^{gt}$ are obtained.

(b) Use the above correspondences to obtain the indices of the nodes in the $\boldsymbol{E}^{gt}$ to the generated nodes $\boldsymbol{V}$.

(c) Following HOTR, with indices from the last step and the output of EdgeFormer, we can obtain the matching cost matrix of the generated edges $\boldsymbol{E}$ and the GT edges $\boldsymbol{E}^{gt}$.

(d) With the cost matrix in the last step, we use Hungarian matching on the matrix from (c) to obtain the indices of the generated edges $\boldsymbol{E}$ and the GT edges $\boldsymbol{E}^{gt}$.

To train EdgeFormerr, we also perform Hungarian matching algorithm as introduced in the node generation for edge matching. After that, we adopt the cross entropy loss to compute the loss of the matched edges. More details can be found in HOTR.

Avoiding redundant prediction. In EdgeFormer, we referenced HOTR as mentioned in the paper. Generating indexed representations of edge triples can avoid redundant prediction. If nodes are predicted directly, the repeated predictions for the same node are not exactly the same, requiring complex post-processing. BubbleFormer generates the node only once and then generates pointers to the node in multiple edges by Hungarian matching, which avoids redundant prediction of nodes. In our experiments, the predicted two triplets rarely correspond to the same node indices, which can also be avoided by the presence flag e.

5. Experiment

We first present the implementation details of BubbleFormer. Then, we conduct various experiments to demonstrate our method.

5.1. Network training

We implemented BubbleFormer using PyTorch and trained our model on an NVIDIA GeForce GTX 3090 GPU. The training data of bubble diagrams is extracted from the RPLAN dataset, with 80% used for training, 5% for validation, and 15% for testing. More implementation details of BubbleFormer are given in the supplementation material. Although our method consists of several subnetworks, including *BNet*, *ENet*, and two Transformer models: NodeFormer and EdgeFormer, these sub-networks can be trained end-to-end thanks to our clever network design and the use of a non-autoregressive generation approach. This means that we can generate all the nodes and edges of the bubble diagram in a single forward pass using the non-autoregressive manner, allowing our method to be trained and inferred end-to-end. Furthermore, the non-autoregressive generation approach exhibits excellent time performance.

5.2. Ablation study

Different from NodeFormer, we do not include a VAE module in EdgeFormer. Instead, we incorporate the generated embeddings of nodes by NodeFormer as inputs to the EdgeFormer after performing dimensionality transformations. We aim to enable EdgeFormer to perceive the node features generated by NodeFormer and use

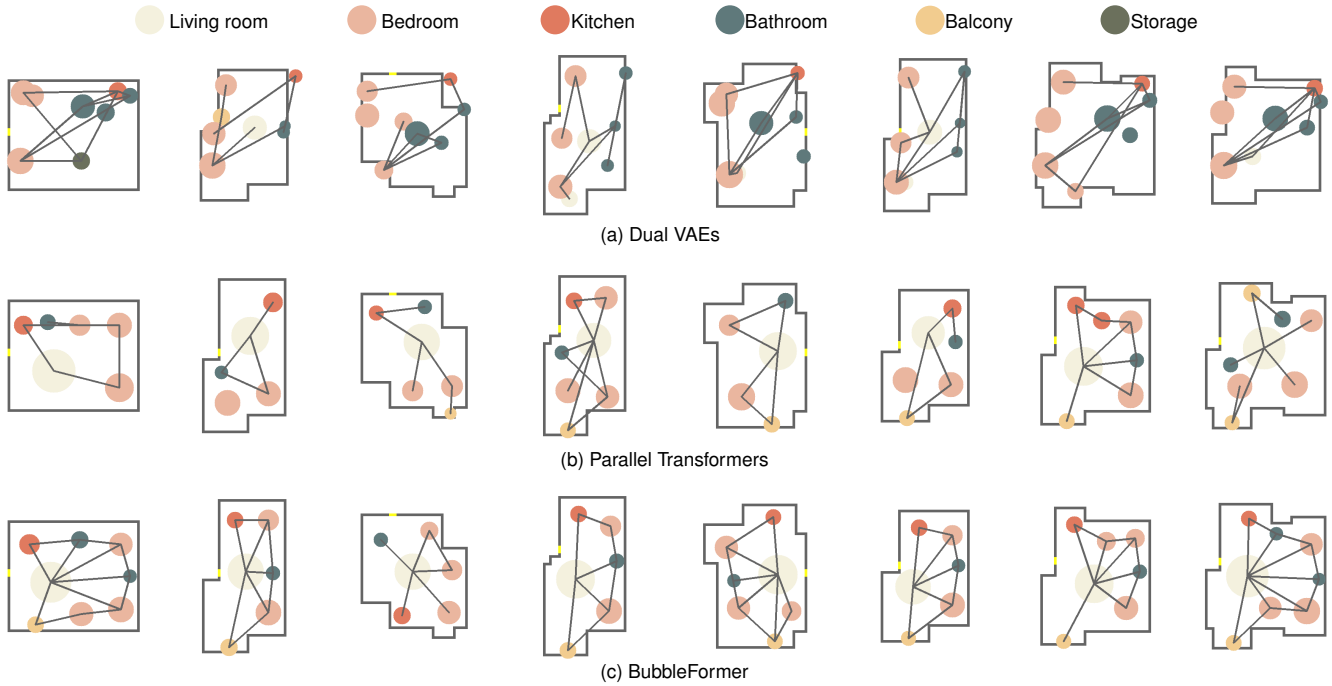


Figure 5: *Ablation studies. To verify the effectiveness of our algorithm design, we have conducted ablation experiments. On the one hand, we add another VAE module into EdgeFormer to form dual VAES, to validate our network design in edge generation, as shown in the first row. On the other hand, we delete the connection of NodeFormer and EdgeFormerthe to form a fully parallel Transformer architecture, as shown in the second row.*

them as conditions to generate plausible edges. Our rationale behind this decision is that the diversity of the bubble diagram is primarily manifested in the distribution of nodes. Once the spatial positions of these nodes are determined, the topological structure can be optimized and finalized. In other words, our objective is to find the optimal solution for the topological structure.

To validate the effectiveness of this strategy, we conduct relevant ablation studies, as depicted in Figure 5. We add another VAE module into EdgeFormer to enable variable edge generation to form dual VAEs (Figure 5(a)). It can be seen that using dual VAEs not only make it difficult to generate plausible connections but also compromise the quality of node generation due to conflicting noise sampling. The conflicting noise map sampled by the two VAE modules makes training challenging and results in inferior results. Figure 5(a) illustrates several semantic issues in the generated bubble diagrams, including the absence of living room nodes. These semantic problems further contribute to topological chaos, making it challenging to establish satisfactory connectivity relationships.

On the other hand, we also modify the network to a fully parallel Transformer architecture of NodeFormer and EdgeFormer (Figure 5(b)), where EdgeFormer can not receive inputs from NodeFormer, and the two Transformer models are completely independent, as HOTR. Compared to the complete BubbleFormer, this ablation experiment exhibits a clear inclination toward generating unstable connections. As shown in Figure. 5(b), although the ablated version generates a relatively reasonable set of rooms. However, there are some problems with the topology of the generated room nodes. In conclusion, the results of both ablation experiments are inferior to the complete BubbleFormer.

5.3. Bubble diagram generation.

Our model can achieve constrained and unconstrained bubble diagram generation.

Diversity generation. Our method can generate bubble diagrams that accommodate different user needs. We sample the results from the output of BubbleFormer to implement constraints on the type and number of rooms, consisting of first sampling the user-requested rooms and then generating other rooms, similar to the rooms constraint operations in RPLAN [WFT*19].

Figure 6 illustrates the results generated with different boundaries and user constraints. We applied the same user constraints on different boundaries in each column. Each column controls the number of bedrooms, bathrooms, and balconies. The results demonstrate the generative capability of our method.

Unconstrained generation. We have observed that some existing layout generative models [NCC*20, NHC*21, SHF23] only use bubble diagrams as inputs. Therefore, we have also tested the ability of BubbleFormer to generate bubble diagrams without any input, allowing our method to drive this type of layout generation approach. In the experiment, we remove the boundary embedding network *BNet* of BubbleFormer. This enables BubbleFormer to generate different bubble diagrams by sampling from the learned latent

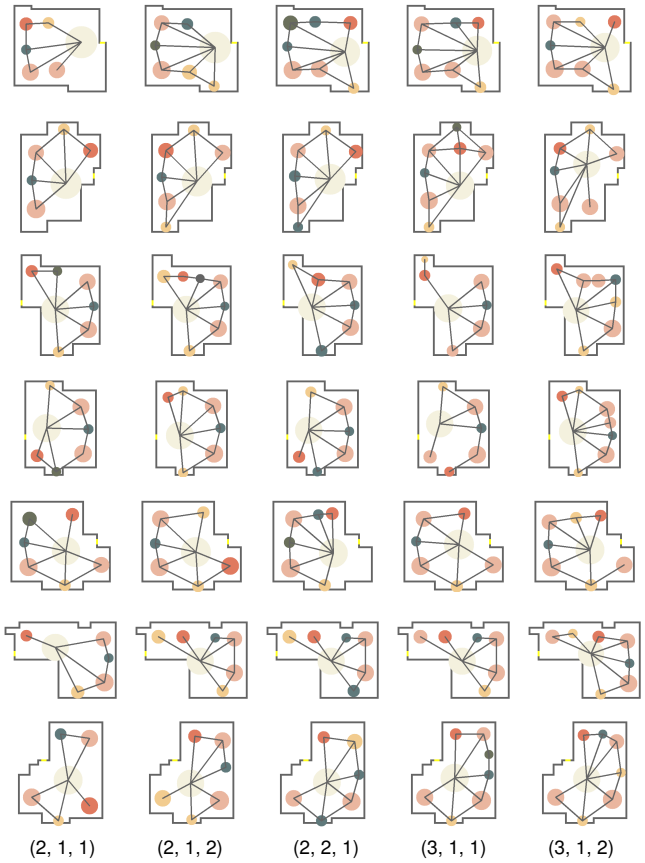


Figure 6: *Constrained generation. We illustrate the bubble diagrams generated with different boundaries and user constraints. The number of bedrooms, bathrooms, and balconies constrained are shown at the bottom.*

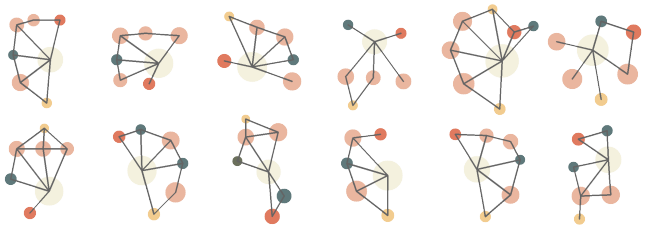


Figure 7: *Unconstrained generation. We have tested the capability of BubbleFormer to generate bubble diagrams without any input.*

space without constraints. As shown in Figure 7, we generate various bubble diagrams without using any constraints, only sampling from the learned latent space.

5.4. Application

BubbleFormer successfully accomplishes the fundamental task of bubble diagram generation. Our method BubbleFormer can further drive floor plan generation and other graphic design tasks. More implementation details are given in the supplementation material.

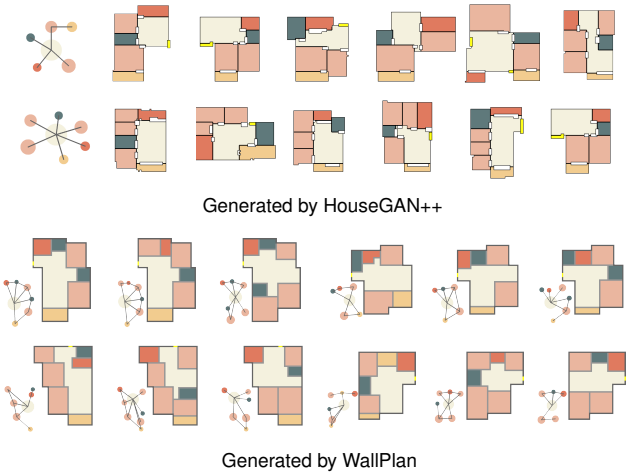


Figure 8: *Bubble diagram driven floor plan generation. Our method can further drive floor plan generation. Our generated bubble diagram can directly drive HouseGAN++ and WallPlan to obtain high-quality floor plans.*

Floor plan generation. We experimentally demonstrate that our generated bubble diagram can directly drive the existing floor plan generation method to obtain high-quality floor plans. We select two state-of-the-art methods. One is HouseGAN++ [NHC*21] which only uses the semantics and topology of the bubble diagram as input, and another is WallPlan [SWL*22] which uses the complete bubble diagram as input. We use the unconstrained generated bubble diagrams as the input of HouseGAN++ and the boundary constrained generated bubble diagrams as the input of WallPlan respectively. The results are shown in Figure 8, where the top shows the results of HouseGAN++, and the bottom shows the results generated using WallPlan. In the results of HouseGAN++, each row shows the generated floor plans with the first input bubble diagram. For the WallPlan, each floor plan is obtained from the bubble diagrams we generated on the left as well as its boundaries.

Graphic deisgn. BubbleFormer exhibits excellent scalability, as we have demonstrated its applicability to a wider range of graphic design tasks. In Figure 9, we expand our method to furniture arrangement and document layout generation. The first row showcases the indoor furniture arrangement results for different room boundaries. In the second row, we present some results of document layout within the same canvas size. For both generation tasks, we only employ a single NodeFormer to achieve their generation in a non-progressive manner. That is we only generate the layout information for each furniture item in the furniture layout or each document element in the document layout.

6. Evaluation

We perform qualitative and quantitative comparisons to comprehensively evaluate our method. In the context of bubble diagrams, the nodes represent spatial elements and their locations and sizes convey important information. Therefore, when evaluating and comparing methods for generating bubble diagrams, it becomes

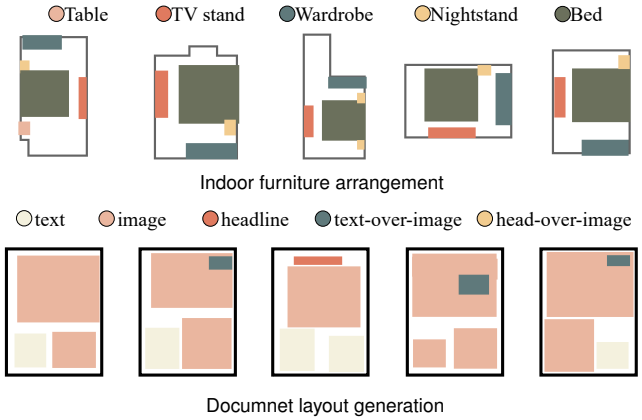


Figure 9: *BubbleFormer can be applied to indoor furniture arrangemen and document layout generation.Top: indoor furniture arrangement of our method; Bottom: documnet layout generation of our method.*

crucial to consider their ability to capture and represent spatial attributes accurately. Since there are currently no existing methods specifically designed for directly generating professional bubble diagrams in the field of architectural design, we have modified existing graph generation methods for comparison and evaluation.

DGMG. We have implemented a graph generation network based DGMG [LVD*18], an autoregressive-based deep generative model for graphs, to directly generate bubble diagrams. We modified the DGMG to enable it to generate bubble diagrams using the layout boundary as input. DGMG utilizes the graph neural network (GNN) to capture the probabilistic dependencies between graph nodes and edges, making it capable of learning the distribution of arbitrary graphs. In the implmentation, we have encode the layout boundary using CNN to accept the boundary image as input. To adress the problem of generating the spatial attributes of bubble diagrams, we added linear layers for predicting node semantics, locations, and sizes.

Retrieval of Graph2plan. Another comparative approach is the retrieval method of Graph2Plan (denoted as Graph2Plan), which utilizes the turning function [ACH*91] of boundaries to represent the characteristics of the boundary and obtains bubble diagrams by retrieving bubble diagrams from a dataset. For comparison, our method selects the bubble diagram of a boundary with the highest similarity of the turning function.

Expert Design. We also request experienced experts to design bubble diagrams for comparing and evaluating our method, denoted as Expert. For the expert design comparison, we selected three graduate students in the College of Architecture and Art, who also have a bachelor's degree in Architecture. The selected designer has strong graphic design experience, especially in the use of bubble diagrams to assist in the generation of floor plans.

6.1. Qualitative evaluation

Dataset boundary constrained generation. We first test bubble diagram generation on dataset boundaries. The comparison results

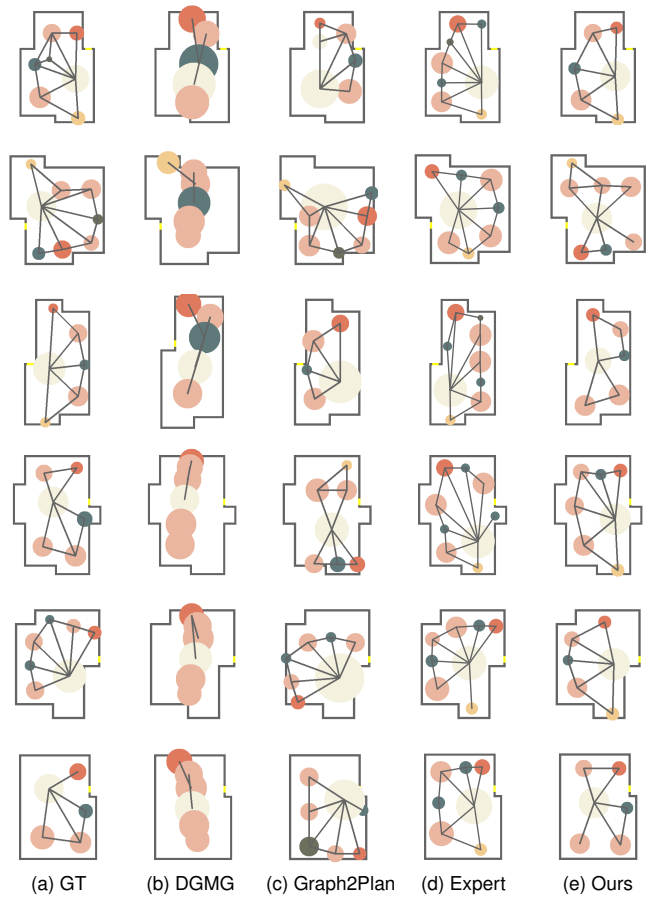


Figure 10: *Dataset boundary constrained generation. We test bubble diagram generation on dataset boundaries.*

can be seen in Figure 10. Given the input boundaries from our bubble daigram dataset, the ground truth bubble daigrams (denoted as GT) are presented in column (a), where column (b), (c), and (d) correspond to DGMG, Graph2Plan, and Expert, respectively. The bubble diagrams geneated by our method are shown in column (e).

BubbleFormer is capable of generating different high-quality bubble diagrams for the same boundary input, thanks to the VAE module that learns a well-defined and continuous latent space. BubbleFormer produces bubble diagrams that are visually closer in quality and style to the ground truth (as seen in the row 1, row 3 and row6). On the other hand, our method is also capable of generating results that are distinctly different from the ground truth (as observed in the row 2, row 4 and row 5). In terms of the generation quality, our method demonstrates a significant improvement over DGMG in terms of visual perception. For example, we can observe that the node sizes in the first row of DGMG's results are irrational, as the living room node does not have a noticeably larger area than the other nodes. Additionally, the node distribution and positioning in the first and second rows of DGMG's results are noticeably unrealistic. In the third result of DGMG, there are evident issues with the room topology, as the living room is not connected to the bedroom and kitchen. It is evident that DGMG, as a

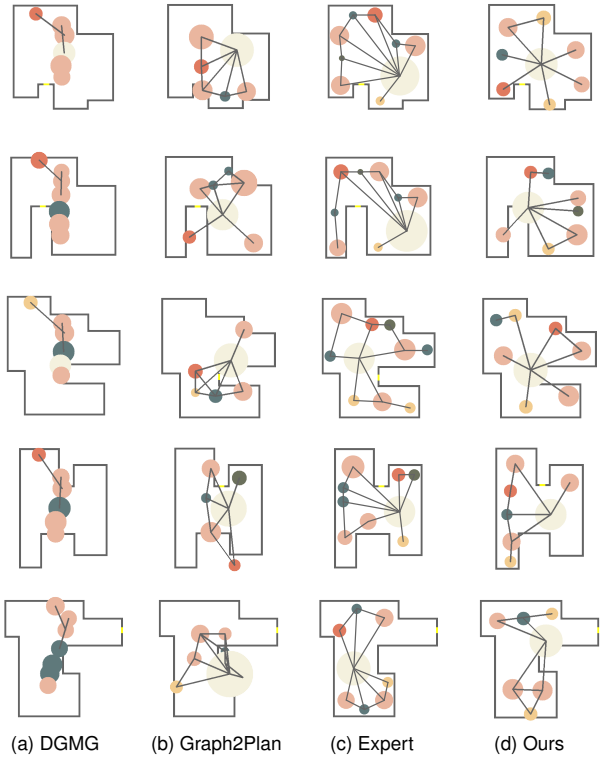


Figure 11: *Hand-drawn boundary constrained generation. We test bubble diagram generation on hand-drawn boundaries.*

topology-based graph generation method, is clearly not suitable for generating bubble diagrams as it produces irrational nodes distribution and topological structures. The bubble diagrams retrieved by Graph2Plan require further post-processing to prevent some rooms from obstructing the front door (see row 1, 2, 3, and 6) or to align the graphics with the boundaries (see row 4 and 5).

For the expert design, it can seen that our results have both similarities and differences with respect to their comparison. For the similarities, the expert design and our method both follow similar design principles, i.e. the rooms are mostly evenly distributed within the boundaries, and most of them are connected to the living room node. The expert's design also contains designs that are similar to (row 1, 3, 5) and different from (row 4, 6) GT respectively. And for the differences, we found that the experts designed a little more rooms than our method, and our number of rooms is closer to GT, due to the fact that we are learning from the GT dataset.

Hand-drawn boundary constrained generation. To demonstrate the validity of BubbleFormer, we introduce hand-drawn boundaries, which are newly created through drawing software. Compared to the real boundaries, the hand-drawn boundaries are simpler, rougher, without design considerations, and exhibit higher variability. The style of hand-drawn boundaries differs from those in the dataset. While the boundaries in the dataset are real architectural boundaries, hand-drawn boundaries exhibit a more arbitrary style. In addition to evaluating our algorithm using boundaries from the dataset, we also attempted algorithm evaluation using hand-drawn boundaries to better assess the generative capacity and generalizability of our method. The comparison results can be seen in Figure 11. Given the hand-drawn boundaries, columns (a), (b) and (c) show the generated bubble diagram of DGMG, Graph2Plan and expert respectively. The bubble diagrams generated our method are shown in column (d).

In the comparison, DGMG generates the least desirable results with a cluttered room distribution and disorganized topological structure. Graph2Plan clearly struggles to retrieve the appropriate boundaries from the dataset. The problem with using retrieval-based methods to obtain bubble diagrams is that if the differences between the boundaries are significant compared to the boundaries in the dataset, it becomes challenging to use the retrieval algorithm to find a bubble diagram that matches the input boundary. The bubble diagrams directly retrieved using Graph2Plan do not match the hand-drawn boundaries well, with some nodes even located outside the boundaries. Such results require further manual adjustments or post-processing algorithms to be usable in subsequent analysis.

The ineffectiveness of DGMG can be attributed to two main reasons. Firstly, the original DGMG was primarily designed for topological graph generation, lacking spatial attention. This prevents DGMG from effectively capturing spatial relationships, which are crucial in bubble diagram generation. Figure 10 clearly illustrates that DGMG tends to generate collapsed nodes. Secondly, DGMG was initially developed for generating topology graphs, which are simpler than bubble diagrams. DGMG is hard to encode complex structures compared to Transformer-based networks.

On the other hand, the results from experts and our method have a higher quality compared to the previous two methods, with the generated bubble diagrams all fitting the boundaries better. Similar to the dataset boundary comparison, this shows that our method is able to outperform DGMG and Graph2Plan methods while approximating the quality of expert designs.

Generalizaion Given the boundary as input, Figure 10 shows our method can generate very different floorplans compared with the ground truth. Our method can generate multiple floorplans from the same input boundary with different input noise maps, as also shown in Figure 6. It also shows that our method explores different possible settings of type and number of rooms, and generates varying bubble diagrams adapting to the corresponding constraints.

To better test the generalization capability, we interpolate two input noise maps and generate bubble diagrams for each interpolated noise map using our method as shown in Figure 12. As the input noise slowly changes, the bubble diagrams also change accordingly. This demonstrates the desirable generalization capability of BubbleFormer.

6.2. Quantitative evaluation

In addition to qualitative evaluations for visual perception of design differences among different generation methods, we have also conducted quantitative experiments to provide a more objective evaluation of the generation quality of BubbleFormer.

FID comparison. FID (Fréchet Inception Distance) is a global metric used to measure the distribution similarity between the

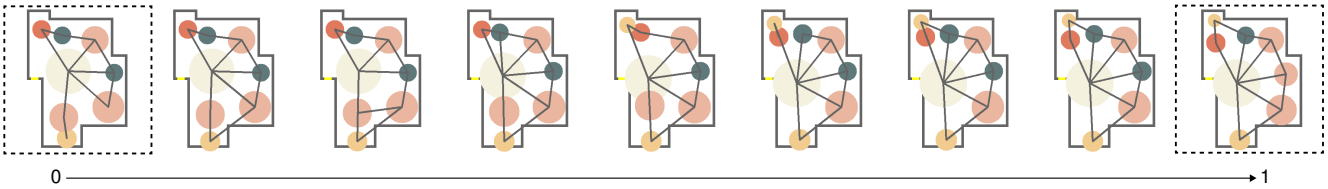


Figure 12: *Interpolation test. BubbleFormer generates bubble diagrams from a set of interpolated noise maps. The first and last ones are generated with two different noise maps sampled from a standard normal distribution, and the intermediate results are interpolated using these two noise maps.*

Table 1: *FID comparison for generated bubble diagrams.*

Method	DGMG	BubbleFormer	Graph2Plan
FID	177.00	16.97	82.64

Table 2: *FID comparison for the generated floor plan with different bubble diagrams input*

Method	DGMG	BubbleFormer	GT
FID	80.18	4.17	0.63

generated results and the real data [HHT*20, NCC*20, PGK*21, SWL*22]. A lower FID score indicates that the generated results are statistically more similar to the ground truth dataset, indicating higher diversity. The FID score is highly dependent on the amount of data, so we have ensured a consistent data volume in our comparisons.

Given the boundary constraint, we have calculated the FID of our model BubbleFormer and the competitor DGMG. For each method, we have generated 9000 bubble diagrams and we compare each generated dataset to the ground truth dataset. The FID result is shown in Table 1. With the same boundary input, the FID score of our generated bubble diagrams is reduced by 90.4% compared with $DGMG^{boundary}$. This quantitative result is also consistent with the performance of our previous qualitative evaluation. This suggests that the graph convolution used by DGMG is clearly not suitable for the generation of spatially informative bubble diagrams. The transformer-based network also has a higher number of parameters to model more complex structural representations than the DGMG. Also compared to Graph2Plan's retrieval method, the FID of our results are also reduced by 79.5%. This is due to the fact that we are comparing bubble diagrams with boundaries. Although Graph2Plan retrieves the GT bubble diagrams, it does not necessarily match that boundary, resulting in a higher FID.

We have experimentally demonstrated that the bubble diagrams generated by BubbleFormer can directly help high-quality floor plan generation. We used the bubble diagrams generated by our method BubbleFormer, DGMG, and the ground truth bubble diagrams as inputs to generate the final floor plans using WallPlan. We have collected a total of 6308 generated bubble diagrams from these methods and calculated their FID scores compared to the ground truth results in the test set. As shown in Table 2, our method achieved a 19.2 times lower FID score compared to DGMG. This

Table 3: *Statistics comparsion. Each statistic is the ratio calculated based on the ground truth floor plans. The ratio close to 1 shows that our method can generate the topology similar to the GT.*

Method	R_{avg}	C^l_{avg}	C^r_{avg}	R^l_{avg}	L^a_{avg}
DGMG	0.895	0.398	0.413	0.536	0.401
BubbleFormer	**0.945**	**0.851**	0.906	**1.001**	**1.026**

significant improvement is attributed to the fact that DGMG struggles to generate reasonably coherent topology and spatial positioning. As WallPlan is sensitive to the quality of the input bubble diagrams, it generates numerous flawed results when fed with DGMG's outputs. However, when compared to the results using the ground truth bubble diagrams as input, our FID score is still relatively high. This is primarily because when the ground truth bubble diagrams are used as input, WallPlan tends to generate layouts that closely resemble the ground truth, resulting in a lower FID score. On the other hand, our method focuses on generating diverse bubble diagrams, which introduces more variation in the resulting floorplans and leads to a higher FID score when compared to the ground truth floorplans.

Statistics comparsion. FID is a metric evaluation that can reflect the similarity between the overall distribution of the generated results and the real, but it does not explicitly reflect the detailed geometrical and topological generation quality for bubble diagrams. Therefore, we have collected some statistical metrics to evaluate the geometry and topology of the generated bubble diagrams.

We have collected some statistics: the number of rooms (denoted as R), the number of rooms connected to the living room (denoted as C^l), the ratio between the number of rooms connected to the living room and the total number of non-living rooms (denoted as C^r), the number of living rooms (denoted as R^l), and the proportion of the living room area (denoted as L^a). We calculate the average of each statistical metric for the test dataset and calculate its ratio with the corresponding data from the ground truth. The comparison results are shown in Table 3. The result reveals that our method is closer to ground truth on all data indicators, while DGMG can only achieve similar results to ours in R. Our method significantly outperforms the baseline method in the other four indicators (C^l, C^r, R^l, L^a) that reflect the topology quality of the generated bubble diagrams.

Perceptual study. To better evaluate from a professional per-

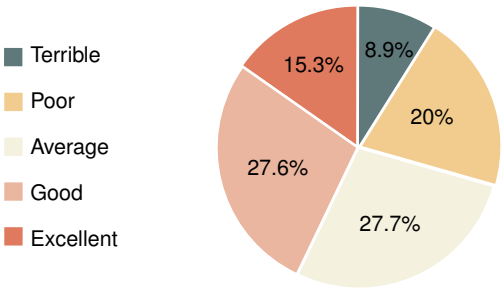


Figure 13: *The result of perceptual study. The results indicate that most of the bubble diagrams generated by our method achieve a high level of quality, and we can provide designers with a variety of inspiring alternative solutions.*

spective, we have also conducted a perceptual study to evaluate the perceived realism of the generated bubble diagrams. We generate the bubble diagrams for the perceptual study using the five floor plan boundaries in the test set. For each input boundary, we generated 10 bubble diagrams, which yield a total of 50 bubble diagrams. For each task, the participants were asked to rate the results on a five-point scale.

We have informed participants of the following basis for scoring: (1) Terrible: The design has significant issues. Major revisions or a complete redesign are needed to reach an acceptable level. (2) Poor: The design has some problems and requires adjustments and improvements to enhance quality and effectiveness. (3) Average: The design meets basic requirements but may need further adjustment to be more competitive or effective. (4) Good: The design excels in multiple aspects, meeting or exceeding general standards. (5) Excellent: The design excels in multiple aspects far beyond general standards and is a highly successful design.

We selected 20 current graduate students with strong graphic design backgrounds in the College of Architecture and Art as subjects for the study. They are also skilled in designing bubble diagrams themselves to assist in the generation of architectural floor plans.

Each participant needs to test a total of 50 tasks, so we count a total of 1000 results for 20 designers. In Figure 13, We show the percentage of scores from "Terrible" to "Excellent". The statistics can reflect that the percentage of results with scores of "Average", "Good", and "Excellent" are 27.7%, 27.6%, and 15.3%. In the opinion of designers with professional design backgrounds, 70.6% of the results achieved a score of three or more. This shows that the majority of our results achieve high quality and that we can provide designers with a variety of inspiring alternatives for the same boundary.

7. Conclusion

For a long time, bubble diagrams have been an important tool for expressing high-level constraints in the fields of floor plan design and architectural design. Many works have introduced the important role of bubble diagrams in architectural design [HHT*20, NCC*20, NHC*21, SWL*22, CWT*20]. However, these bubble diagram-driven methods only use retrieved or user-inputted bubble diagrams. Essentially, they do not generate truly original designs but only complete the transformation from input bubble diagrams to spatial division. This obviously increases the user's usage cost and reduces the generation efficiency.

In view of this situation, we propose a dedicated generation method called BubbleFormer, with the goal of generating diverse bubble diagrams under constrained or unconstrained conditions. To our knowledge, this is the first learning-based approach for generating bubble diagrams, and it complements existing bubble-diagram-driven generation methods by addressing an important missing piece: the generation of its input. We make full use of dual Transformers to model the nodes and edges in bubble diagrams. We use a Hungarian matching algorithm inspired by object detection, which enables end-to-end training and inference for each network. In addition, we have ingeniously embedded VAE into NodeFormer to provide designers and users with a variety of design options. We conduct ablation studies, qualitative evaluations, and quantitative comparisons to fully evaluate the high-quality results. Moreover, we also proved that BubbleFormer can achieve user constraints and support various applications.

Acknowledgments

We would like to thank perceptual study participants for evaluating our system and the anonymous reviewers for their constructive suggestions and comments. This work is supported by the National Natural Science Foundation of China (62102126, 61972128) and the Fundamental Research Funds for the Central Universities of China (JZ2023HGTB0269, JZ2022HGQA0163).

References

[ACH*91] ARKIN E. M., CHEW L. P., HUTTENLOCHER D. P., KEDEM K., MITCHELL J. S.: *An efficiently computable metric for comparing polygonal shapes*. Tech. rep., CORNELL UNIV ITHACA NY, 1991. 9

[CMS*15] CHANG A., MONROE W., SAVVA M., POTTS C., MANNING C. D.: Text to 3d scene generation with rich lexical grounding. *arXiv preprint arXiv:1505.06289* (2015). 2

[CMS*20] CARION N., MASSA F., SYNNAEVE G., USUNIER N., KIRILLOV A., ZAGORUYKO S.: End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16* (2020), Springer, pp. 213–229. 3, 5

[CWT*20] CHEN Q., WU Q., TANG R., WANG Y., WANG S., TAN M.: Intelligent home 3d: Automatic 3d-house design from linguistic descriptions only. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 12625–12634. 2, 12

[CZW*23] CHATTOPADHYAY A., ZHANG X., WIPF D. P., ARORA H., VIDAL R.: Learning graph variational autoencoders with constraints and structured priors for conditional indoor 3d scene generation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (2023), pp. 785–794. 3

[Der12] DEROOS J. A.: Planning and programming a hotel. *The Cornell School of Hotel Administration on hospitality: Cutting edge thinking and practice* (2012), 321–332. 2

[DG01] DO E. Y.-L., GROSS M. D.: Thinking with diagrams in architectural design. *Artificial Intelligence Review 15* (2001), 135–149. 2

[GSM*23] GAO L., SUN J.-M., MO K., LAI Y.-K., GUIBAS L. J., YANG J.: Scenehgn: Hierarchical graph networks for 3d indoor scene

generation with fine-grained geometry. *arXiv preprint arXiv:2302.10237* (2023). 2

[HHT*20] HU R., HUANG Z., TANG Y., VAN KAICK O., ZHANG H., HUANG H.: Graph2plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics (TOG) 39*, 4 (2020), 118–1. 2, 11, 12

[HQZ*18] HUANG S., QI S., ZHU Y., XIAO Y., XU Y., ZHU S.-C.: Holistic 3d scene parsing and reconstruction from a single rgb image. In *Proceedings of the European conference on computer vision (ECCV)* (2018), pp. 187–203. 2

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). 4

[KLK*21] KIM B., LEE J., KANG J., KIM E.-S., KIM H. J.: Hotr: End-to-end human-object interaction detection with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 74–83. 3, 5

[Kuh55] KUHN H. W.: The hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97. 5

[KW13] KINGMA D. P., WELLING M.: Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013). 5

[LJE*20] LEE H.-Y., JIANG L., ESSA I., LE P. B., GONG H., YANG M.-H., YANG W.: Neural design network: Graphic layout generation with constraints. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16* (2020), Springer, pp. 491–506. 2

[LLS*19] LIAO R., LI Y., SONG Y., WANG S., HAMILTON W. L., DUVENAUD D., URTASUN R., ZEMEL R.: Efficient graph generation with graph recurrent attention networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (2019), pp. 4255–4265. 3

[LPX*19] LI M., PATIL A. G., XU K., CHAUDHURI S., KHAN O., SHAMIR A., TU C., CHEN B., COHEN-OR D., ZHANG H.: Grains: Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics (TOG) 38*, 2 (2019), 1–16. 2

[LVD*18] LI Y., VINYALS O., DYER C., PASCANU R., BATTAGLIA P.: Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324* (2018). 3, 9

[LZWT20] LUO A., ZHANG Z., WU J., TENENBAUM J. B.: End-to-end optimization of scene layout. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 3754–3763. 3

[MPF*18] MA R., PATIL A. G., FISHER M., LI M., PIRK S., HUA B.-S., YEUNG S.-K., TONG X., GUIBAS L., ZHANG H.: Language-driven synthesis of 3d scenes from scene databases. In *SIGGRAPH Asia 2018 Technical Papers* (2018), ACM, p. 212. 2

[NCC*20] NAUATA N., CHANG K.-H., CHENG C.-Y., MORI G., FURUKAWA Y.: House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16* (2020), Springer, pp. 162–177. 2, 7, 11, 12

[NHC*21] NAUATA N., HOSSEINI S., CHANG K.-H., CHU H., CHENG C.-Y., FURUKAWA Y.: House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 13632–13641. 2, 7, 8, 12

[NRS13] NOURIAN P., REZVANI S., SARIYILDIZ I.: Designing with space syntax: A configurative approach to architectural layout, proposing a computational methodology. In *eCAADe 2013: Computation and Performance–Proceedings of the 31st International Conference on Education and research in Computer Aided Architectural Design in Europe, Delft, The Netherlands, September 18-20, 2013* (2013), Faculty of Architecture, Delft University of Technology; eCAADe (Education 2

[PGK*21] PARA W., GUERRERO P., KELLY T., GUIBAS L. J., WONKA P.: Generative layout modeling using constraint graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 6690–6700. 2, 11

[PSOI19] POPOVA M., SHVETS M., OLIVA J., ISAYEV O.: Molecularrnn: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372* (2019). 3

[RMMB22] RAHBAR M., MAHDAVINEJAD M., MARKAZI A. H., BEMANIAN M.: Architectural layout design through deep learning and agent-based modeling: A hybrid approach. *Journal of Building Engineering 47* (2022), 103822. 2

[SCW*19] SHI Y., CHANG A. X., WU Z., SAVVA M., XU K.: Hierarchy denoising recursive autoencoders for 3d scene layout prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 1771–1780. 3

[SHF22] SHABANI M. A., HOSSEINI S., FURUKAWA Y.: Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. *arXiv preprint arXiv:2211.13287* (2022). 2

[SHF23] SHABANI M. A., HOSSEINI S., FURUKAWA Y.: Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 5466–5475. 7

[SWL*22] SUN J., WU W., LIU L., MIN W., ZHANG G., ZHENG L.: Wallplan: synthesizing floorplans by learning to generate wall graphs. *ACM Transactions on Graphics (TOG) 41*, 4 (2022), 1–14. 2, 3, 8, 11, 12

[VSP*17] VASWANI A., SHAZEER N., PARMAR N., USZKOREIT J., JONES L., GOMEZ A. N., KAISER Ł., POLOSUKHIN I.: Attention is all you need. *Advances in neural information processing systems 30* (2017). 4

[WFT*19] 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 (2019), 1–12. 3, 7

[WLW*19] 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 (2019), 1–15. 2

[YLL*18] YANG J., LU J., LEE S., BATRA D., PARIKH D.: Graph r-cnn for scene graph generation. In *Proceedings of the European conference on computer vision (ECCV)* (2018), pp. 670–685. 3

[YYR*18] YOU J., YING R., REN X., HAMILTON W., LESKOVEC J.: Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning* (2018), PMLR, pp. 5708–5717. 3

[ZNF20] ZHANG F., NAUATA N., FURUKAWA Y.: 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* (2020), pp. 2798–2807. 3

[ZWK19] ZHOU Y., WHILE Z., KALOGERAKIS E.: Scenegraphnet: Neural message passing for 3d indoor scene augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 7384–7392. 2