Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)
Special Track on AI, the Arts and Creativity

Learn and Sample Together: Collaborative Generation for Graphic Design Layout

Haohan Weng[1*], **Danqing Huang**[2†], **Tong Zhang**[1], **Chin-Yew Lin**[2]
[1]South China University of Technology
[2]Microsoft Research Asia
cswhaohan@mail.scut.edu.cn, dahua@microsoft.com, tony@scut.edu.cn, cyl@microsoft.com

Abstract

In the process of graphic layout generation, user specifications including element attributes and their relationships are commonly used to constrain the layouts (e.g., "*put the image above the button*"). It is natural to encode spatial constraints between elements using a graph. This paper presents a two-stage generation framework: a spatial graph generator and a subsequent layout decoder which is conditioned on the previous output graph. Training the two highly dependent networks separately as in previous work, we observe that the graph generator generates out-of-distribution graphs with a high frequency, which are unseen to the layout decoder during training and thus leads to huge performance drop in inference. To coordinate the two networks more effectively, we propose a novel collaborative generation strategy to perform round-way knowledge transfer between the networks in both training and inference. Experiment results on three public datasets show that our model greatly benefits from the collaborative generation and has achieved the state-of-the-art performance. Furthermore, we conduct an in-depth analysis to better understand the effectiveness of graph condition modeling.

1 Introduction

Graphic layout generation is the process of determining the position and size of each object on a page, which plays a crucial role in creating a successful design (e.g., user interface, articles, presentation slides). It establishes the relationships between elements as well as the overall coherent appearance for better content display. Layout generation is a challenging research topic that dates back to the 1980s when dominant approaches were constraint-optimization-based with heuristic rules or templates [Hurst *et al.*, 2009; Kumar *et al.*, 2011; O'Donovan *et al.*, 2014]. In recent years, an increasing number of works try to tackle this problem with powerful generative models such as Variational AutoEncoders (VAE) [Jyothi *et al.*, 2019; Lee *et al.*, 2020], Generative Adversarial Networks (GAN) [Li *et al.*, 2019; Kikuchi *et al.*, 2021] and AutoRegressive models (AR) [Gupta *et al.*, 2021; Arroyo *et al.*, 2021], which have achieved promising results.

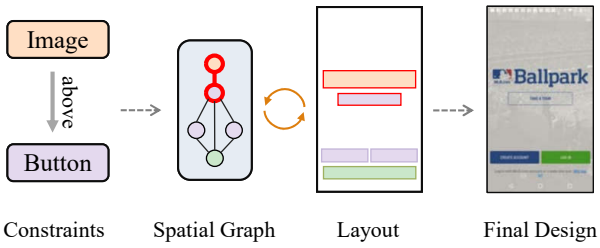

Figure 1: Graphic layout generation with constraints. Given the user specification of element categories and their spatial relationships, our model can generate satisfying layouts.

Considering downstream applications (e.g., novel layout suggestion and layout retrieval), user specifications including element attributes and their relationships are useful preconditions to constrain the generated layouts. For example in Figure 1, given the user specification "*put the image above the button*", the generated layouts need to satisfy such spatial constraint. Nevertheless, most of the current systems fail to handle the relationship constraints. A natural way to encode the relationships is by using a graph, where the nodes represent element categories and the edges represent spatial relationships. The graph condition for layout generation can not only provide users the flexibility to specify requirements, but also serve as an interpretable intermediate representation for better model control.

Recently, Neural Design Network (NDN) [Lee *et al.*, 2020] is an initial attempt to incorporate graphs into layout generation with multiple components in series, including graph generation, layout synthesis and refinement. Each component is dependent on the output of the previous one. NDN uses a training pipeline in which components are separately optimized. In our initial experiment, we observe that the system has a high frequency of generating out-of-distribution graphs which are unseen for the succeeding layout generation component during training. Errors in each component could be propagated and thus decreasing the overall performance (e.g., it is less likely to generate promising layouts if conditioned on

[*]Work done during internship at Microsoft Research Asia.
[†]Corresponding Author.

an unreasonable graph). We will give a more detailed analysis of this issue in the later section.

In this paper, we aim to better utilize graph conditioning in layout generation. We simplify the components in NDN and present a two-stage framework consisting of a VAE-based graph generator followed by an autoregressive layout decoder. To prevent error propagation in the pipeline, we propose a novel collaborative generation strategy to jointly optimize these two generative networks. During training, one network is fed with the output from the other and conducts a round-way knowledge transfer. Similarly in inference, we apply cyclic sampling to refine the layout for several iterations. Additionally, we conduct a comprehensive study of graph conditioning such as graph sparsity, the ratio of user constraints in the graph, and its consistency to generated layouts.

We evaluate our model on three public datasets related to graphic design: RICO [Deka *et al.*, 2017], PubLayNet [Zhong *et al.*, 2019] and InfoPPT [Shi *et al.*, 2022]. Experiment results show that our model outperforms current baselines by a large margin in terms of both quantitative and qualitative evaluations.

In summary, the main contributions of this paper include:

- We in-depth analyze the issue of the separate training pipeline in graph-conditioned layout generation.
- To better utilize the two highly correlated generative networks, we propose a novel collaborative generation strategy that boosts the model performance to the state-of-the-art.
- We conduct extensive studies to verify the effectiveness of graph conditioning. Our model is robust against the input constraint ratio in the graph as well as the sequence order in the autoregressive layout generation.

2 Related Work

2.1 Layout Generation

Early works [Hurst *et al.*, 2009; Kumar *et al.*, 2011; O'Donovan *et al.*, 2014; Tabata *et al.*, 2019] are mostly based on heuristic rules or predefined templates with constraint optimization. These methods usually ensure high-quality outputs but with very limited variations, thus restricting the applications of layout generation in complicated scenarios.

Unconstrained Layout Generation. In recent years, deep generative models have shown great power in learning the complex distribution from given data and generating samples with high fidelity and diversity, such as GAN [Goodfellow *et al.*, 2014] and VAE [Kingma and Welling, 2013]. They have been also adapted to layout generation and achieved promising results. LayoutGAN [Li *et al.*, 2019] applies GAN to synthesize the layout bounding box and proposes a differential wireframe rendering module to enable the training of discriminator, and LayoutGAN++ [Kikuchi *et al.*, 2021] extends LayoutGAN with Transformer backbone. LayoutVAE [Jyothi *et al.*, 2019] trains two VAEs separately, one to predict the element categories and the other to generate the layouts given the category condition. Several methods also follow the VAE-based generative framework [Patil *et al.*, 2020; Lee *et al.*, 2020; Jiang *et al.*, 2022]. Recent works [Gupta *et al.*, 2021; Kong *et al.*, 2021; Arroyo *et al.*, 2021; Jiang *et al.*, 2022] build the generative backbone based on Transformer [Vaswani *et al.*, 2017] to model the long-distance dependency and yield better performance. There are also some content-aware generation methods [Zheng *et al.*, 2019; Wang *et al.*, 2022; Cao *et al.*, 2022; Zhou *et al.*, 2022; Li *et al.*, 2022; Vaddamanu *et al.*, 2022] that further considers the element content into modeling, which we will leave for future exploration.

Layout Generation with Constraints. Spatial constraints are shown to be crucial in this task which allows users to specify the desired spatial relations between elements. Previous works mainly encode the constraints as auxiliary losses or optimization functions. Attribute-conditioned LayoutGAN [Li *et al.*, 2020b] considers element attributes (e.g., area, aspect ratio) and incorporates them by forcing the model to meet attribute conditions with extra training objectives. LayoutGAN++ [Kikuchi *et al.*, 2021] views it as a constrained optimization problem in post-processing. Neural Design Network (NDN) [Lee *et al.*, 2020] is an initial attempt to represent constraints as graph condition and incorporate it into model learning. It is a graph-based system with a separate training pipeline that optimizes different components separately, which raises error propagation in a series of components. In the next sections, we will analyze this issue in more detail and improve the graph condition modeling with a collaborative generation strategy.

As a similar task, related works in floorplan generation commonly use graph to model floorplan [Wang *et al.*, 2019; Hu *et al.*, 2020; Nauata *et al.*, 2020; Nauata *et al.*, 2021; Para *et al.*, 2021; He *et al.*, 2022], exhibiting great capability of graph representation in layout modeling.

2.2 Graph Generative Networks

Graph generative models [Kipf and Welling, 2016; Li *et al.*, 2020a; Hasanzadeh *et al.*, 2019; Zhang *et al.*, 2019] mainly adopt graph convolutional networks to learn the distribution of graphs. One of the most representative models is Variational Graph AutoEncoder (VGAE) [Kipf and Welling, 2016], which embeds each node to a random variable in the latent space and uses an inner-product decoder to generate the adjacency matrix. In this work, we extend the framework of VGAE to multi-class edge matrix prediction and propose a variational graph autoencoder for generating spatial graphs with user constraints.

3 Approach

In this section, we first present our two-stage framework for graph-conditioned layout generation, including a graph generator and a layout decoder. Next, we analyze the issue of the separate pipeline for training the two networks, and further propose our collaborative generation framework.

3.1 Graph-Conditioned Layout Generation

Layout generation can be viewed as a sequence generation of $\boldsymbol{s}$ [Gupta *et al.*, 2021]:

$$\boldsymbol{s} = ([bos], v_1, x_1, y_1, h_1, w_1, ..., v_n, x_n, y_n, h_n, w_n, [eos])$$

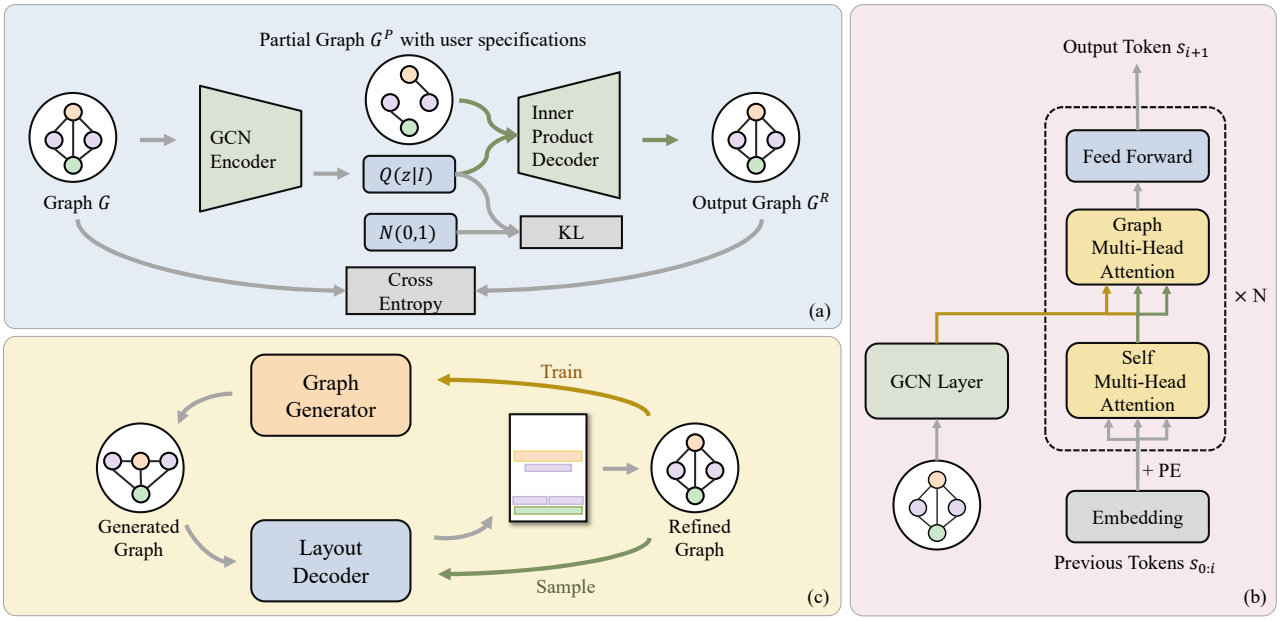

Figure 2: The overview of our approach. (a) Graph generator extends from the variational graph autoencoder; (b) Attention mechanism with graph in the layout decoder; (c) Collaborative generation framework. *These two generative nets are trained collaboratively.* The output of the graph generator is fed to the layout decoder. Layout decoder extract graphs from its generated layouts to guide the graph generator for better generation. In cyclic sampling, our model extracts the graph from the decoded layout and feeds it back to generate new layouts for iterative refinement.

where v_i is the category label of the i-th element in the layout (e.g., `title`, `text`, `figure`), x_i, y_i, h_i, w_i represent the position and size which are converted to discrete tokens. $[bos]$, $[eos]$ are special tokens for beginning and end.

To consider adding user specifications of element categories and their spatial relationships, it is natural to introduce a graph as a condition to constrain the generated layouts. Moreover, a graph serves as an intermediate representation in the generation process to improve the model's interpretability. We define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the node $v_i \in \mathcal{V}$ is the element category and the edge $e_i \in \mathcal{E}$ encodes the spatial relationship between two nodes. The edge can be one of the nine types: *overlap*, *above-left*, *above*, *above-right*, *right*, *below-right*, *below*, *below-left*, *left*. The spatial graph so far is a complete directed graph, which is dense and contains many spatially-true but redundant edges. For example, given *above*(A, B) and *above*(B, C), the edge *above*(A, C) may be not necessary to be stored. This edge redundancy problem will be likely to introduce noise in (1) predicting self-consistent edges during graph generation; (2) capturing the key structural information in graph conditioning. Therefore, we apply a heuristic pruning strategy to obtain a sparse graph: delete the edges which the distance of the two connected nodes is longer than a pre-defined threshold. We show later in the experiment that such a simple pruning strategy works surprisingly well.

Our generation process consists of two networks: (1) graph generator, producing a graph given partial user constraints; (2) layout decoder, synthesizing layouts conditioned on the graph. We briefly describe them in the following.

Network 1: Graph Generator. We extend Variational Graph Autoencoder (VGAE) [Kipf and Welling, 2016] for multi-classes edges generation. In VGAE, a graph is encoded to the latent code z by a GCN, and the edges are generated by the dot product of z. In our model, the latent code z_i for each node is encoded as the concatenation of the input graph embedding by GCN and the node embedding with a learnable embedding matrix. The probability of an edge $e_{ij} \in \mathcal{E}$ belonging to a relation type k between nodes $v_i, v_j \in \mathcal{V}$ can be given by an inner product between the latent of two nodes z_i, z_j:

$$p(\mathcal{E}|\mathcal{V}) = \prod_{i=1}^{n} \prod_{j=1}^{n} p(e_{ij}|z_i, z_j)$$
$$p(e_{ij} = k|z_i, z_j) = \text{softmax}(\rho^k(z_i)^\top \phi^k(z_j)) \quad (1)$$

where ρ^k and ϕ^k is the non-linear transformation corresponding to k. During training, the ground truth of spatial graphs can be extracted from real layouts in the dataset.

Network 2: Layout Decoder. Conditioned on a generated graph from the previous step or an extracted graph from a real layout, our layout decoder autoregressively generates the layout sequence $\boldsymbol{s}$:

$$p(\boldsymbol{s}) = \prod_{i=1}^{5n+2} p(s_i|s_{1:i-1}, \mathcal{G})$$
$$h_i = h_i + \text{GraphAttn}(h_{1:i}, \mathcal{G}) \quad (2)$$

where h_i is the hidden representation of the i-th token. In each decoder layer, tokens will attend to the graph using a cross-attention.

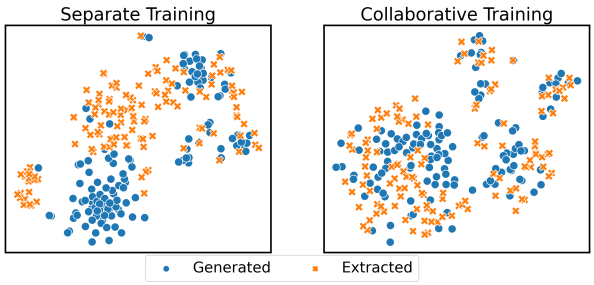

Figure 3: T-SNE visualization of generated and extracted graph embeddings. In separate training, there is a distribution shift between these two types of graphs, resulting in performance reduction using generated graphs. The issue is alleviated in collaborative training.

In our observation, one major limitation of the previous autoregressive models is that the generated results are very sensitive to the input element order [Lee *et al.*, 2020; Gupta *et al.*, 2021]. For example, LayoutTransformer sorts the elements from top left to bottom right as the input order in both training and inference. The performance drops significantly when the order is set randomly. In our experiments, we show that our model is robust against different input order variations, in which the graph condition plays a crucial role.

3.2 Issue of Separate Training Pipeline

Similar to NDN, it is natural to connect the two networks using a separate training/inference pipeline. During training, the graph generator and layout decoder are separately trained with ground-truth extracted graphs from real layouts. Instead in inference, the layout decoder receives generated graphs from the graph generator as input. There exists a gap in the layout decoder's conditioned graphs between training and inference time. In our initial experiment, we observed huge performance reduction when the layout decoder input is changed from the extracted graph to the generated graph. Similar results can also be found in Lee [2020].

To better understand this issue, we visualize the latent space of extracted/generated graphs in Figure 3. As we can see on the left (separate training), the generated graphs have a distribution shift from the extracted graphs. With limited-scale and highly-imbalanced training data, it is difficult to train a fully generalized graph generator and can easily yield out-of-distribution graphs that are likely to be self-inconsistent among edges with increasing node size. Moreover, the layout decoder, only trained with ground-truth extracted graph inputs, is not robust to the unseen generated graphs. Being compared, our proposed learning strategy in the next subsection alleviates the distribution shift of generated graphs as shown in Figure 3 (right), and therefore reduces the error propagation from the graph generator to the layout decoder.

3.3 Collaborative Generation

To resolve the issue mentioned above, we propose a strategy including collaborative training and cyclic sampling for better unifying the graph generator and layout decoder. The overall pipeline is shown in Figure 2.

Collaborative Training. As shown in Algorithm 1, two networks teach each other alternatively during training iterations. On one hand, besides ground-truth extracted graphs, the layout decoder accepts generated graph input from the graph generator to update its parameters. On the other hand, a graph can be derived from the layout decoder output and is used as the training data for the graph generator. This training framework has two major benefits. First, it can be viewed as an approach of curriculum data augmentation, which enforces model robustness to more unseen data. Second, it provides instant communication to perform round-way knowledge transfer between the two networks, and adjusts their learning pace when training together.

Cyclic Sampling. We apply several rounds of decoding in the inference. After the first round of the generation process from graph generation to layout generation, the layout decoder will input the graph derived from its sampling layout in the previous round for iterative refinement. Empirically, we apply the cyclic sampling only on the layout decoder side as it improves sampling time efficiency while performing considerably well.

Algorithm 1 Collaborative Generation.

Require: Target layouts L_t, extracted graphs G_t from L_t
Initialize Graph generator θ_g, layout decoder θ_l
for training iteration $i = 1, \cdots, T_t$ **do**
  Update θ_g with G_t, generate graphs G_g
  Update θ_l with L_t conditioned on either G_t or G_g
  Sample layouts L_g conditioned on G_g with θ_l
  Update θ_g with graphs derived from L_g
end for
for sampling iteration $i = 1, \cdots, T_s$ **do**
  if i == 1 **then**
    Generate graphs G^i with θ_g
  end if
  Sample layouts L^i conditioned on G^i
  Derive G^{i+1} from L^i
end for

4 Experiments

In this section we show our experiment details, system comparison results and the in-depth analysis of graph modeling.

4.1 Experiment Setup

Datasets. We conduct our experiments with three public datasets for graphic design, RICO [Deka *et al.*, 2017] and PubLayNet [Zhong *et al.*, 2019] and InfoPPT [Shi *et al.*, 2022]. **RICO** consists of over 66k unique UI layouts from Android mobile apps. Following previous works, we exclude elements whose labels are not in the 13 most frequent sets and exclude layouts with more than 9 elements. After filtering there are 20,507 layouts in total. **PubLayNet** is a large collection of over 360k scientific documents crawled from PubMed Central. Similarly, layouts with more than 9 elements are excluded, totaling 173,225 layouts in the final set. **InfoPPT** contains 23k information presentations collected from the Internet. We exclude several unnecessary categories such as

Dataset	Rico				PubLayNet				InfoPPT			
Model	Max. IoU	Alignment	Overlap	FID ↓	Max. IoU	Alignment	Overlap	FID ↓	Max. IoU	Alignment	Overlap	FID ↓
LayoutVAE	0.24	0.98	66.20	95.81	0.28	0.63	8.09	58.99	0.15	0.99	61.95	22.03
LayoutGAN++	0.36	0.60	61.19	26.13	0.36	0.24	21.52	25.67	0.09	0.32	127.02	19.01
NDN-none	0.34	0.51	**58.06**	16.86	0.30	0.30	19.38	38.37	0.15	0.74	96.61	36.81
LayoutTransformer	0.21	0.10	71.12	73.73	0.35	0.43	12.05	66.37	0.21	0.36	53.35	**12.99**
Ours-none	**0.44**	**0.18**	67.95	**6.36**	**0.42**	**0.07**	**5.15**	**6.30**	**0.30**	**0.30**	**48.66**	15.34
Real data	0.68	0.27	51.31	1.93	0.53	0.04	0.22	1.78	0.75	0.14	17.20	0.40

Table 1: Overall results of unconstrained layout generation on three datasets. For the metrics Max. IoU, Alignment and Overlap, closer value to the *real data* indicates better system performance. "none" means no user constraint input and the layout decoder is conditioned fully on model-generated graphs.

footnote and decorator, and select layouts with element numbers ranging from 4 to 20, which are in total 46,654 layouts.

Baselines. We consider the following recent works as baselines. **LayoutVAE** [Jyothi *et al.*, 2019] takes the latent code and category labels (optional) as input and generates the element bounding boxes in an autoregressive manner. **LayoutGAN++** [Kikuchi *et al.*, 2021] improves LayoutGAN with Transformer backbone and applies several beautification post-process for alignment and non-overlap. **NDN** [Lee *et al.*, 2020] is a pipeline system with graph generation, layout synthesis and refinement. **LayoutTransformer** [Gupta *et al.*, 2021] autoregressively generates a sequence of element tokens.

Evaluation Metrics. There are 4 metrics commonly used to measure the generated layout quality:

- **Maximum IoU.** Given the generated layouts and the references, this metric computes the intersection over the union of the two sets with a permutation to maximize the IoU as a similarity measurement.
- **Alignment.** Layout elements are usually aligned with each other to create an organized composition. Alignment calculates on average the minimum distance in the x- or y-axis between any element pairs in a layout.
- **Overlap.** It is assumed that elements should not overlap excessively. Overlap computes the average IoU of any two elements in a layout. Layouts with small overlap values are often considered to be high quality.
- **FID.** Compared to the above heuristic metrics, FID is a sample-based metric for image generation [Heusel *et al.*, 2017] and has been adopted in layout generation. It pretrains a feature network to classify real or fake layouts which is then used to extract features of two data sets and calculate the Fréchet distance.

For all the evaluation metrics, we use the implementation from LayoutGAN++[1].

Implementation Details. For model architecture, the graph generator stacks 3 GCN layers with hidden dimension 128 and each node with hidden size 32. The layout decoder contains 6 attention layers each with 8 heads. The attention dropout rate is set to 0.5. The token embedding size is 512. The cyclic step for sampling is set to 2. We train the model for 100 epochs with a learning rate 3×10^{-4} and batch size 64. We use early stopping based on validation error. Adam [Kingma and Ba, 2014] is used as optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

[1]https://github.com/ktrk115/const_layout

4.2 Unconstrained Generation Results

Quantitative Comparisons. For the three heuristic metrics, the closer value to real data, the better performance. For FID score, we pursue the lowest absolute value. Table 1 shows the overall results of unconstrained layout generation. In this setting, our pipeline generates graphs without any constraints to initialize the graph as the same in NDN (labeled as "none"). Our model performs significantly better than the baselines with a large margin on most metrics, especially on the FID. Except that it does not perform well in terms of Overlap on RICO. We argue that elements in UI layouts are more frequently to be overlapped which might encourage the model to learn strong overlap behavior. To highlight, NDN is the most similar approach to ours which also synthesizes layouts conditioned on the generated graph but mainly differs in training strategy. Our model beats NDN across most metrics, indicating the effectiveness of our collaborative generation framework to improve the final performance of generated layouts.

Qualitative Comparisons. As the case study shown in Figure 4, our model generates layouts with better alignment and less overlap. With the aid of spatial graphs, we find that our model can learn to better capture the element relationships. For example, it frequently places icons inside the toolbar as navigation widget in UI layouts (RICO), and text captions on the top of tables while in the bottom of figures in scientific document layouts (PubLayNet).

Ablation of Different Components. We show the results by removing different components of our model in Table 2. All experiments in the following are conducted on PubLayNet. As we can see, coupled with all three components it achieves the best result. Especially with collaborative training, the performance is significantly improved, with FID optimized from 52.70 to 6.64. Graph pruning is also useful, meaning that sparse graphs can help the model better capture the key structural information by removing redundant edges.

Robustness to Input Sequence Order. The order of the input elements is also an important factor in previous autoregressive generation methods. Table 3 shows the result of different element orders on RICO. Without graph condition, the

Figure 4: Qualitative comparisons of generated layouts.

Graph Pruning	Collaborative Training	Cyclic Sampling	Max. IoU	Alignment	Overlap	FID ↓
✗	✗	✗	0.28	0.18	46.36	58.25
✓	✗	✗	0.28	0.21	41.84	52.70
✗	✗	✓	0.35	0.10	18.41	18.85
✓	✓	✗	**0.43**	**0.07**	8.55	6.64
✓	✓	✓	0.42	**0.07**	**5.15**	**6.30**

Table 2: Ablation of different components on PubLayNet.

Graph	Order	Max. IoU	Alignment	Overlap	FID↓
✗	sort	**0.21**	**0.10**	**71.12**	**73.73**
	random	0.20	0.16	81.07	79.83
✓	sort	0.30	**0.16**	**66.99**	23.73
	random	**0.44**	0.18	67.95	**6.36**

Table 3: Effect of different element input order. With graph conditions, the model is more robust to random order.

result is sensitive to the input element order (the random order performs much worse than the sorted order). After adding the graph, our model is robust against the order, with even FID score improved under the random setting. It may probably be the reason that random order serves as an approach to data augmentation with graph conditioning.

4.3 Conditioned Graph Analysis

Here we investigate the impact of different graph settings on the overall performance.

Effect of Input Constraints Ratio in Graph. With a smaller input constraint ratio to initialize a graph, the graph generator has to complete more missing edges, which increases the task difficulty. As shown in Figure 5, when training the graph generator and layout decoder separately (i.e., NDN and Separate), we observe great performance reduction with less constraint input. With our collaborative generation (purple line), the performance becomes stable across different constraint ratios. This indicates that collaborative generation improves the generated graph quality as well as the robustness of the layout decoder, minimizing the performance gap among graphs with different input edge ratios. We also show the performance of LayoutGAN++ with a post-processing optimization of user constraints (red line). Interestingly, it shows a different trend of performance decreasing with a higher constraint ratio, which is expected since more constraints might increase the optimization problem difficulty.

Graph Consistency with Layouts. To evaluate how well the generated layouts conform to the conditioned graphs, we

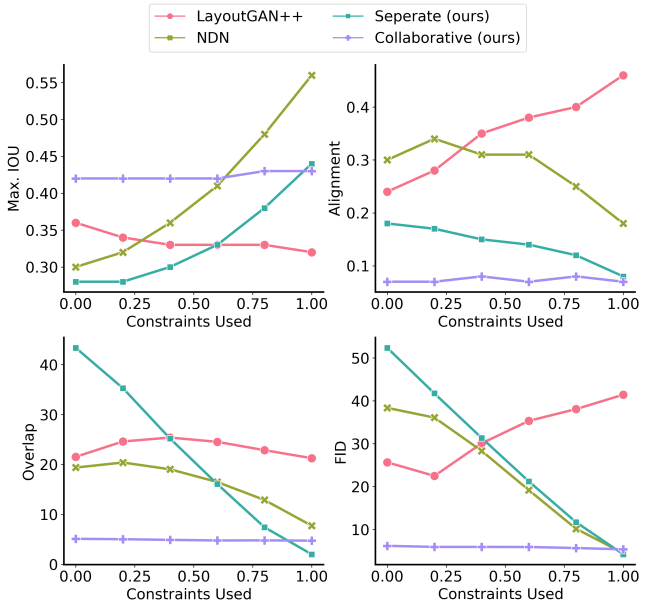

Figure 5: Compared to the unstable separate training, NDN and optimization-based LayoutGAN++, our collaborative generation maintains high performance across different input constraint ratios on all metrics.

Matching Degree (k)	RICO	PubLayNet	InfoPPT
0	55.80	60.61	41.47
1	68.17	79.24	63.38
2	78.17	87.90	80.57

Table 4: Graph consistency with the generated layout.

measure the graph consistency in Table 4. Specifically, given a graph $\boldsymbol{g}$ and the corresponding generated layout $\boldsymbol{l}$ where a graph $\boldsymbol{g'}$ which can be derived from, the consistency value λ can be calculated by matching the edges between $\boldsymbol{g}$ and $\boldsymbol{g'}$:

$$\lambda_k = \frac{1}{|E|} \sum_{e \in E} [|\boldsymbol{e}_g - \boldsymbol{e}_{g'}| \leq k]$$

where the matching degree k indicates the direction closeness between two edges. For example, the degree of <*above*, *above-left*> is 1 and <*above*, *left*> is 2. From the table, we can see that the exact match of the conditioned graphs and generated layouts ($k = 0$) is challenging, with only 40%-60% of samples satisfying this hard matching. While setting loose the degree to $k = 1, 2$, the consistency values can be reached up to around 80%, which indicates that most generated layouts can reasonably conform to the conditioned graph.

Effect of Graph Sparsity. As we mentioned in Section 3.1, complete graphs usually contain redundant edges and are likely to increase the learning difficulty for both the graph generator and layout decoder. Here we investigate the effect of graph sparsity on the overall performance. As shown in Figure 6, pruning the redundant edges to a reasonable

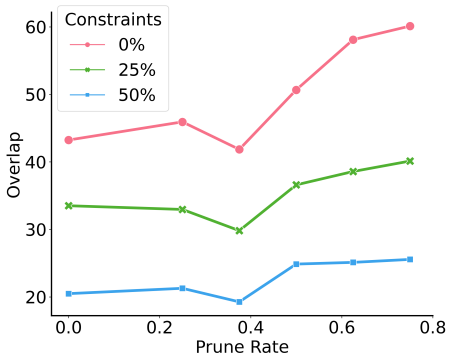

Figure 6: Pruning redundant edges with a reasonable percentage (up to 40%) can improve the quality of generated layouts.

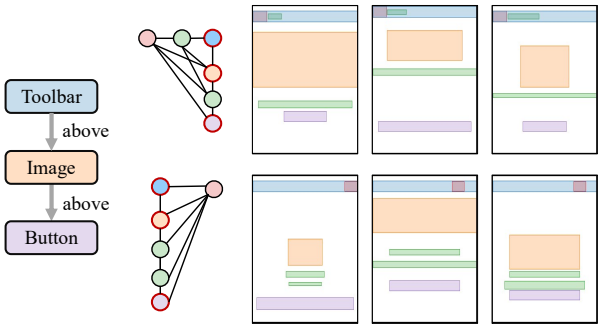

Figure 7: Given the same specification, our model can generate different graphs following the same constraint, accompanied by diverse layouts per graph.

percentage (up to 40%) can improve the quality of generated layouts. As the number of pruning edges continues to get increased, the performance deteriorates, which is expected since there will be more information lost in the pruned graph. This conclusion consistently holds when given different amounts of input constraints (0/25%/50%).

Generation Diversity. Given the user specification, a well-performed system should generate diverse layouts satisfying the same input constraints. As shown in Figure 7, in terms of graph diversity, our model outputs different reasonable spatial graphs. Furthermore, for each conditioning graph, our model generates diverse layouts with varying sizes and positions.

5 Conclusion

User specifications are commonly used to constrain graphic layout generation. In this work, we introduce a graph-conditioned layout generation system that accepts flexible user constraints. Our system consists of a graph generator and a layout decoder. Instead of the separate training pipeline, we propose a novel collaborative generation strategy to better utilize the two networks. Experiment results show the effectiveness of our approach. In the future, we aim for better solutions for generation control. Also, content-aware layout generation would be another promising direction.

Acknowledgements

This work was funded in part by the Major Research plan of the National Natural Science Foundation of China under number 92267203, in part by the National Key Research and Development Program of China under number 2019YFA0706200, in part by the National Natural Science Foundation of China grant under number 62076102, U1813203, and U1801262, in part by the Guangdong Natural Science Funds for Distinguished Young Scholar under number 2020B1515020041, in part by the Science and Technology Major Project of Guangzhou under number 202007030006, in part by the Science and Technology Program of Guangzhou under number 202002030250, in part by The Program for Guangdong Introducing Innovative and Entrepreneurial Teams (2019ZT08X214), in part by Guangdong-Hong Kong-Macao Greater Bay Area Center for Brain Science and Brain-Inspired Intelligence Fund (NO. 2019016).

References

[Arroyo *et al.*, 2021] Diego Martin Arroyo, Janis Postels, and Federico Tombari. Variational Transformer Networks for Layout Generation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13637–13647. IEEE, June 2021.

[Cao *et al.*, 2022] Yunning Cao, Ye Ma, Min Zhou, Chuanbin Liu, Hongtao Xie, Tiezheng Ge, and Yuning Jiang. Geometry Aligned Variational Transformer for Image-conditioned Layout Generation. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 1561–1571, Lisboa Portugal, October 2022. ACM.

[Deka *et al.*, 2017] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 845–854, 2017.

[Goodfellow *et al.*, 2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[Gupta *et al.*, 2021] Kamal Gupta, Justin Lazarow, Alessandro Achille, Larry Davis, Vijay Mahadevan, and Abhinav Shrivastava. LayoutTransformer: Layout Generation and Completion with Self-attention. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 984–994, 2021.

[Hasanzadeh *et al.*, 2019] Arman Hasanzadeh, Ehsan Hajiramezanali, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Semi-implicit graph variational auto-encoders. *Advances in neural information processing systems*, 32, 2019.

[He *et al.*, 2022] Feixiang He, Yanlong Huang, and He Wang. iPLAN: Interactive and Procedural Layout Planning. *arXiv:2203.14412 [cs]*, March 2022. arXiv: 2203.14412.

[Heusel *et al.*, 2017] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[Hu *et al.*, 2020] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. Graph2Plan: learning floorplan generation from layout graphs. *ACM Transactions on Graphics*, 39(4), August 2020.

[Hurst *et al.*, 2009] Nathan Hurst, Wilmot Li, and Kim Marriott. Review of automatic document formatting. In *Proceedings of the 9th ACM symposium on Document engineering*, pages 99–108, 2009.

[Jiang *et al.*, 2022] Zhaoyun Jiang, Shizhao Sun, Jihua Zhu, Jian-Guang Lou, and Dongmei Zhang. Coarse-to-fine generative modeling for graphic layouts. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(1):1096–1103, Jun. 2022.

[Jyothi *et al.*, 2019] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. LayoutVAE: Stochastic Scene Layout Generation From a Label Set. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9894–9903, 2019.

[Kikuchi *et al.*, 2021] Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. Constrained Graphic Layout Generation via Latent Optimization. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 88–96, October 2021. arXiv: 2108.00871.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kingma and Welling, 2013] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[Kong *et al.*, 2021] Xiang Kong, Lu Jiang, Huiwen Chang, Han Zhang, Yuan Hao, Haifeng Gong, and Irfan Essa. BLT: Bidirectional Layout Transformer for Controllable Layout Generation, December 2021. arXiv: 2112.05112.

[Kumar *et al.*, 2011] Ranjitha Kumar, Jerry O Talton, Salman Ahmad, and Scott R Klemmer. Bricolage: example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2197–2206, 2011.

[Lee *et al.*, 2020] Hsin-Ying Lee, Lu Jiang, Irfan Essa, Phuong B. Le, Haifeng Gong, Ming-Hsuan Yang, and Weilong Yang. Neural Design Network: Graphic Layout Generation with Constraints. In *Computer Vision – ECCV 2020: 16th European Conference, 2020, Proceedings, Part III*, pages 491–506. Springer-Verlag, 2020.

[Li *et al.*, 2019] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. LayoutGAN: Generating Graphic Layouts with Wireframe Discriminators. In *ICLR*, January 2019. arXiv: 1901.06767.

[Li *et al.*, 2020a] Jia Li, Jianwei Yu, Jiajin Li, Honglei Zhang, Kangfei Zhao, Yu Rong, Hong Cheng, and Junzhou Huang. Dirichlet graph variational autoencoder. *Advances in Neural Information Processing Systems*, 33:5274–5283, 2020.

[Li *et al.*, 2020b] Jianan Li, Jimei Yang, Jianming Zhang, Chang Liu, Christina Wang, and Tingfa Xu. Attribute-conditioned layout gan for automatic graphic design. *IEEE Transactions on Visualization and Computer Graphics*, 27(10):4039–4048, 2020.

[Li *et al.*, 2022] Chenhui Li, Peiying Zhang, and Changbo Wang. Harmonious Textual Layout Generation Over Natural Images via Deep Aesthetics Learning. *IEEE Transactions on Multimedia*, 24:3416–3428, 2022. Conference Name: IEEE Transactions on Multimedia.

[Nauata *et al.*, 2020] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-GAN: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, volume 12346, pages 162–177. Springer International Publishing, 2020. Series Title: Lecture Notes in Computer Science.

[Nauata *et al.*, 2021] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. House-GAN++: Generative Adversarial Layout Refinement Network towards Intelligent Computational Agent for Professional Architects. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13627–13636. IEEE, June 2021.

[O'Donovan *et al.*, 2014] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. Learning layouts for single-pagegraphic designs. *IEEE transactions on visualization and computer graphics*, 20(8):1200–1213, 2014.

[Para *et al.*, 2021] Wamiq Para, Paul Guerrero, Tom Kelly, Leonidas Guibas, and Peter Wonka. Generative Layout Modeling using Constraint Graphs. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6670–6680. IEEE, October 2021.

[Patil *et al.*, 2020] Akshay Gadi Patil, Omri Ben-Eliezer, Or Perel, and Hadar Averbuch-Elor. READ: Recursive Autoencoders for Document Layout Generation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 2316–2325, June 2020.

[Shi *et al.*, 2022] Danqing Shi, Weiwei Cui, Danqing Huang, Haidong Zhang, and Nan Cao. Reverse-engineering information presentations: Recovering hierarchical grouping from layouts of visual elements. *arXiv preprint arXiv:2201.05194*, 2022.

[Tabata *et al.*, 2019] Sou Tabata, Hiroki Yoshihara, Haruka Maeda, and Kei Yokoyama. Automatic layout generation for graphical design magazines. SIGGRAPH '19. Association for Computing Machinery, 2019.

[Vaddamanu *et al.*, 2022] Praneetha Vaddamanu, Vinay Aggarwal, Bhanu Prakash Reddy Guda, Balaji Vasan Srinivasan, and Niyati Chhaya. Harmonized Banner Creation from Multimodal Design Assets. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7, New Orleans LA USA, April 2022. ACM.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wang *et al.*, 2019] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X. Chang, and Daniel Ritchie. PlanIT: planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics*, 38(4):1–15, August 2019.

[Wang *et al.*, 2022] Yizhi Wang, Guo Pu, Wenhan Luo, Yexin Wang, Pengfei Xiong, Hongwen Kang, and Zhouhui Lian. Aesthetic Text Logo Synthesis via Content-aware Layout Inferring. *arXiv:2204.02701 [cs]*, April 2022. arXiv: 2204.02701.

[Zhang *et al.*, 2019] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. *Advances in Neural Information Processing Systems*, 32, 2019.

[Zheng *et al.*, 2019] Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson W. H. Lau. Content-aware generative modeling of graphic design layouts. *ACM Transactions on Graphics*, 38(4):1–15, August 2019.

[Zhong *et al.*, 2019] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. Publaynet: largest dataset ever for document layout analysis. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1015–1022. IEEE, 2019.

[Zhou *et al.*, 2022] Min Zhou, Chenchen Xu, Ye Ma, Tiezheng Ge, Yuning Jiang, and Weiwei Xu. Composition-aware Graphic Layout GAN for Visual-Textual Presentation Designs. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 4995–5001, Vienna, Austria, July 2022. International Joint Conferences on Artificial Intelligence Organization.