# Coarse-to-Fine Generative Modeling for Graphic Layouts

**Zhaoyun Jiang**[1,3*], **Shizhao Sun**[2],
**Jihua Zhu**[3†], **Jian-Guang Lou**[2†], **Dongmei Zhang**[2]
[1] School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China
[2] Microsoft Research Asia, Beijing, China
[3] School of Software Engineering, Xi'an Jiaotong University, Xi'an, China
jzy124@stu.xjtu.edu.cn, shizsu@microsoft.com, zhujh@xjtu.edu.cn, jlou@microsoft.com, dongmeiz@microsoft.com

## Abstract

Even though graphic layout generation has attracted growing attention recently, it is still challenging to synthesis realistic and diverse layouts, due to the complicated element relationships and varied element arrangements. In this work, we seek to improve the performance of layout generation by incorporating the concept of regions, which consist of a smaller number of elements and appears like a simple layout, into the generation process. Specifically, we leverage Variational Autoencoder (VAE) as the overall architecture and decompose the decoding process into two stages. The first stage predicts representations for regions, and the second stage fills in the detailed position for each element within the region based on the predicted region representation. Compared to prior studies that merely abstract the layout into a list of elements and generate all the element positions in one go, our approach has at least two advantages. First, by the two-stage decoding, our approach decouples the complex layout generation task into several simple layout generation tasks, which reduces the problem difficulty. Second, the predicted regions can help the model roughly know what the graphic layout looks like and serve as global context to improve the generation of detailed element positions. Qualitative and quantitative experiments demonstrate that our approach significantly outperforms the existing methods, especially on the complex graphic layouts.

## Introduction

Graphic design appears almost everywhere, e.g., posters, documents and mobile applications. In achieving a successful graphic design, the *layout*, presented by *positions and sizes* of all the elements on a design, plays a critical role. To aid the creation of graphic layouts, growing interest has been devoted to automatic layout generation. Most studies abstract the layout into a list of bounding boxes and generate layouts by predicting positions of all the elements in one go (Li et al. 2019; Patil et al. 2020; Lee et al. 2020; Gupta et al. 2020; Arroyo, Postels, and Tombari 2021). Specifically, recent work explores generic layout generation by leveraging Transformer, while early studies usually impose restrictions when generating layouts (e.g., using heuristic-based labels for element relationships and handling a limited number of elements). Though meaningful attempts are made, it is still challenging to generate realistic and complex layouts, due to the complicated relationships between elements and the enormous ways that individual elements can be combined into a layout.

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

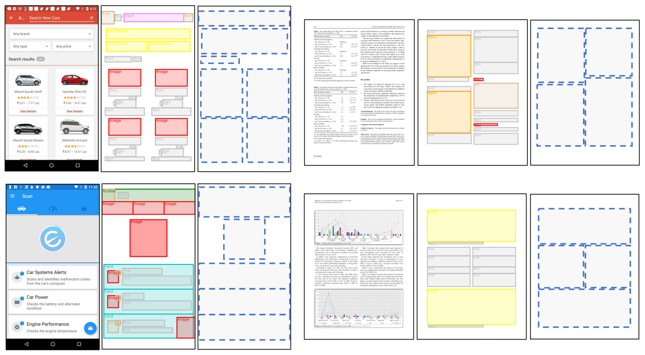


Figure 1: Examples for graphic layouts. Each subfigure contains a graphic design (left), a corresponding layout (middle) and a possible region segmentation (right).

In this work, we propose generating graphic layouts in a coarse-to-fine manner. While most studies consider the entire layout as a single list of elements, our key insight is to segment the layout into several regions, each of which appears like an individual simple layout and contains a smaller number of elements from the whole layout. Figure 1 shows examples for realistic layouts and their possible region segmentations. Specifically, we leverage Variational Autoencoder (Kingma and Welling 2013) as the overall architecture and decompose the decoding process into two stages (see Figure 2(a)). The first decoder, called *region decoder*, predicts a set of region representations based on the latent code. As the region segmentation is not explicitly provided by the layout itself, we propose a simple yet effective method based on grid lines to extract the pseudo region segmentation from the layout and take it as the supervision for the training of the region decoder. The second decoder, called *element decoder*, fills in the detailed position for each element within the region by conditioning on the region representation. To make the model treat each region as a simple layout, element positions are transformed into relative positions to the corresponding region during the training of the element decoder.

There are at least two advantages to the proposed approach. First, compared to generating the entire layout, predicting positions for the elements within a region is easier, as a region contains a smaller number of elements and the element arrangement within a region is relatively simple. Thus, if we could first segment a layout into several regions and then generate element positions within each region individually, a complex layout generation task will be decoupled into several simple layout generation tasks. Second, the regions can roughly depict what a graphic layout looks like. Such information can serve as the global context and help improve the generation of detailed element positions.

We evaluate our approach qualitatively and quantitatively on UI layouts from RICO (Deka et al. 2017) and document layouts from PubLayNet (Zhong, Tang, and Yepes 2019). Experiments show that our approach outperforms existing approaches, especially on complex layouts which have many elements and complicated element arrangements.

## Related Work

**Graphic Layout Generation.** Recently, deep generative models have been studied for graphic layout generation. LayoutGAN (Li et al. 2019) proposes a wireframe rendering layer to capture the alignment characteristic of graphic layouts. NDN (Lee et al. 2020) leverages Graph Convolution Networks (GCNs (Scarselli et al. 2008; Kipf and Welling 2016)) to learn the layout representation, where the labels of relationships are based on heuristics (e.g., top, below and larger). Similarly, READ (Patil et al. 2020) also uses heuristics to determine relationships between elements and then leverage Recursive Neural Networks (RvNNs (Goller and Kuchler 1996)) for layout generation. These studies impose unreasonable restrictions when generating layout. First, LayoutGAN and NDN only handle simple layouts with limited number of elements (e.g., single-column layouts with less than 10 elements), while real layouts contain lots of elements and are much complex. Second, NDN and READ use heuristic-based labels, which are difficult to model element relationships comprehensively and objectively. Recently, to achieve generic layout generation, VTN (Arroyo, Postels, and Tombari 2021) and (Gupta et al. 2020) propose leveraging Transformer (Vaswani et al. 2017) to handle arbitrary number of elements and discover element relationships without heuristic-based labels. All the above prior studies regard the entire layout as a single list of elements and generate element positions in one go. Unlike them, this work considers segmenting a layout into several regions and decomposing the generation process into two stages.

Besides, some studies also explore how to incorporate user intents or constraints when generating layouts. Zheng et al. studies generating layouts conditioned on visual and textual semantics of the user input. Lee et al. synthesis layouts by considering user-specified position and size relationships. Kikuchi et al. formulate the layout generation as a constraint optimization problem to satisfy implicit and explicit constraints specified by users. Moreover, a recent study leverages multi-modal set of attributes for canvas and elements to help layout generation (Yamaguchi 2021). These works are orthogonal to the unconditional layout generation discussed in this work.

**Vector Image Generation.** Vector image generation, e.g., sketches, strokes and icons, catches attention until very recently, despite that raster image generation has achieved great success (Radford, Metz, and Chintala 2015; Zhu et al. 2017; Arjovsky, Chintala, and Bottou 2017). For example, SketchRNN (Ha and Eck 2017) models all strokes in a sketch as a sequence; Sketchformer (Ribeiro et al. 2020) leverages Transformer to learn longer term temporal structure in the stroke sequence; DeepSVG (Carlier et al. 2020) disentangles high-level shapes from the low-level commands to reconstruct complex icons; and CoSE (Aksan et al. 2020) factors local appearance of a stroke from the global structure of the drawing to model stroke-based data. As graphic layouts have different data structures with aforementioned vector images, recent progress on them cannot be directly adopted. Nevertheless, these studies inspire us to think deeply about segmenting graphic layouts into several regions, which is overlooked by existing studies about graphic layout generation.

## Problem Formulation

In this work, we aim at synthesizing a plausible graphic layout, denoted as $\mathbf{x}$. Concretely, a graphic layout is composed of elements, i.e., $\mathbf{x} = \{x_1, \ldots, x_N\}$, where $N$ is the number of elements and $x_i$ stands for the placement of the $i$-the element. Here $x_i = (s_i, t_i)$, where $s_i$ represents the left-top and right-bottom coordinates of element's bounding box and $t_i$ is the element type (e.g., buttons, texts and images).

Moreover, as introduced in Section , we segment a layout into several regions, each of which appears like a simple layout and contains a set of elements. We denote the regions on a layout $\mathbf{x}$ as $\mathbf{r} = \{r_1, \ldots, r_M\}$, where $M$ is the number of regions and $r_j$ stands for the representation of the $j$-the region. In this work, we represent a region by two kinds of information, i.e., where the region is and which elements are in this region. Thus, we have $r_j = (u_j, v_j)$, where $u_j$ represents the left-top and right-bottom coordinates of the minimal enclosing rectangle for all the elements within this region, and $v_j$ is a vector, each dimension of which represents the number of one element type appearing in this region.

## Approach

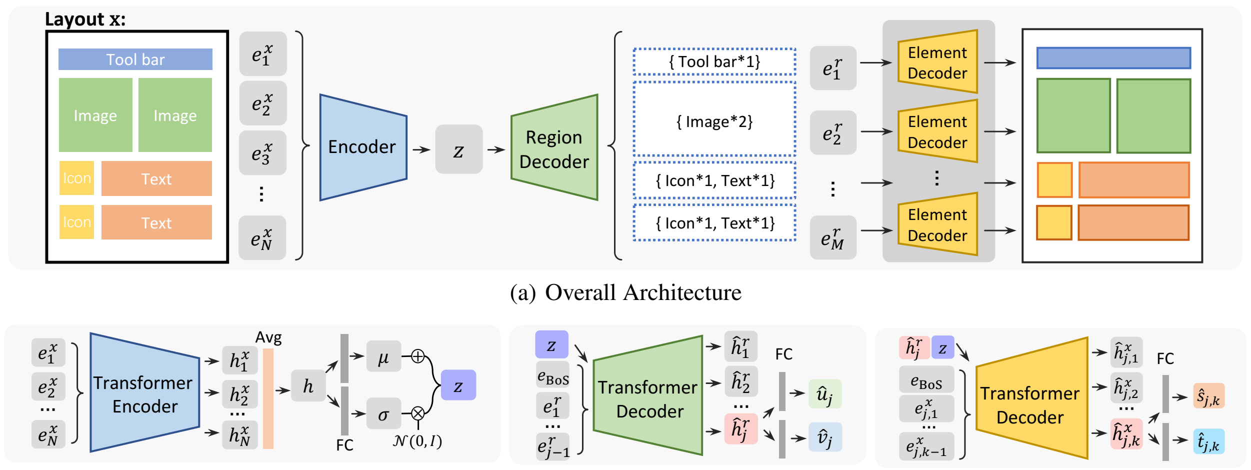
### Architecture Overview

To synthesize a graphic layout $\mathbf{x}$ , we leverage VAE to maximize the data log likelihood $\log p(\mathbf{x})$, which is equivalent to maximizing its Evidence Lower BOund (ELBO),

$$\text{ELBO} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta\left(\mathbf{x}|\mathbf{z}\right)\right] - \text{KL}\left(q_\phi\left(\mathbf{z}|\mathbf{x}\right) \| p_\theta\left(\mathbf{z}\right)\right), \quad (1)$$

where $\mathbf{z}$ stands for the latent code, $p_\theta(\mathbf{z})$ is a fixed Gaussian $\mathcal{N}(0, I)$, and $\theta$ and $\phi$ refer to the learnable parameters.

In this work, we seek to improve the performance of layout generation by incorporating the concept of the regions $\mathbf{r}$ into the generation process. Specifically, we propose de-



(a) Overall Architecture

(b) Encoder (c) Region Decoder (d) Element Decoder

Figure 2: Model architecture.

composing the decoding process $p_\theta(\mathbf{x}|\mathbf{z})$ into two stages,

$$
\begin{aligned}
&\text{ELBO}_{\text{two-stage}} \\
=&\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log \sum_{\mathbf{r}} p_\theta\left(\mathbf{x}, \mathbf{r}|\mathbf{z}\right)] - \text{KL}\left(q_\phi\left(\mathbf{z}|\mathbf{x}\right) \| p_\theta\left(\mathbf{z}\right)\right) \quad (2)\\
\triangleq&\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta\left(\mathbf{x}, \mathbf{r}|\mathbf{z}\right)] - \text{KL}\left(q_\phi\left(\mathbf{z}|\mathbf{x}\right) \| p_\theta\left(\mathbf{z}\right)\right) \quad (3)\\
=&\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta\left(\mathbf{x}|\mathbf{r}, \mathbf{z}\right)] \quad (4)\\
&+ \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta\left(\mathbf{r}|\mathbf{z}\right)] - \text{KL}\left(q_\phi\left(\mathbf{z}|\mathbf{x}\right) \| p_\theta\left(\mathbf{z}\right)\right).
\end{aligned}
$$

Here, Eqn 3 holds under the natural assumption that $p(\mathbf{r}|\mathbf{x}, \mathbf{z})$, i.e., the distribution of region $\mathbf{r}$ given layout $\mathbf{x}$ and latent code $\mathbf{z}$, is a prior distribution that can be obtained from data. To be more specific, when maximizing ELBO$_{\text{two-stage}}$, as the prior $p(\mathbf{r}|\mathbf{x}, \mathbf{z})$ is a constant and $\log \sum_{\mathbf{r}} p(\mathbf{x}, \mathbf{r}|\mathbf{z}) = \log p(\mathbf{x}|\mathbf{z}) = \log p(\mathbf{x}, \mathbf{r}|\mathbf{z}) - \log p(\mathbf{r}|\mathbf{x}, \mathbf{z})$, optimizing $\log \sum_{\mathbf{r}} p(\mathbf{x}, \mathbf{r}|\mathbf{z})$ can be regarded as equivalent to optimizing $\log p(\mathbf{x}, \mathbf{r}|\mathbf{z})$.

In detail, as shown in Figure 2, the first stage $p_\theta\left(\mathbf{r}|\mathbf{z}\right)$, called *region decoder*, predicts regions $\mathbf{r}$ conditioned on the latent code $\mathbf{z}$, while the second stage $p_\theta\left(\mathbf{x}|\mathbf{r}, \mathbf{z}\right)$, called *element decoder*, generates the element placement $\mathbf{x}$ based on the predicted regions $\mathbf{r}$. Besides, $q_\phi\left(\mathbf{z}|\mathbf{x}\right)$ refers to the *encoder* that transforms the graphic layout $\mathbf{x}$ to the latent code $\mathbf{z}$. Their details will be introduced in Section .

## Model Details

Following the successful practice of the state-of-the-art approach (Arroyo, Postels, and Tombari 2021), we leverage Transformer (Vaswani et al. 2017) as our backbone network, design our decoders in an autoregressive manner, and use normal distribution when optimizing KL-divergence.

**Encoder.** First, for the $i$-th element $x_i$ in a layout, the element embedding $e_i^x$ is produced by concatenating its position embedding $e_i^s$ and type embedding $e_i^t$, i.e.,

$$
e_i^x = f_{\text{FC}}\left(\left[e_i^s; e_i^t\right]\right), \; e_i^s = f_{\text{FC}}\left(s_i\right), \; e_i^t = f_{\text{FC}}\left(t_i\right). \quad (5)
$$

Here $f_{\text{FC}}$ is the fully connected layer and $t_i$ is the one-hot encoding for the element type. Regarding $s_i$, following recent studies (Gupta et al. 2020; Arroyo, Postels, and Tombari 2021), we discretize each coordinate and concatenate one-hot encodings of the four discretized coordinates. Besides, $[\cdot;\cdot]$ stands for the concatenation of two vectors.

Then, the layout embedding $h$ is produced by leveraging Transformer to learn relationships between elements, i.e.,

$$
h = \frac{1}{N}\sum_i h_i^x, \; \{h_1^x, \dots, h_N^x\} = f_{\text{TF-ENC}}\left(e_1^x, \dots, e_N^x\right). \quad (6)
$$

Here $f_{\text{TF-ENC}}$ stands for Transformer encoder.

Lastly, we learn the parameters $\mu$ and $\sigma$ of a Gaussian distribution based on the layout embedding $h$, and then leverage reparametrization trick to get the latent code $\mathbf{z}$ of VAE,

$$
\mathbf{z} = \mu + \sigma \cdot \epsilon \text{ and } \epsilon \sim \mathcal{N}(0, I), \;\; \mu = f_{\text{FC}}\left(h\right), \sigma = f_{\text{FC}}\left(h\right). \quad (7)
$$

**Region Decoder.** In the first stage, we generate the representation for each region. Taking the $j$-th region as an example, the information about where the region is $\hat{u}_j$ and the information about which elements are in this region $\hat{v}_j$ is predicted by conditioning on the latent code $\mathbf{z}$ and the embeddings of previous regions $e_1^r, \dots, e_{j-1}^r$, i.e.,

$$
\begin{aligned}
&\hat{u}_j = f_{\text{FC}}\left(\hat{h}_j^r\right), \; \hat{v}_j = f_{\text{FC}}\left(\hat{h}_j^r\right), \quad (8)\\
&\{\hat{h}_1^r, \dots \hat{h}_j^r\} = f_{\text{TF-DEC}}\left(\mathbf{z}, \{e_{\text{BoS}}, e_1^r, \dots, e_{j-1}^r\}\right),
\end{aligned}
$$

where $f_{\text{TF-DEC}}$ stands for Transformer decoder and $e_{\text{BoS}}$ is the embedding of a special token indicating the beginning of the sequence. Besides, $e_c^r$ $(1 \le c \le j-1)$ is obtained following a similar process to Eqn 5.

**Element Decoder.** In the second stage, we generate the placement detail for each element. This process is repeated for every region predicted in the first stage. We take the $k$-th element in the $j$-th region as an example. Specifically, its position $\hat{s}_{j,k}$ and the element type $\hat{t}_{j,k}$ are generated by considering the latent code $\mathbf{z}$, the region representation $\hat{h}_j^r$ and the embeddings of previous elements in this region $e_{j,1}^x, \dots, e_{j,k-1}^x$, i.e.,

$$
\begin{aligned}
&\hat{s}_{j,k} = f_{\text{FC}}\left(\hat{h}_{j,k}^x\right), \; \hat{t}_{j,k} = f_{\text{FC}}\left(\hat{h}_{j,k}^x\right), \quad (9)\\
&\{\hat{h}_{j,1}^x, \dots, \hat{h}_{j,k}^x\} = f_{\text{TF-DEC}}\left(\left[\hat{h}_j^r; \mathbf{z}\right], \{e_{\text{BoS}}, e_{j,1}^x, \dots, e_{j,k-1}^x\}\right),
\end{aligned}
$$

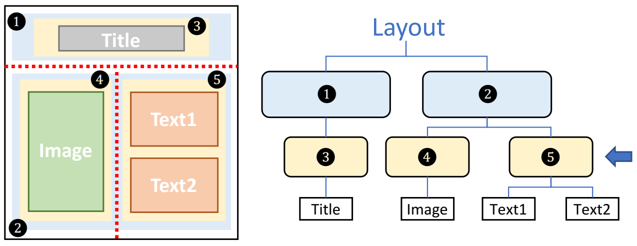


Figure 3: An illustration for extracting a segmentation tree from a graphic layout. The penultimate layer of the segmentation tree is taken as the pseudo region segmentation.

where $e^x_{j,c}$ $(1 \leq c \leq k-1)$ is get by the same process as Eqn 5. Note that $\hat{s}_{j,k}$ is transformed into the relative position to the region position $\hat{u}_j$. This helps model to treat generating elements in a region the same as generating an individual simple layout, and thus facilitates the knowledge sharing between different regions during the training.

In the implementation, input sequences to each Transformer mentioned above are padded to have the same length, and the special tokens representing the beginning of the sequence (BoS) and the end of the sequence (EoS) are added. In both two decoders, we rely on the EoS to decide whether to terminate the generation. Besides, during the training, when producing the embeddings of previous regions $e^r_1, \ldots, e^r_{j-1}$ and previous elements $e^x_{j,1}, \ldots, e^r_{j,k-1}$, we use the ground-truth instead of the prediction.

## Training Objective

According to ELBO in Eqn 4, our training objective combines three parts, i.e., the reconstruction error between generated elements and ground-truth elements, denoted as $\mathcal{L}_{\text{ele}}$, the reconstruction error between generated regions and ground-truth regions, denoted as $\mathcal{L}_{\text{reg}}$, and KL-divergence between the distribution of latent code $\mathbf{z}$ and $\mathcal{N}(0, I)$,

$$\mathcal{L} = \mathcal{L}_{\text{ele}} + \omega_1 \mathcal{L}_{\text{reg}} + \omega_2 \text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) || \mathcal{N}(0, I)), \quad (10)$$

where $\omega_1$ and $\omega_2$ are hyper-parameters to trade-off the weights among three parts. Specifically, $\mathcal{L}_{\text{ele}}$ (or $\mathcal{L}_{\text{reg}}$) is the total loss for all the elements (or regions) in a layout,

$$\mathcal{L}_{\text{ele}} = \sum_i l_{\text{CE}}(\hat{s}_i, s_i) + l_{\text{CE}}(\hat{t}_i, t_i), \quad (11)$$

$$\mathcal{L}_{\text{reg}} = \sum_j l_{\text{CE}}(\hat{u}_j, u_j) + l_{\text{MSE}}(\hat{v}_j, v_j).$$

Here $l_{\text{MSE}}$ is mean square error and $l_{\text{CE}}$ is cross-entropy loss. Note that as all the coordinates are discretized, the cross entropy loss is leveraged for position reconstruction.

## Supervision for Region Representation

As discussed in Section , to learn the region representation, the ground-truth region segmentation, represented by $r_j = (u_j, v_j)$, is an indispensable supervision information for the region decoder. However, graphic layouts hardly provide such information explicitly, since it is not a necessary part of a final graphic design. To solve this problem, we seek to automatically extract a pseudo region segmentation from the graphic layout and regard it as the supervision for learning the region representation. Prior studies (Armstrong 2009; Dayama et al. 2020; Jacobs et al. 2003) indicate that when creating graphic designs, designers often leverage grid lines to decide the overall spatial organization and guide the placement of elements. This motivates us to consider obtaining a pseudo region segmentation with the help of grid lines. In detail, as shown in Figure 3, we scan the layout along either the horizontal direction or the vertical direction, and check whether the layout can be divided into several fragments along a certain direction. We repeat such operation recursively for each fragments until there is no divisible fragments, and finally get a segmentation tree. In our implementation, for simplicity, we take the penultimate layer in the segmentation tree as the pseudo region segmentation and find it can achieve good performance. Note that the region segmentation is not unique and some regions obtained by this way may not be the most plausible solution. Nevertheless, they can also help factor a complex graphic layout into several smaller regions to some extent, and thus will be beneficial to the overall generation performance.

## Experiments

### Experimental Setup

**Datasets.** We evaluate our method on the following publicly available datasets, which are widely used in recent studies about graphic layout generation.

**RICO** (Deka et al. 2017) contains 66K+ mobile app UI layouts with 25 element categories. The annotations are structured by Android view hierarchies. Similar to (Arroyo, Postels, and Tombari 2021), we omit layouts with more than 100 elements due to the memory constraint. In total, we get 54K screenshot data for training and 6K data for validation.

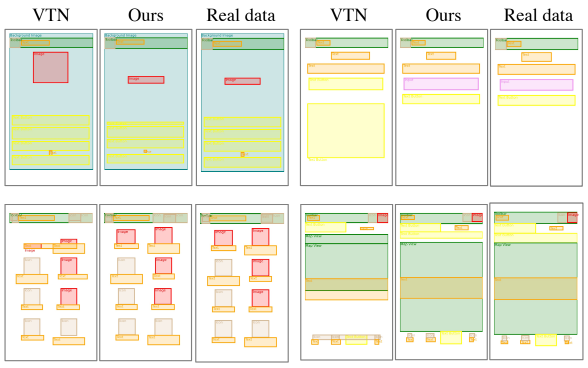
**PubLayNet** (Zhong, Tang, and Yepes 2019) contains 360K+ document layouts with 5 element categories. We also omit layouts with more than 100 elements, and finally get 300K data for training and 33K for validation.

**Compared Methods.** Early studies usually impose restrictions when generating layout, e.g., using heuristic-based labels for element relationships (Patil et al. 2020; Lee et al. 2020) or handling a limited number of elements (Li et al. 2019; Jyothi et al. 2019) (see Section ). The recent work, VTN (Arroyo, Postels, and Tombari 2021), explores generic layout generation and achieves state-of-the-art performance. Thus, we compare against **VTN** and its similar model from **Gupta et al.**. We also compare against **NDN** (Lee et al. 2020) using their proposed metric based on rendered layout images though it only tackles a limited number of elements.

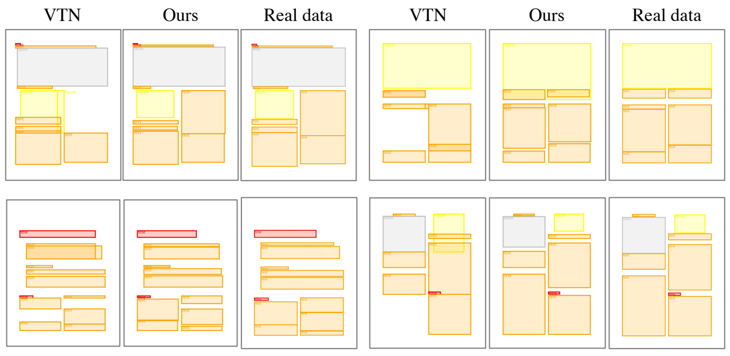
**Implementation Details.** Our approach is implemented by PyTorch. For Transformer blocks, we stack 4 layers with a representation size of 512 and a feed-forward representation size of 1024, and use multi-head attentions with 4 heads. We use Adam optimizer (Kingma and Ba 2014) with initial learning rate $10^{-3}$ reduced by a factor of 0.8. The dropout rate of transformer blocks is set to 0.1. All models are trained for 300 epochs on RICO and 100 epochs on PubLayNet, with batch size of 256 on two V100 GPUs.

| | RICO | | | | PubLayNet | | | |
|---|---|---|---|---|---|---|---|---|
| | Overlap ↓ | Align. ↓ | W class ↓ | W bbox ↓ | Overlap ↓ | Align. ↓ | W class ↓ | W bbox ↓ |
| Gupta et al. | 0.145 | 0.366 | **0.004** | 0.023 | 0.006 | 0.361 | 0.018 | 0.012 |
| VTN | 0.165 | 0.373 | 0.007 | 0.018 | 0.017 | 0.347 | 0.022 | 0.012 |
| Ours | **0.139** | **0.276** | 0.007 | **0.012** | **0.005** | 0.352 | **0.007** | **0.012** |
| Real data | 0.175 | 0.410 | - | - | 0.007 | 0.353 | - | - |

Table 1: Quantitative comparisons of generation. For each method, we generate 1000 layouts for evaluation.



(a) RICO



(b) PubLayNet

Figure 4: Qualitative comparisons of reconstruction performance.

| | | CD ↓ | | | CD ↓ |
|---|---|---|---|---|---|
| RICO | VTN | 0.0573 | PubLayNet | VTN | 0.0304 |
| | Ours | **0.0276** | | Ours | **0.0278** |

Table 2: Quantitative comparisons of reconstruction.

## Quantitative Comparisons

**Evaluation Metrics.** To evaluate reconstruction performance, we leverage Chamfer Distance (**CD**). Regarding generation performance, we resort to a set of metrics, each of which represents one aspect of perceptual quality and diversity, including overlap index (**Overlap**), alignment score (**Align.**), Wasserstein Distance (**W class** and **W bbox**) and Fréchet Inception Distance (**FID**).

**CD.** Referring to the definition of CD on SVG images (Carlier et al. 2020), we define CD between the ground-truth layout $\mathbf{x}$ and the reconstructed layout $\hat{\mathbf{x}}$ as the sum of the minimal distance between elements, i.e., $s_{\text{CD}}(\hat{\mathbf{x}}, \mathbf{x}) = \frac{1}{N}\sum_{i=1}\min_j \|s_i - \hat{s}_j\|^2$.

**Overlap.** Usually, the elements on the layout should not overlap excessively. Following (Li et al. 2020), we calculate the intersection area of any two elements in the layout, i.e., $s_{\text{Overlap}}(\hat{\mathbf{x}}) = \frac{1}{2N}\sum_{i=1}^{N}\sum_{j\neq i}\frac{c_i \cap c_j}{c_i}$, where $c_i$ denotes the $i$-th element's area and $c_i \cap c_j$ denotes the overlapping area between element $i$ and $j$.

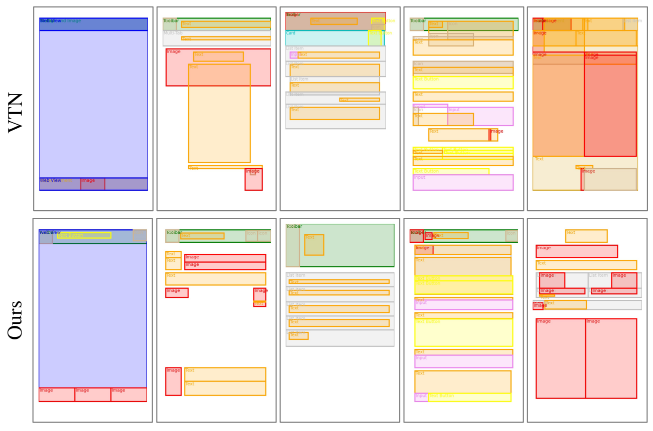
**Align.** Good alignment can create a sense of tidiness. Following (Lee et al. 2020), we consider 3 common alignment types in a graphic layout (including left, center and right), i.e., $s_{\text{Align.}}(\hat{\mathbf{x}}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq i}\min d_T(o_i^T, o_j^T)$, where $T$ denotes the alignment type, $o_i^T$ denotes the corresponding coordinate of the element $x_i$ under the alignment type $T$, and $d_T$ is the distance between the two coordinates.

**W class** and **W bbox.** The distribution for generated layouts should be close to that for real layouts. Following (Arroyo, Postels, and Tombari 2021), we approximate Wasserstein distance between generated and real layouts for two marginal distributions about element types and positions.

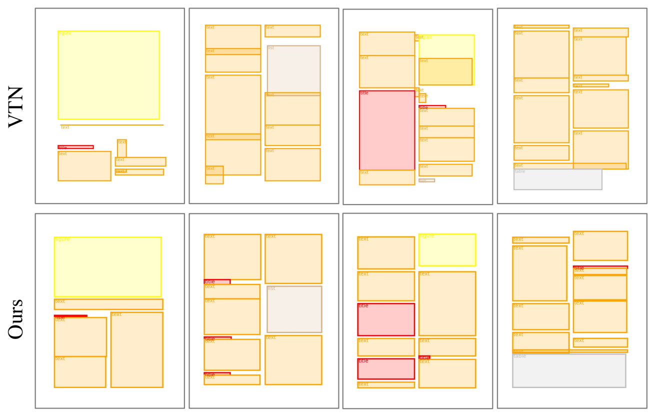
**FID** also describes distribution difference between real and generated layouts. Unlike above metrics that are based on element types and position values, FID depends on rendered layout images. Following (Lee et al. 2020), we get layout embedding for FID calculation by training a CNN classifier to discriminate ground-truth layouts from randomly-perturbed layouts. The classifier is trained to achieve the accuracy of 95% and 96% on RICO and PubLayNet.

**Results and Analysis.** Table 2 shows quantitative comparisons of reconstruction performance, where lower CD indicates better performance on the training set. On both datasets, our method achieves the best performance, demonstrating the superiority of the proposed two-stage decoder.

Table 1 shows quantitative comparisons of generation performance, where lower values indicate better performance. Our approach achieves the best performance on almost every metric. To further investigate the advantage of our approach, we classify layouts into different groups by the number of elements and conduct evaluations on these groups. As shown



(a) RICO



(b) PubLayNet

Figure 5: Qualitative comparisons of generation performance.

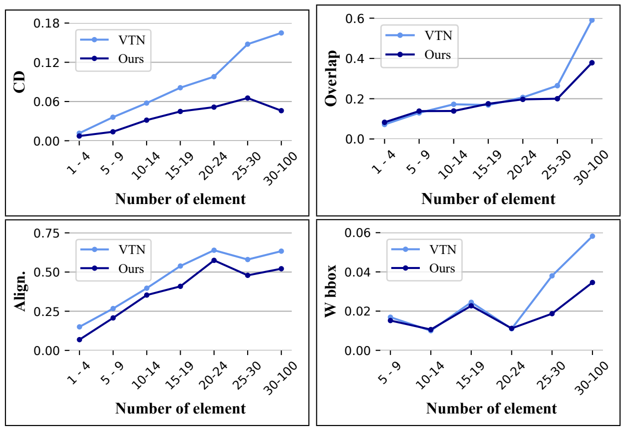


Figure 6: Quantitative comparisons when the layouts are grouped by the number of elements (on RICO).

| | FID↓ | |
|---|---|---|
| | RICO | PubLayNet |
| NDN | $143.51 \pm 22.36$ | - |
| VTN | $104.67 \pm 29.51$ | $66.41 \pm 20.55$ |
| Ours | $\mathbf{61.87 \pm 5.77}$ | $\mathbf{32.42 \pm 19.76}$ |

Table 3: Quantitative comparisons of generation by rendered layout images. Each experiment is repeated for 5 times.

in Figure 6, on each metric, the performance gap between our approach and VTN increases with the number of elements. This indicates that the advantage of our approach lies in handling complex layout with more elements. It is also consistent with the intuition that by introducing regions, our approach can decouple a complex layout into several simple layouts and thus reduce generation difficulty.

Besides, Table 3[1] compares generation performance from another angle, i.e., rendered layout images instead of element types and position values. Our approach outperforms other methods, further demonstrating that our approach can generate more realistic and diverse layouts.

## Qualitative Comparisons[2]

Figure 4 shows qualitative comparisons of reconstruction performance on training set. For ease of comparison, each subfigure shows reconstructed layouts by VTN (left) and our approach (middle) for the same real layout (right). Although both approaches perform well on the reconstruction, our approach still outperform VTN in terms of reconstructing positions more precisely and predicting element types correctly.

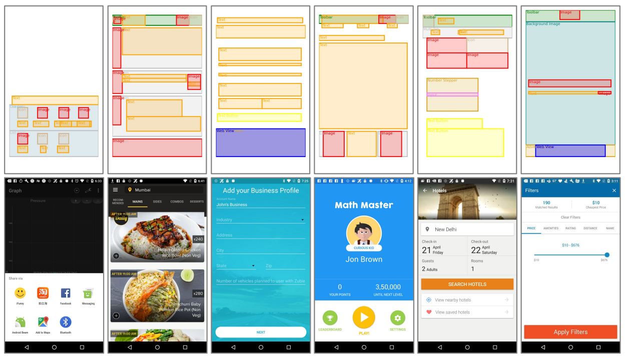
Figure 5 shows qualitative comparisons of generation performance. On RICO, when there are lots of elements on a layout, VTN struggles to handle overlaps between elements, and tends to arrange elements in a disorderly manner. By contrast, our approach seldom generates layouts with unreasonable overlaps and can arrange elements in more diverse and complicated way. Similarly, on PubLayNet, our approach better handles the right alignment and the overlap issue compared to VTN.

Figure 7 shows generated layouts and their closest real graphic designs in the training set, which are obtained using DocSim match mentioned in (Patil et al. 2020). While the very similar real data can be found from the training set, our generated layouts still have some differences with the real one. This indicates that our generated layouts are reasonable, realistic and diverse at the same time.

Figure 8 shows generated layouts and corresponding regions predicted by the region decoder. It can be clearly observed that the predicted regions depict a rough appearance for the layouts. Besides, compared to the entire layout, each region contains a much smaller number of elements and the element arrangement within it also becomes simpler.

[1]On PubLayNet, FID of NDN is not displayed because we fail to reproduce it by ourselves and the authors do not release codes.

[2]Please refer to Supplementary for more qualitative results.


(a) RICO

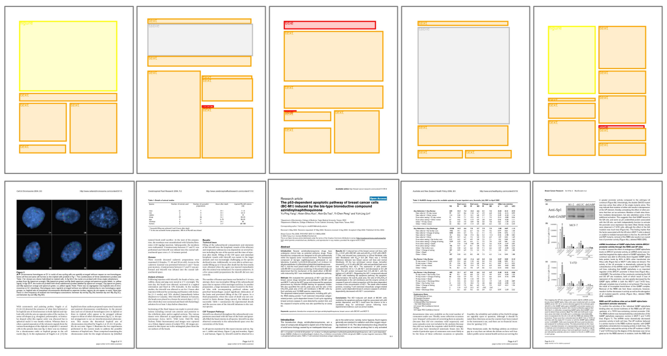
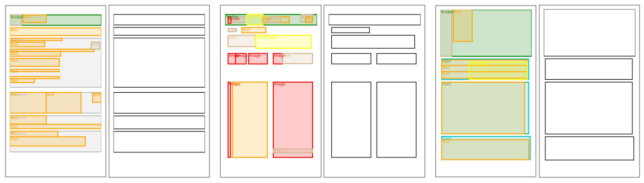

(b) PubLayNet

Figure 7: Generated layouts and their closest real graphic designs in the training set.


(a) RICO

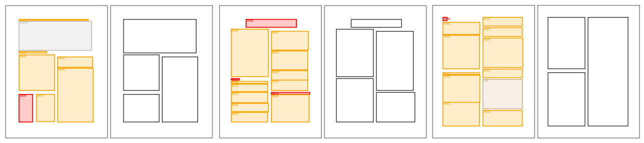

(b) PubLayNet

Figure 8: Generated layouts and their predicted regions.

## Model Variants

Table 4 shows performance for different model variants. In the region decoder, we transform the absolute element positions into the relative positions w.r.t. the region, in order to help the model treat generating elements in a region the same as generating elements in an individual simple layout. In the first model variant, we examine whether directly using absolute positions will result in better performance, denoted as **AbsPos** in Table 4. Experimental results show that this variant does not perform as well as our original model (denoted as **Ours** in Table 4) on most metrics.

Moreover, we study the possible impact of different kinds of region segmentation. Specifically, we leverage the Android view hierarchy provided by RICO itself as the region segmentation during the training. We denote this model variant as **AndView** in Table 4. It is observed that this variant still outperforms other baselines shown in Table 1 on most metrics, indicating the our approach is relatively robust to the choice of the region segmentation. As for that this variant does not perform as well as our original model, we hypothesize the reason as that Android view hierarchy is mainly based on relationships of widget functions rather than relationships of element placement.

| Model | CD ↓ | Overlap↓ | Align.↓ | W class↓ | W bbox↓ |
|---|---|---|---|---|---|
| Ours | **0.028** | 0.139 | **0.276** | **0.007** | **0.012** |
| AbsPos | 0.041 | **0.131** | 0.470 | 0.011 | 0.016 |
| AndView | 0.049 | 0.142 | 0.320 | 0.009 | 0.017 |
| TwoEnc | 0.265 | 0.158 | 0.522 | 0.012 | 0.027 |

Table 4: Performance for model variants (on RICO).

Besides, we consider a model variant that leverages an extra two-stage encoder to incorporate the concept of regions. Specifically, the two-stage encoder mirrors the two-stage decoder, which first encodes elements into region embedding and then encodes all the region embeddings into a latent vector $\mathbf{z}$. We denote it as **TwoEnc** in Table 4. We find that this variant does not perform better than our original model. We think it is comprehensible and reasonable. On the one hand, by learning a two-stage decoder, the model are already equipped to capture the concept of regions. On the other hand, an extra two-stage encoder introduces more parameters and thus increases the training difficulty.

## Conclusion

In this work, we propose a coarse-to-fine approach for graphic layout generation. Specifically, we decompose the generation process into two stages, where the first stage predicts the region representations that depict rough appearance of the layout, and the second stage generates the detailed element placement conditioned on the predicted region representation. Quantitative and qualitative experiments demonstrate that our approach achieve superior performance than the previous methods which generates the entire layout in one go. In the future, we plan to evaluate the performance of our approach when user intents are incorporated, e.g., the length of texts and the fixed aspect ratio of images. Besides, we will continue to explore more effective region segmentations, e.g. the one consistent with the layout semantics, to better help graphic layout generation.

## References

Aksan, E.; Deselaers, T.; Tagliasacchi, A.; and Hilliges, O. 2020. CoSE: Compositional Stroke Embeddings. *arXiv preprint arXiv:2006.09930*.

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*, 214–223. PMLR.

Armstrong, H. 2009. *Graphic design theory: Readings from the field*. Chronicle Books.

Arroyo, D. M.; Postels, J.; and Tombari, F. 2021. Variational Transformer Networks for Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13642–13652.

Carlier, A.; Danelljan, M.; Alahi, A.; and Timofte, R. 2020. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation. *arXiv preprint arXiv:2007.11301*.

Dayama, N. R.; Todi, K.; Saarelainen, T.; and Oulasvirta, A. 2020. Grids: Interactive layout design with integer programming. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–13.

Deka, B.; Huang, Z.; Franzen, C.; Hibschman, J.; Afergan, D.; Li, Y.; Nichols, J.; and Kumar, R. 2017. 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*, 845–854.

Goller, C.; and Kuchler, A. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, 347–352. IEEE.

Gupta, K.; Achille, A.; Lazarow, J.; Davis, L.; Mahadevan, V.; and Shrivastava, A. 2020. Layout Generation and Completion with Self-attention. *arXiv preprint arXiv:2006.14615*.

Ha, D.; and Eck, D. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*.

Jacobs, C.; Li, W.; Schrier, E.; Bargeron, D.; and Salesin, D. 2003. Adaptive grid-based document layout. *ACM transactions on graphics (TOG)*, 22(3): 838–847.

Jyothi, A. A.; Durand, T.; He, J.; Sigal, L.; and Mori, G. 2019. Layoutvae: Stochastic scene layout generation from a label set. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 9895–9904.

Kikuchi, K.; Simo-Serra, E.; Otani, M.; and Yamaguchi, K. 2021. Constrained Graphic Layout Generation via Latent Optimization. *arXiv preprint arXiv:2108.00871*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Lee, H.-Y.; Jiang, L.; Essa, I.; Le, P. B.; Gong, H.; Yang, M.-H.; and Yang, W. 2020. 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*, 491–506. Springer.

Li, J.; Yang, J.; Hertzmann, A.; Zhang, J.; and Xu, T. 2019. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767*.

Li, J.; Yang, J.; Zhang, J.; Liu, C.; Wang, C.; and Xu, T. 2020. Attribute-conditioned layout gan for automatic graphic design. *IEEE Transactions on Visualization and Computer Graphics*.

Patil, A. G.; Ben-Eliezer, O.; Perel, O.; and Averbuch-Elor, H. 2020. Read: Recursive autoencoders for document layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 544–545.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Ribeiro, L. S. F.; Bui, T.; Collomosse, J.; and Ponti, M. 2020. Sketchformer: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14153–14162.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Yamaguchi, K. 2021. CanvasVAE: Learning to Generate Vector Graphic Documents. *arXiv preprint arXiv:2108.01249*.

Zheng, X.; Qiao, X.; Cao, Y.; and Lau, R. W. 2019. Content-aware generative modeling of graphic design layouts. *ACM Transactions on Graphics (TOG)*, 38(4): 1–15.

Zhong, X.; Tang, J.; and Yepes, A. J. 2019. Publaynet: largest dataset ever for document layout analysis. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, 1015–1022. IEEE.

Zhu, J.-Y.; Park, T.; Isola, P.; and Efros, A. A. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, 2223–2232.