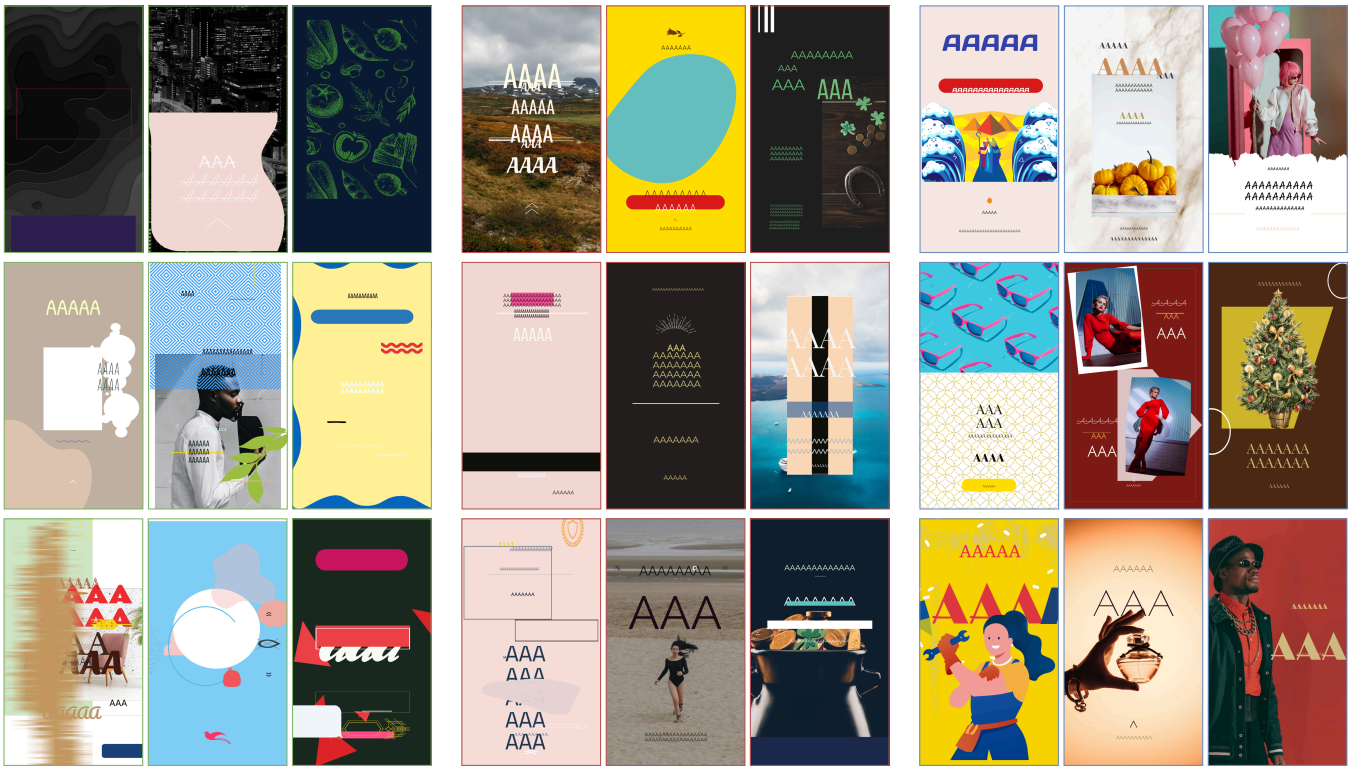
Order Matters: Learning Element Ordering for Graphic Design Generation

BO YANG, ShanghaiTech University, China
YING CAO*, ShanghaiTech University, China



Random Raster Neural (Ours)

Fig. 1. Comparison of graphic design sampled from our autoregressive generator trained with different approaches for ordering elements in training design sequences, including random, raster and our neural order. For each order, we generate 3000 samples and sort them in descending order of sample quality measured by negative log likelihood. Then, we show 3 random examples from the top 10%, 3 randomly selected examples among all the samples, and 3 random examples from the bottom 10%. Our neural order shows significantly improved visual quality compared to random order and raster order, resulting in high-quality designs with favorable properties, such as rich visual contents, well-aligned elements, avoiding occlusion of important image regions, great layout variations and harmonious colors.

*Corresponding author.

Authors' addresses: Bo Yang, yangbo2022@shanghaitech.edu.cn, ShanghaiTech University, Shanghai, China; Ying Cao, yingcao59@gmail.com, ShanghaiTech University, Shanghai, China.

ACM 0730-0301/2025/8-ART
https://doi.org/10.1145/3730858

The past few years have witnessed an emergent interest in building generative models for the graphic design domain. For adoption of powerful deep generative models with Transformer-based neural backbones, prior approaches formulate designs as ordered sequences of elements, and simply order the elements in a random or raster manner. We argue that such naive ordering methods are sub-optimal and there is room for improving sample quality through a better choice of order between graphic design elements. In this paper, we seek to explore the space of orderings to find the ordering strategy that optimizes the performance of graphic design generation models. For this, we propose a model, namely **G**enerative **O**rder **L**earner (GOL), which trains an autoregressive generator on design sequences, jointly with an ordering network that sort design elements to maximize the generation quality. With unsupervised training on vector graphic design data, our model is capable of learning a content-adaptive ordering approach, called neural order. Our experiments show that the generator trained with our

neural order converges faster, achieving remarkably improved generation quality compared with using alternative ordering baselines. We conduct comprehensive analysis of our learned order to have a deeper understanding of its ordering behaviors. In addition, our learned order can generalize well to diffusion-based generative models and help design generators scale up excellently.

CCS Concepts: • **Computing methodologies** → **Computer graphics**.

Additional Key Words and Phrases: Graphic Design, Design Generation, Generative Model

ACM Reference Format:
Bo Yang and Ying Cao. 2025. Order Matters: Learning Element Ordering for Graphic Design Generation. *ACM Trans. Graph.* 44, 4 (August 2025), 16 pages. https://doi.org/10.1145/3730858

1 INTRODUCTION

Graphic designs, such as advertisements, posters and documents, are a common form of creative mediums that effectively delivers the intended message to audiences. To create good graphic designs, one usually needs to navigate through a large solution space spanned by various fundamental dimensions (e.g., color, text, image and layout), while considering some important aesthetic and functional objectives. As a result, graphic design is often an iterative process, requiring extensive professional knowledge, rich hands-on experience and some creativity. The importance and challenges of graphic design have motivated an emerging stream of research efforts in developing computational models and systems to facilitate the design process, with many encouraging results.

Recent deep generative models (e.g., autoregressive models and diffusion models) have demonstrated revolutionary capabilities and have seen great success in image and language generation. This stimulates tremendous interest in generative modeling of some core components of graphic design (e.g., layout) [Gupta et al. 2021; Inoue et al. 2023a; Jyothi et al. 2019; Li et al. 2019] and complete designs [Inoue et al. 2024; Jia et al. 2023; Yamaguchi 2021]. Unlike images that are pixels organized on 2D grids, graphic designs are essentially composed of basic multi-modal elements, such as text and images, and are characterized by complex global relationships between the elements—the choice and configuration of one elements hinge crucially on those of other elements in a design and vice versa. This makes the attention-based Transformer architecture the de facto choice of neural backbone in most of the prior generative models for graphic design, due to its inductive bias towards learning the relationships between elements. To adapt powerful classes of generative models with Transformer backbones into the graphic design domain, graphic designs are formulated as *ordered* sequences of tokens [Arroyo et al. 2021; Gupta et al. 2021; Inoue et al. 2023a; Jiang et al. 2023a; Zhang et al. 2023]. Unfortunately, unlike sentences in which the positions of words are inherently available, the order of graphic design elements is *unknown*. To mitigate this issue, existing methods sort elements per design randomly or using simple handcrafted rules (e.g., raster order) [Gupta et al. 2021; Inoue et al. 2023a; Zhang et al. 2023]. Despite promising results, it remains unclear whether such simple ordering approaches hinder the performance of the trained generative models. Thus, it is natural to ask: *is there any better order that can optimize the capabilities of graphic design generation models?*

In this paper, we aim to demystify the significance of element ordering in generative models for graphic design. We show that the random and raster order used by existing works is sub-optimal for the performance of the generation models and seek to find a *better* order that can contribute to a noticeable improvement of generation quality. We refrain from utilizing any hand-engineered rules, and instead learn an element ordering strategy end-to-end from unlabeled data. For this end, we propose a model, dubbed as *Generative Order Learner*, or GOL for short, which encompasses two core components: an ordering network and a design generator. The ordering network learns to sort a set of elements in a design based on their semantic, spatial and stylistic properties, while the design generator is trained to autoregressively generate design tokens in the order predicted by the ordering network. The two components are trained *simultaneously*, in such a way that the ordering network is guided to predict the order that can maximize the sample quality of the generator. By training the GOL on a vector graphic design dataset without any human annotations, we end up with a learned ordering approach, which is referred to as *neural order*.

We first compare our neural order against several baseline ordering approaches, including random, raster, saliency orders, and show that the neural order can speed up training and lead to a significant boost in generation quality compared with the other orders across different graphic design datasets. We also show that our neural order has great generalizability to a diffusion-based generator. Then, we probe the neural order to reveal some of its intriguing behaviors, such as putting textual elements before non-text elements and content-aware ordering. Finally, we demonstrate that the generator trained with our neural order exhibits promising scaling behaviors: remarkable quality gains are observed by scaling up the model. The code of this work is at https://borisyang326.github.io/ordermatters.

In summary, we make the following contributions:

- We make the first attempt to investigate the optimal element ordering approach for training graphic design generation models, as opposed to using random or heuristic-based ones as in existing works.
- We present a model, which is capable of learning a content-aware ordering approach that effectively optimizes the generative capabilities of autoregressive and diffusion-based design generators.
- We conduct extensive analysis of our learned order, providing insights into how it sorts elements based on their low-level attributes and high-level semantics.
- We study the scalability of the design generator trained with different ordering approaches, showing that our learned order outperforms the other orders in helping design generative models benefit from scaling up model size.

2 RELATED WORK

2.1 Graphic Layout Generation

Graphic layout generation determines the positions and sizes of elements in a design, which is a fundamental task in graphic design. Layout generation has been extensively studied in recent years, and the community has explored different kinds of generative modeling frameworks for layout generation under both unconditional and

conditional scenarios. In the area of layout generation, a common practice is to represent a layout as a set of elements, each described by its semantic class and bounding box coordinates.

Early works exploited *Generative Adversarial Networks* (GAN) to learn layout distributions unconditionally [Li et al. 2019] or conditioned on some user inputs [Kikuchi et al. 2021; Zheng et al. 2019; Zhou et al. 2022]. For instance, LayoutGAN [Li et al. 2019] is a GAN-based model for unconditional layout generation, where the generator is attention-based to capture contextual relationships between elements and both attention-based and convolutional discriminators are explored. Zheng et al. [Zheng et al. 2019] proposes a conditional GAN based on convolutional networks to model the impact of input visual and textual contents upon layouts. CLG-LO [Kikuchi et al. 2021] trains a Transformer-based layout GAN and formulates constrained layout generation as a constrained optimization over the latent space of the pre-trained GAN.

Variational Autoencoders (VAE) has also been leveraged to approach the layout generation problem [Arroyo et al. 2021; Cao et al. 2022; Jyothi et al. 2019; Lee et al. 2020; Patil et al. 2020]. For example, LayoutVAE [Jyothi et al. 2019] generates a layout from a given multiset of element class labels by training two separate VAEs to first predict element counts and then element bounding boxes autoregressively. Arroyo et al. [Arroyo et al. 2021] adopt a VAE-based framework for layout generation with attention layers as the building blocks and find that an autoregressive decoder works better than a non-regressive one. Some VAE-based methods also consider conditional layout generation, e.g., from user-specified relationships between elements [Lee et al. 2020] and given background images[Cao et al. 2022].

Motivated by the great success of *Autoregressive Models* (ARM) in text generation and pre-training in NLP, more recent works have experimented with using them for layout generation and achieved impressive performance [Gupta et al. 2021; Horita et al. 2024; Jiang et al. 2023a]. LayoutTransformer [Gupta et al. 2021] flattens the attributes of all elements in a layout into an ordered 1D sequence using raster order and generates them with an autoregressive Transformer decoder. This work also empirically shows that the choice of element ordering approaches can critically affect generation performance, but did not suggest how to find a better order. In view of the effectiveness of LayoutTransformer, later literatures build upon its design of the sequence format and network architecture to flexibly handle various types of user constraints [Jiang et al. 2023a] and boost the performance of image-based layout generation via retrieval augmentation [Horita et al. 2024]. BLT [Kong et al. 2022] follows the sequence format of LayoutTransformer, but exploits a bi-directional Transformer to predict all element attributes in parallel. There is also a recent explosion of attempts to represent layouts as sequences in HTML or SVG format and leverage the vast pre-trained knowledge of large language models and large multimodal models for layout generation [Feng et al. 2023; Lin et al. 2023a; Seol et al. 2024; Tang et al. 2023]; these works use randomized order of elements in a layout sequence.

Diffusion models (DM) have recently emerged as a powerful class of generation models, demonstrating strong capabilities in high-fidelity and diverse image generation [Dhariwal and Nichol 2021; Ho et al. 2020; Podell et al. 2023; Rombach et al. 2022]. This inspires several recent efforts to learn distributions of complex and diverse layout data using diffusion models [Cheng et al. 2023; Inoue et al. 2023a; Zhang et al. 2023]. By following the convention of representing layouts as sequences of discrete tokens, some works [Inoue et al. 2023a; Zhang et al. 2023] specially tailor discrete state-space diffusion models [Austin et al. 2021] for layout generation, and opt to leverage the Transformer encoder as the neural backbones. PLay [Cheng et al. 2023] takes a two-stage approach that projects layouts from the discrete representation space into a continuous latent space and learns a continuous Transformer-based diffusion model over the latents. Although the architecture of the Transformer encoder is permutation invariant, these methods find it beneficial to *add positional embeddings* to the Transformer inputs, which is further backed by our experiments. This effectively indicates the importance of the order of elements in a layout representation.

Graphic layout generation can be viewed as a specialized form of graphic design generation that synthesizes a richer set of element attributes (e.g., layer depth, image and font family, etc.) beyond merely class labels and bounding box coordinates. Therefore, most of the aforementioned works can be extended to generate full graphic designs by considering more attributes for each element. Inspired by this, we build our design generator with an autoregressive Transformer decoder, which has been shown to be effective for layout generation [Gupta et al. 2021; Horita et al. 2024; Jiang et al. 2023a]. Notably, although the leading methods cast layout generation as a sequence modeling problem, they simply order elements in a training sequence using random or raster order, which is not optimal for full design generation (we will show this in the experiments). To address this, this work attempts to learn a better ordering approach for optimizing generation performance, which has not been investigated yet.

2.2 Graphic Design Generation

Graphic design generation aims to generate the attributes of elements that are necessary to render complete graphic designs, such as position, size, color, image and font family, etc. Compared to layout generation, the problem of synthesizing realistic and diverse full designs is much more challenging, and is still less explored in the community.

Recently, several promising attempts have been made towards this end. CanvasVAE [Yamaguchi 2021] describes a vector graphic design as a canvas followed by a sequence of elements, and trains a VAE on such sequential representations. It sorts elements in a sequence based on layer depth, which is not directly applicable to datasets without such information. Our model learns an ordering approach that works well for a variety of datasets without need for such layering depth information. There is also a body of works on automating generation for advertising posters [Guo et al. 2021; Lin et al. 2023b], magazines [Yang et al. 2016] and banners [Vaddamanu et al. 2022]. These methods require a background image as input, and more importantly, are domain-specific, rendering it difficult to generalize them across different domains. In contrast, our model can generate design samples from scratch, and generalizes well to different domains since any designs can be formatted as sequences uniformly. Recently, there is an emerging line of research

on developing text-to-design systems by fine-tuning and combining large language models (LLM), large multimodal models (LMM), and image diffusion models [Inoue et al. 2024; Jia et al. 2023]. Due to the high cost of building and training such complex cascaded systems, it is non-trivial to adapt them to input conditions beyond text. Furthermore, these methods merge all visual elements, such as images and SVGs, into one or two image layers. While this allows them to generate highly aesthetic and diverse images with the help of the diffusion models, it inevitably limits the editability of the generated designs, where modifying small, elementary visual components (e.g., icons and embellishments) is not allowed. In contrast, we aim to develop an unconditional foundation model for graphic design generation, which can be effortlessly extended to handle conditional inputs of different forms and modalities. In addition, we keep the training vector graphic designs in the original format without element merging, which enables our model to produce highly editable outputs.

2.3 Neural Sort

Sorting of a given list of elements is one of the most classical problems in computer science [Cormen et al. 2009]. Conventional sorting algorithms are non-differentiable, preventing the use of sorting operations in neural networks that are optimized with gradient-based methods. This problem has been addressed with the introduction and development of differentiable sorting algorithms, i.e., neural sort [Bhunia et al. 2020; Grover et al. 2019]. Neural sort enables neural networks with sorting operations to be trained end-to-end using gradient descent. Like classical sorting methods, neural sort operates on a list of values and outputs a sorting of the values represented by a permutation; gradients can be back-propagated from the output permutation to the input values during optimization.

In this work, we incorporate a neural sorting layer in our network to learn an element ordering approach from unlabeled graphic design data end-to-end. Our work is related to an existing method [Bhunia et al. 2020] that makes use of neural sort to learn a strategy of ordering sketch strokes to maximize the chance of the sketch's early recognition during drawing. Their model is trained under the guidance of a discriminative model: a pre-trained, fixed sketch recognization network. The main distinctions between our method and [Bhunia et al. 2020] are: 1) we learn an ordering model for graphic design elements to maximize the performance of a design generator; 2) our method faces a more challenging learning problem since the design generator, which guides the learning of our ordering model, needs to updated jointly with the ordering model, instead of being fixed, during training.

3 METHOD

Given a training dataset of vector graphic designs (or rasterized graphic designs with element-level annotations), we aim to order the elements within each design so that the performance of a design generator, trained on the reordered design data, can be maximized. Instead of enforcing any heuristic rules as in existing works, we seek to learn an ordering approach from graphic design data, without any human annotations on element ordering. To this end, we propose a Generative Order Learner (GOL), which is illustrated in Figure 2. Our GOL consists of a design generator along with an ordering network inside it. With designs being formatted as sequences of discrete tokens, the design generator learns a generative model of design sequences based on a Transformer backbone that captures contextual relationships between design tokens. The ordering network learns to permute the elements in a design according to their geometric and visual properties, for training the design generator. The weights of both design generator and ordering network are updated simultaneously in training for mutual improvements—a stronger design generator is able to provide more useful supervisory signal for the ordering network while a better ordering network can contribute the performance improvement of the design generator. It should be noted that, with our GOL, the ordering network is learned *without* need for human-annotated ordering targets; instead, its learning signal totally comes from the design generator, which is optimized by minimizing the negative log-likelihood of unlabeled design sequence data. Below, we first introduce the unified sequence format of graphic designs and then describe the design generator and ordering network in detail.

3.1 Design Representation

A graphic design is composed of a canvas and a sequence of N elements. We divide graphic design elements into two broad categories: *text* and *non-text*, and introduce two special tokens `[TXT]` and `[NTXT]` to indicate them, respectively. Thus, a graphic design $\mathbf{X}$ can be reformatted as: $\mathbf{X} = \{\mathbf{v}, \{\mathbf{e}^{\text{txt}}\}, \{\mathbf{e}^{\text{ntxt}}\}\}$. $\mathbf{v} = (w, h, r)$ denotes the canvas with width w, height h and color r. A text element $\mathbf{e}^{\text{txt}}$ encompasses a set of attributes: $\mathbf{e}^{\text{txt}} = (c, x, y, w, h, z, f_f, f_c, f_s)$, where c denote the element's subcategory (e.g., headline or subheadline in some posters), and (x, y, w, h) denote its bounding box with top-left position (x, y) , width w and height h. z is the element's layer depth that determines which element is on top of which element when rendering a generated design. Such depth information is available in vector graphic design datasets such as Crello [Yamaguchi 2021]. (f_f, f_c, f_s) denotes common typographic properties including font family, font color and font size. Similarly, a non-text element $\mathbf{e}^{\text{ntxt}}$ has subcategory c (e.g., SVG or photo in some posters), bounding box (x, y, w, h), layer depth z, but without the typographic properties. We additionally add an image attribute, denoted as u, to $\mathbf{e}^{\text{ntxt}}$, obtaining $\mathbf{e}^{\text{ntxt}} = (c, x, y, w, h, z, u)$. Rather than using image pixels directly, we represent u using a lower-dimensional feature vector constructed by concatenating an image-level embedding extracted with a pre-trained visual encoder [Fu et al. 2023] and a geometric feature vector formed by duplicating the size and aspect ratio of the image.

Following existing works on graphic layout generation [Gupta et al. 2021; Horita et al. 2024; Inoue et al. 2023b; Jiang et al. 2023a], we format a design as a sequence of discrete tokens as illustrated in Figure 3. More specifically, we quantize each of the continuous element attributes (i.e., x, y, w, h, u) into discrete bins. We also perform quantization on f_c and f_s to account for imbalanced distributions of font colors and sizes in training datasets while reducing the computational overhead of training. All the quantization is based on the k-means clustering algorithm [MacQueen 1967]. After such discretization, each element in the design can be formulated a subsequence of

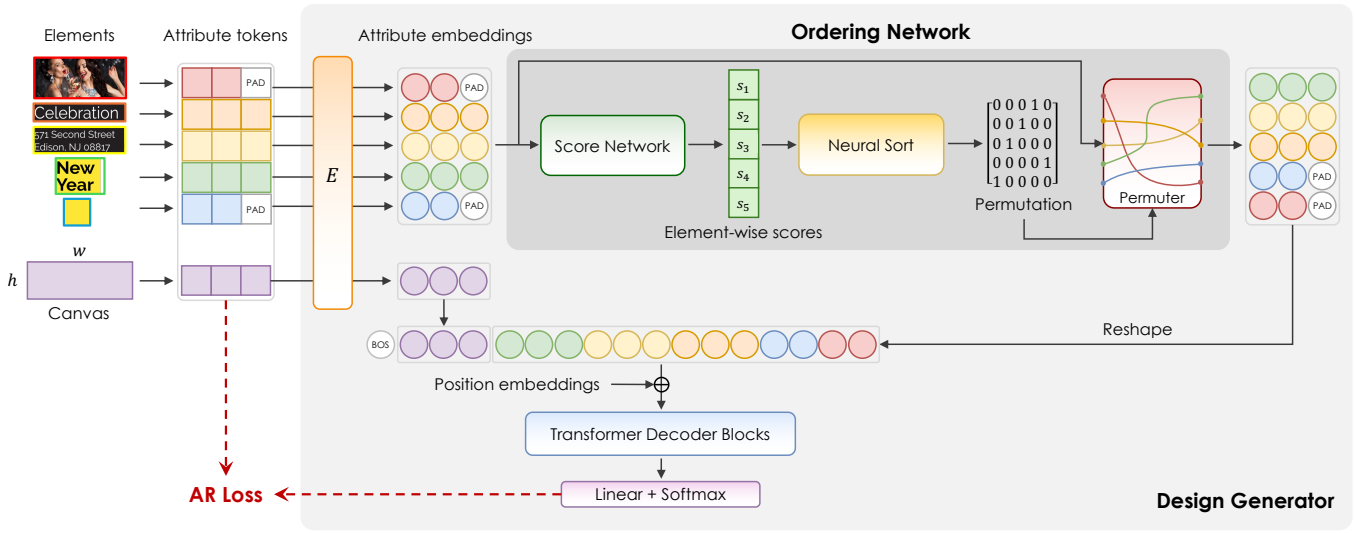


Fig. 2. Overview of our Generative Order Learner (GOL) framework with an autoregressive design generator. During training, given a set of elements along with a canvas (described by its color, height and width) from a graphic design, each element (and the canvas) is represented as a sequence of discrete attribute tokens, which are then projected into a sequence of embeddings. An ordering network takes as input the embeddings of all the elements and predicts an ordering (i.e., permutation) of the elements, which is used to reorder the embeddings in an element-wise manner. A complete sequence of embeddings is formed by flattening the reordered attribute embeddings into 1D sequence and prepending the `[BOS]` embedding and the canvas embeddings to it. The generated embedding sequence are processed by a series of Transformer decoder blocks (with a causal attention mask) to predict the attribute tokens autoregressively. The entire model is trained end-to-end by optimizing an autoregressive loss *without* direct supervision on the outputs of the ordering network.

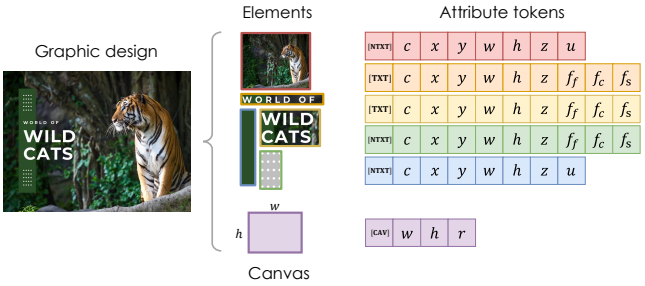


Fig. 3. Design representation. A graphic design is decomposed into a canvas and a set of basic element. Each element is represented as a subsequence of discrete attribute tokens, starting with either `[TXT]` (text) or `[NTXT]` (non-text) to indicate its general category. The canvas is also converted into a discrete subsequence with `[CAV]` at the beginning. All the subsequences are concatenated to produce a sequential representation of the design. The design is from VistaCreate (© Crello).

discrete attribute tokens, with the `[TXT]` or `[NTXT]` token added at the beginning to indicate its category. Similarly, the canvas can be represented as a discrete subsequence starting with a `[CAV]` token. Then, the N elements in the design can be represented as a sequence of element subsequences: $\mathbf{s}^{\mathrm{e}} = (\mathbf{s}^{\mathrm{e}}_{\pi_1}, \mathbf{s}^{\mathrm{e}}_{\pi_2}, \cdots, \mathbf{s}^{\mathrm{e}}_{\pi_N})$, where $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^{N}$ is a permutation of the set $\{1, 2, \cdots, N\}$. By combining $\mathbf{s}^{\mathrm{e}}$ with the canvas subsequence $\mathbf{s}^{\mathrm{c}}$, we obtain a design sequence:

$$\mathbf{S} = (\texttt{[BOS]}, \mathbf{s}^{\mathrm{c}}, \mathbf{s}^{\mathrm{e}}_{\pi_1}, \mathbf{s}^{\mathrm{e}}_{\pi_2}, \cdots, \mathbf{s}^{\mathrm{e}}_{\pi_N}, \texttt{[EOS]}), \tag{1}$$

where `[BOS]` and `[EOS]` are two special tokens that denote the beginning and end of the sequence, respectively. Moreover, we also introduce a `[PAD]` token to pad variable-length design sequences in a batch during training.

3.2 Design Generator

The design generator learns a distribution $p_\theta(\mathbf{S})$ of design sequences $\mathbf{S}$, where θ contains trainable weights. We can sample from $p_\theta(\mathbf{S})$ to generate realistic and diverse designs. While our GOL framework is flexible to use a variety of generative modeling approaches, we opt to exploit an autoregressive (AR) model based on a Transformer backbone as the designer generator, because 1) its effectiveness has been demonstrated by existing layout generation methods [Gupta et al. 2021; Horita et al. 2024; Jiang et al. 2023a] and 2) the AR model is sensitive to the order of input tokens and thus can provide strong learning signal for the ordering network. We later demonstrate that the neural order learned with the AR generator can generalize well to other types of generative models in Section 5.2.3.

Architecture. The architecture of our design generator is shown in Figure 2. It is a standard Transformer decoder (with a causal attention mask), but with an ordering network inside it to dynamically permute the token embeddings in training. For a training design sequence of the format as described in Equation 1, we assume that it has an identity element permutation, i.e., $\pi_i = i$, for $i = 1, \ldots, N$. That is, the training design sequence has the form $\mathbf{S}_0 = (\texttt{[BOS]}, \mathbf{s}^{\mathrm{c}}, \mathbf{s}^{\mathrm{e}}_1, \mathbf{s}^{\mathrm{e}}_2, \cdots, \mathbf{s}^{\mathrm{e}}_N, \texttt{[EOS]})$, where the order of elements is arbitrary. To facilitate batch processing in training, all the element subsequences in $\mathbf{S}_0$ are padded to have the same length K with special `[PAD]` tokens (with all-zeros embeddings) before being fed

into the design generator. Just as in a standard Transformer decoder, the padded sequence is linearly projected into a sequence of attribute embeddings $\mathbf{H}_0 = (\mathbf{h}^c, \mathbf{h}^e_1, \mathbf{h}^e_2, \ldots, \mathbf{h}^e_N)$, where $\mathbf{h}^c \in \mathbb{R}^{d\times 3}$ are the embeddings of the canvas and $\mathbf{h}^e_i \in \mathbb{R}^{d\times K}$ are the embeddings of the i-th element. The embeddings of all the elements $(\mathbf{h}^e_1, \mathbf{h}^e_2, \ldots, \mathbf{h}^e_N)$, which encode the elements' semantic, spatial and visual features, are then processed by the ordering network, resulting in an ordering (permutation) $\boldsymbol{\pi}$ of the elements. $\boldsymbol{\pi}$ is used to permute $(\mathbf{h}^e_1, \mathbf{h}^e_2, \ldots, \mathbf{h}^e_N)$ *element-wise*, giving rise to the permuted embeddings $(\mathbf{h}^e_{\pi_1}, \mathbf{h}^e_{\pi_2}, \ldots, \mathbf{h}^e_{\pi_N})$. This is effectively equivalent to reordering the element attributes in the input sequence $\mathbf{S}_0$. $\mathbf{h}^c$ and $(\mathbf{h}^e_{\pi_1}, \mathbf{h}^e_{\pi_2}, \ldots, \mathbf{h}^e_{\pi_N})$ are recombined to yield an embedding sequence $\mathbf{H} = (\mathbf{h}^c, \mathbf{h}^e_{\pi_1}, \mathbf{h}^e_{\pi_2}, \ldots, \mathbf{h}^e_{\pi_N})$. After removing the `[PAD]` embeddings from $\mathbf{H}$, prepending a `[BOS]` embedding to it and adding position embeddings, a stack of Transformer decoder blocks processes the resulting sequence, producing a sequence of output embeddings in an autoregressive manner. Finally, each output embedding is linearly projected to class logits, on which a softmax function is applied to produce the conditional distribution at each position in the output sequence.

Training. Let $\mathbf{S}$ be the reordered design sequence obtained by applying the permutation $\boldsymbol{\pi}$ to the input training sequence $\mathbf{S}_0$. The AR design generator models the probability of $\mathbf{S}$ of length T as:

$$p_\theta(\mathbf{S}) = \prod_{t=2}^{T} p_\theta(\mathbf{S}_t|\mathbf{S}_{1:t-1}). \tag{2}$$

The model is trained by minimizing the autoregressive objective (the negative data log-likelihood):

$$\mathcal{L}_{\text{AR}} = \sum_{\mathbf{S}\in\mathcal{D}} -\log p_\theta(\mathbf{S}), \tag{3}$$

where $\mathcal{D}$ is a dataset of training samples.

3.3 Ordering Network

Architecture. The input to the ordering network is the attribute embeddings for each element $(\mathbf{h}^e_1, \mathbf{h}^e_2, \ldots, \mathbf{h}^e_N)$. It estimates an ordering of the elements that is used to perform element-wise reordering of the embeddings. First, a score network aggregates the attribute embeddings of each element into an element-level embedding, and passes them through a Transformer encoder followed by a MLP to compute per-element scores $(s_1, s_2, \ldots, s_N)$. The scores are then fed into a differentiable neural sorting module, which outputs a permutation $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_N\}$ of the set $\{1, 2, \ldots, N\}$. π_i specifies where the i-th element in the reordered sequence appears in the original sequence. $\boldsymbol{\pi}$ is then applied to $(\mathbf{h}^e_1, \mathbf{h}^e_2, \ldots, \mathbf{h}^e_N)$ to generate the permuted embeddings $(\mathbf{h}^e_{\pi_1}, \mathbf{h}^e_{\pi_2}, \ldots, \mathbf{h}^e_{\pi_N})$. More specifically, let $\mathbf{P}_{\boldsymbol{\pi}} \in \{0, 1\}^{N\times N}$ be the corresponding permutation matrix of $\boldsymbol{\pi}$, with $\mathbf{P}_{\boldsymbol{\pi}}[i, j]$ as:

$$\mathbf{P}_{\boldsymbol{\pi}} i[i, j] = \begin{cases} 1, & \text{if } \pi^i = j \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

We organize the attribute embeddings of all the elements into a tensor $\mathbf{U} \in \mathbb{R}^{N\times K\times d}$, where N, K, d are the number of elements, the number of attributes per element and the embedding dimensionality, respectively. The permuted embeddings can be obtained simply as $\mathbf{P}_{\boldsymbol{\pi}}\mathbf{U} \in \mathbb{R}^{N\times K\times d}$, which is then reshaped into $(\mathbf{h}^e_{\pi_1}, \mathbf{h}^e_{\pi_2}, \ldots, \mathbf{h}^e_{\pi_N})$.

Training. During training, we also regularize the output scores of the score network to prevent them from collapsing to similar values by introducing an additional regularization objective, which pushes the scores apart from each other:

$$\mathcal{L}_{\text{REG}} = \frac{1}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} max(0, \lambda - |s_i - s_j|), \tag{5}$$

where the loss will go to zero if the difference between two scores is greater than a margin hyperparameter λ (set to 1e-5). As a result, our final objective is:

$$\mathcal{L} = \mathcal{L}_{\text{AR}} + \beta\mathcal{L}_{\text{REG}}, \tag{6}$$

where $\beta = 0.01$ is a weighting parameter to control the contribution of $\mathcal{L}_{\text{REG}}$. By optimizing our model end-to-end on a graphic design dataset using $\mathcal{L}$, we obtain a trained ordering network, defining an ordering approach, which we refer to as neural order. While the ordering network and the design generator are coupled in training, the ordering network can be used independently of the design generator in inference to sort graphic design elements. In addition, compared to random and raster order, our ordering approach makes ordering decisions based on a wider spectrum of element attributes, ranging from low-level appearance (e.g., color, texture), mid-level properties (e.g., layout), to high-level semantics (e.g., an object's category in an image.)

4 EXPERIMENTAL SETUP

4.1 Datasets

We conduct experiments on the three publicly available datasets of various design domains: Crello [Yamaguchi 2021], CGL [Zhou et al. 2022], CLAY [Li et al. 2022]. Crello is a dataset of vector graphic designs, each of which is a structured documents describing a variety of attributes (e.g., layout, color, image) for each element. Crello spans a wide range of design domains, such as blog header, banner, printed poster, and includes four main element categories: text, scalable vector graphics (SVG), image, and background. CGL comprises rasterized advertising posters of commercial products, each of which has a background image, and a set of elements annotated with their categories and bounding boxes. CGL contains four element categories including text, logo, underlay and embellishment. CLAY contains a large corpus of annotated mobile UI screenshots, where each UI element is labeled as one of 25 semantic categories and has its annotated bounding box coordinates. In general, designs in CLAY are more complex than those of Crello and CGL, with a larger number of elements per design. However, the design patterns in CLAY are relatively regular and repetitive, in order to accommodate the limited display space and dominant top-to-down reading pattern on mobile devices. Compared to Crello and CLAY, CGL has simpler and less diverse designs: every design is composed of a large background image that spans the entire canvas, with a sparse set of elements spreading on top of it. In contrast, Crello encompasses a large variety of design variations since its designs come from several different domains. Furthermore, Crello lies in-between CGL and CLAY in terms of design complexity. More importantly, designs in Crello

have a significantly richer set of element attributes beyond category and bounding box coordinates. In view of these favorable properties, we opt to use Crello as our main dataset for the subsequent evaluation and analysis. For Crello, we first discard designs that can not be rendered properly due to the lack of some rendering-relevant information (e.g., CSS styles, rotation and opacity of elements). Then, after excluding designs containing very small elements (occupying less than 0.01% of the canvas), we obtain 10,162 samples (average element number per design = 9.35). For CGL, we filter out samples with less than 6 elements to ensure sufficient design complexity, leading to 28,747 samples (average element number per design = 7.39). For CLAY, we obtain a set of high-quality and more diverse designs via the following filtering process: first, we remove samples containing invalid elements based on CLAY's annotations; second, we ignore too complex designs by only retaining designs with no more than 20 elements; third, we discard samples with misaligned annotations; fourth, we drop designs with too regular layouts to encourage more design variations. This results in 19,631 samples (average element number per design = 10.19).

4.2 Training, Inference and Rendering

4.2.1 Neural Order Learning. To learn a neural order, we optimize the full objective (Equation 6) to train our GOL on a graphic design dataset for 100 epochs. We use the AdamW optimizer [Loshchilov and Hutter 2017], with $\beta_1 = 0.9$ and $\beta_2 = 0.95$ for the ordering network, and $\beta_2 = 0.999$ for the autoregressive generator. We use a constant learning rate of 2.5e-4, and a batch size of 256. Gradient clipping is also used for the ordering network. Throughout training, we periodically save the checkpoints of our ordering network. For each checkpoint, we use the ordering network with the corresponding weights to order elements in each design of the dataset, producing a re-ordered dataset. We then train the design generator (without the ordering network) on the re-ordered dataset with the approach in Section 4.2.2, and use its generated samples to compute SeqFID. Finally, we take the checkpoint with the lowest SeqFID and treat the corresponding ordering network as the learned neural order.

4.2.2 Design Generator Training and Inference. Given a candidate order (e.g., our neural order or one of the baseline order in Section 4.4), we train a design generator on a graphic design dataset whose elements are permuted by the order. In our experiment, we train autoregressive and diffusion-based generators using the AdamW optimizer. We train the autoregressive generator using the AR loss function (Equation 3) for 800 epochs, with a batch size of 256 and a learning rate of 5e-5. We maintain an exponential moving average (EMA) of the model weights over training, with a decay rate of 0.9999. For inference, we use the EMA model, employing nucleus sampling with $p = 0.95$. To train a diffusion generator (Section 5.2.3), we largely follow the training protocol from [Inoue et al. 2023a], but without their learning rate schedule. We train the generator for 2,000 epochs using a batch size of 512 and a learning rate of 5e-4, and maintain an EMA of the model weights throughout training. At inference time, we use random sampling on the EMA model.

4.2.3 Design Rendering. Given a generated design sequence, we render it into a multi-layer, editable graphic design in SVG format using the SVG render of [Inoue et al. 2023b]. For discrete bounding box tokens in the sequence, we use the center values of their corresponding bins as the continuous bounding box coordinates. For a font size token, we also directly use its bin center value. For a font color token, we add a small amount of random perturbation to the value of its bin center to introduce some color variations. Give an image token, we retrieve an existing image from a set of candidate images inside its bin in the training dataset. The retrieved image is returned by sampling a categorical distribution over the candidate images, where the probabilities are obtained by applying a softmax function to the distances between the candidate images and the bin center in the image feature space described in Section 3.1. This will introduce some randomness, which can increase the visual diversity of rendered designs, while ensuring that candidate images closer to the bin center is more likely to be retrieved. To compute VisFID and VisCoverage, we utilize an open-source tool [Ayoub et al. 2024] to convert the SVG into an image.

4.3 Evaluation Metrics

Inspired by literature in generative models of images, we employ Fréchet Inception Distance (*FID*) [Heusel et al. 2017] as the primary metric to evaluate overall sample quality. FID measures the distance between the distributions of real and generated samples in an embedding space. We additionally report *Density* and *Coverage* [Naeem et al. 2020] as secondary metrics, to gauge sample fidelity and diversity, respectively.

Since all the metrics rely upon a pre-trained embedding space, we consider two embedding spaces for a more comprehensive assessment of generation performance: a sequence embedding space and a visual embedding space. For the sequence embedding space, we train a transformer-based autoencoder on each dataset and use its encoder to extract embeddings from design sequences generated by design generators. The sequence embeddings enable use to compute *SeqFID*, *SeqDensity* and *SeqCoverage*, which directly measure the quality of generated design sequences. To learn the visual embedding space, we train a masked autoencoder (MAE) [He et al. 2022] on the rendered images of designs in Crello from scratch. We compute *VisFID*, *VisDensity* and *VisCoverage* based on the visual embeddings to evaluate the perceptual quality of design images. To calculate these two metrics, we project the images rendered from design sequences output by design generators into the visual embedding space through the MAE encoder. It should be noted that VisFID, VisDensity and VisCoverage are sensitive to the rendering method (as described in Section 4.2.3, and thus may not accurately reflect the generative capabilities of design generators; however, they are used in the evaluation to give a quantitative sense of how well the models perform in real use cases where generated design sequences need to be rendered into multi-layer designs (in SVG format) or images.

Given the limited scale of the three graphic design datasets in Section 4.1, we compute the above four metrics against the entire training set, following some image generative modeling works [Brock 2018; Dhariwal and Nichol 2021; Heusel et al. 2017].

4.4 Baselines

We compare our neural order with two commonly used orders in prior studies [Gupta et al. 2021; Inoue et al. 2023a; Zhang et al. 2023]: *random order* and *raster order*. Random order is implemented by randomly shuffling elements in a design, while raster order arranges elements in a design by sorting the top-left coordinates of their bounding boxes first by y-coordinate (from top to down) and then by x-coordinate (from left to right).

We additionally evaluate a saliency-based ordering approach, *saliency order*, which sorts elements in a design in descending order of their importance. Specifically, given a design, we leverage an off-the-shelf visual saliency detector [Jiang et al. 2023b] to predict its saliency map. Then, for each element, we estimate its importance score by summing up the saliency values within the element. To reduce the impact of element size, we omit the pixels of the element whose saliency values are below a threshold (the 25th percentile of all saliency values inside the element). We also exclude the occluded part of an element when computing its importance score, based on the layer depth of elements. For Crello, the depth information is available; for CGL and CLAY, we approximate layer depth based on some simple heuristic rules (see the supplemental for more details).

We also consider two layer-based ordering methods: *layer order* and *layer-and-raster order*. Layer order sorts elements from bottom to top, based on their layer depth. On CGL and CLAY where layer depth is not available, we adopt the same heuristic methods used in saliency order to estimate depth information. Layer-and-raster order first groups elements based on their categories into a text layer, a foreground layer and a background layer, and ranks the layers in the order of background → foreground → text, which is common ordering practice in layered graphic designs. Then, it sorts elements within each layer by raster order. For Crello, the foreground layer contains SVG elements, while image and background elements are in the background layer. For CGL, the foreground layer contains logo, underlay and embellishment elements, with background elements in the background layer. For CLAY, the background layer contains image, pictogram and background elements, and the text layer contains text and text input elements, while the remaining elements are in the foreground layer.

5 EXPERIMENTS

We first examine the impact of neural order on model training (Section 5.1). We then investigate how neural order can improve design generative models across multiple datasets, and evaluate the generalization performance of neural order learned with an autoregressive approach to a diffusion-based model (Section 5.2). We also extensively analyze neural order, gaining its insight into ordering behaviors (Section 5.3). Finally, we probe the ability of neural order in helping design generators scale up excellently (Section 5.4).

5.1 How does neural order help training?

To study how neural order influences the training procedure of design generators, we train our GOL on the Crello dataset to learn a neural order, and train six variants of our autoregressive (AR) generator on Crello, each using a different order. Figure 4 shows how the AR loss of the six generators change over training. Random order performs the worst among all the orders, showing significantly slower convergence and higher loss values. This is an indication that finding a reasonable order is a critical ingredient for training design generative models effectively and efficiently. Our neural order outperforms all the other orders, enabling the model to learn faster and converge to a lower loss value. This suggests that the effectiveness of our neural order in improving the training process.

5.2 How does neural order improve generation performance?

5.2.1 Evaluation on Crello. We conduct comprehensive experiments on the Crello dataset, with diverse designs with adequate complexity and rich element attributes, to evaluate how well our neural order can contribute to the improvement of design generators in visual quality. To do this, we learn a neural order by optimizing our GOL on Crello, and use it to train our AR generator on Crello. For each of the baseline orders, we also train a separate AR generator on Crello. The quantitative results of different orders are reported in Table 1. Our neural order outperforms the other orders by large margins in terms of both SeqFID and VisFID. We also find that our neural order achieves higher SeqCoverage, VisDensity and VisCoverge scores compared to the other orders. This suggests element ordering approaches can critically affect the quality of generative models for graphic designs, and more importantly, training with our learned neural order, rather than the commonly used ones (i.e., random order and raster order), can yield noticeable improvement in sample quality.

Figure 5 visually compares samples obtained from the models trained using the best four orders in terms of VisFID (as shown in Table 1), including random, raster, saliency and neural orders. The samples for the two layer-based orders can be found in the supplemental.

The results from random order suffers from some obvious and severe design problems, such as significant amounts of misalignment between elements (e.g., row 6), undesirable overlap between elements (e.g., row 7) and simplistic outputs with two few elements (e.g., row 1), and excessive white space (e.g., row 8). Raster order improves visual quality upon random order, but exhibits limited layout variation with a strong bias towards spatially centering elements in the design, which can be observed, e.g., by comparing the samples in row 1, 3, 4 and 5. Furthermore, raster order has difficulty in placing some SVG or text elements properly when a background image is present, resulting in some important image regions being occluded. For example, the bike SVG blocks the woman in row 7, and the text partially occludes the human faces in row 8. Saliency order is prone to generating designs with a variety of images, which enhances visual interest. Unfortunately, its results lack font color diversity, predominantly using black color for text regardless of the background image content, which often degrades text readability (e.g., row 3 and row 5). In addition, saliency order sometimes fall short of finding visually harmonious and semantically relevant colors—e.g., the yellow block on the right side of the design in row 2 looks visually incompatible with its context, and in row 3, the pure dark blue background may suppress the "delicious" feeling of the food on the right. In contrast, our neural order enables the

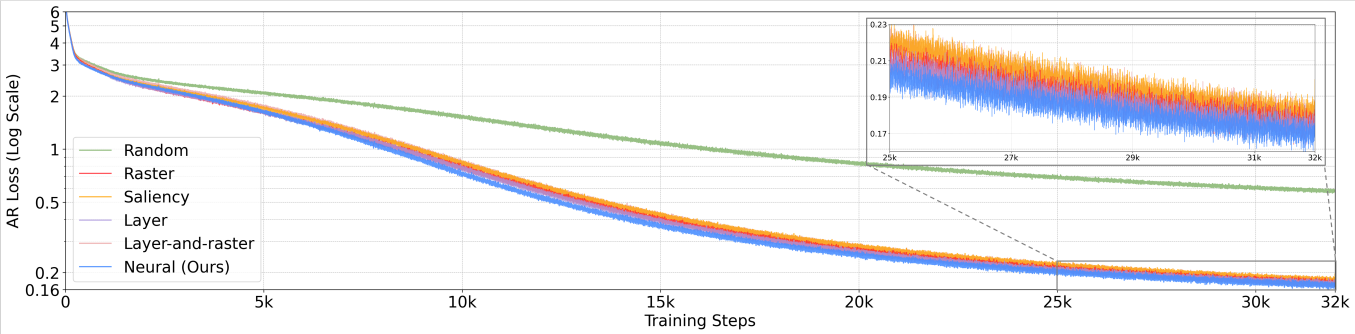


Fig. 4. Loss curves of training our AR generator with different orders. Our neural order leads to faster convergence and ends up with a lower loss value compared to the other orders.

Table 1. Quantitative results of our AR generators trained with different orders on Crello dataset. The best results are in **bold**; the second best results underlined.

Dataset	Order	SeqFID ↓	SeqDensity ↑	SeqCoverage ↑	VisFID ↓	VisDensity ↑	VisCoverage ↑
Crello	Random	19.64	1.46	0.923	5.82	0.83	0.899
	Raster	6.28	**1.55**	0.904	4.13	1.26	0.923
	Saliency	6.01	1.49	0.871	3.57	1.22	0.908
	Layer	6.83	1.46	0.899	7.21	1.22	0.906
	Layer-and-raster	6.65	1.49	0.884	7.05	1.21	0.913
	Neural (Ours)	**3.25**	1.53	**0.954**	**1.79**	**1.28**	**0.969**

model to generate outputs of varying complexity and diverse design patterns, with many favorable properties including: high-quality layouts (with well-aligned elements and great layout variations), avoiding occlusion of important image contents, high text readability (due to high text-background color contrast), and harmonious and contextually appropriate colors.

To further gauge the perceptual quality of designs resulting from different orders, we conduct a user study, involving 28 participants(including both novice and expert designers). The participants are shown six groups of designs, where each group corresponds to one order and contains 3 generated designs, and asked to choose their favorite design group by considering several aspects including composition, visual consistency and typography. The results are shown in Figure 6. Our neural order is the best choice 43.33 % of the time, outperforming the other orders by significant margins.

5.2.2 Evaluation on CLAY and CGL. We also perform evaluations on CLAY and CGL datasets. Unlike Crello, these two datasets are rasterized graphic designs, where elements are annotated with only categories and bounding box coordinates without other attributes, such as layer depth, font family and color. Consequently, we modify the design representation in Section 3.1 to address the characteristics of these datasets. We represent each element, be it text or non-text, as a set of attributes (c, x, y, w, h, u), with subcategory c, bounding box (x, y, w, h) and image u. For both text and non-text elements, we crop their regions on the design as their images. We follow the same procedure as in Section 5.2.1 to train GOL on each of the two datasets for neural order learning, followed by training the AR generator with the neural order on the dataset.

The results are shown in Table 2. Since CGL and CLAY lack necessary information for high-quality rendering of design sequences, we only report SeqFID, SeqDensity and SeqCoverage. On CGL, our neural order yields the best results on all the metrics. On CLAY, our neural order performs the other orders in terms of SeqFID and SeqDensity, while achieving the second best SeqCoverage, which is slightly lower than that of random order. Note that, although random order can well cover the data distribution, its sample fidelity is greatly compromised with the worst SeqFID. In contrast, our neural order can produce high-fidelity samples while maintaining good diversity.

5.2.3 Generalizability to Diffusion-based Generator. In the experiments so far, we have learned neural orders with our GOL (based on an *autoregressive* generator) and demonstrated that they can noticeably boost the quality of *autoregressive* design generation models. It is natural to ask if the neural orders can generalize to benefit other types of generative modeling frameworks. Motivated by the astonishing performance of recent diffusion models in image generation, we study this question by learning a neural order with our GOL on Crello and training a discrete diffusion model [Inoue et al. 2023a] with the neural order on Crello. As opposed to our AR generator, the diffusion model exploits the neural backbone composed of bidirectional Transformer encoder blocks. For training the diffusion model, we modify the design representation in Section 3.1, by filling each potentially missing attribute (e.g., font color for a non-text element)

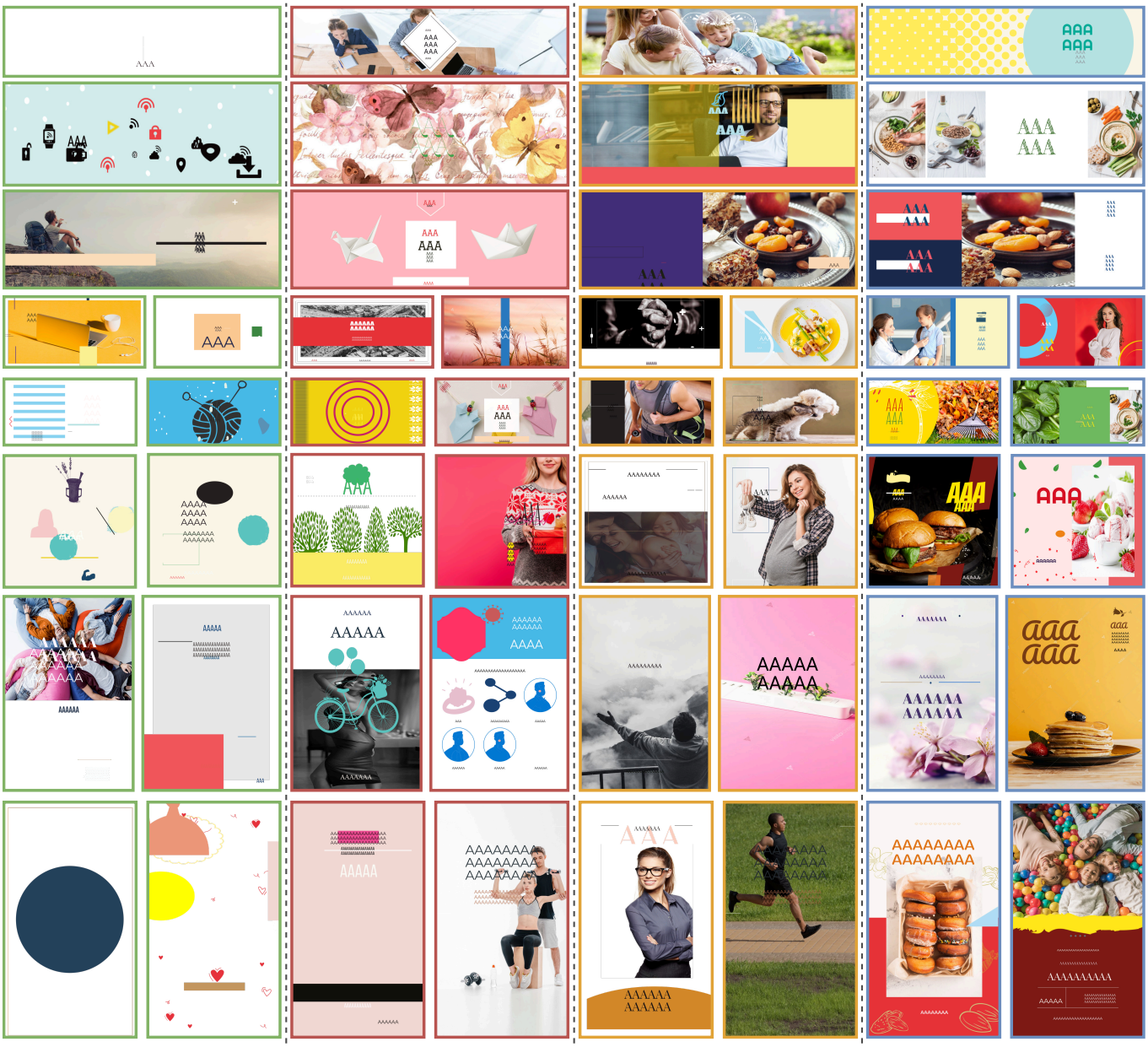


Fig. 5. Qualitative comparison of samples generated from models trained with different orders. Training with our neural order yields noticeable improvements in generation quality.

with a separate [NULL] (with a learnable embedding), instead of the [PAD] token.

As shown in Table 3, our neural order still outperforms the other five orders on all the metrics. This indicates that our GOL has learned a *general-purpose* ordering strategy, independent of the architecture of the design generator, which can enable benefits for a wide range of generative model classes for graphic design modeling.

Since the backbone architecture of the diffusion model is permutation invariant, we also experiment with training the model *without* position embeddings. This is equivalent to viewing designs as unordered sets, instead of sequences. However, we find that this variant performs significantly worse (with SeqFID 623.66), implying that it is advantageous to make generative models of graphic designs have a notion of order of elements. This further backs the common practice of recent studies that add position embeddings to the inputs

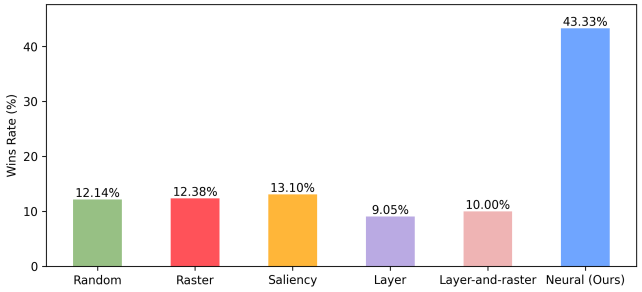


Fig. 6. Results of the user study comparing designs generated by the models trained with different orders. Our neural order is strongly preferred over the other orders by humans.

Table 2. Quantitative evaluations of our AR generators trained with different orders on CGL and CLAY. The best results are in **bold**; the second best results are underlined.

Dataset	Order	SeqFID ↓	SeqDensity ↑	SeqCoverage ↑
CGL	Random	3.98	1.04	0.946
	Raster	2.81	**1.65**	0.966
	Saliency	2.47	1.63	0.974
	Layer	2.43	1.62	0.974
	Layer-and-raster	2.46	1.64	0.971
	Neural (Ours)	**1.96**	**1.65**	**0.986**
CLAY	Random	23.82	1.52	**0.761**
	Raster	19.10	1.68	0.707
	Saliency	18.11	1.69	0.736
	Layer	16.53	1.65	0.751
	Layer-and-raster	17.94	1.64	0.728
	Neural (Ours)	**15.23**	**1.71**	0.756

to the permutation-invariant underlying architectures [Inoue et al. 2023a; Zhang et al. 2023].

5.3 How does neural order rank elements?

Having observed that our neural order can improve both training efficiency (Section 5.1) and generation quality (Section 5.2), we dive deeper into our neural order, attempting to understand how it makes ordering decisions. To this end, we analyze the behaviors of a neural order learned from the Crello dataset.

5.3.1 Element Categories at the Top and Bottom. We first study what element categories are ranked higher or lower by our neural order in a sequence. For each order, we visualize the distributions of element categories near the top and bottom of a sequence in Figure 7. Compared to the other orders, the neural order has a strong trend of putting text elements at the beginning of the sequence *before* elements of other categories. Random order and raster order have similar distributions, mainly because they are both agnostic of element categories (raster order sorts elements merely based on their bounding box coordinates). Saliency order tends to place SVG elements towards the end of the sequence, while some text and image elements are ranked higher possibly due to their visual importance. Layer order prefers placing text elements last and image/background elements first, since the text elements often appear on the top of layered graphic designs, while image/background elements are put on the bottom. Layer-and-raster order follows the same trend as layer order because it ranks the text layer last and the background layer (consisting of image/background elements) first.

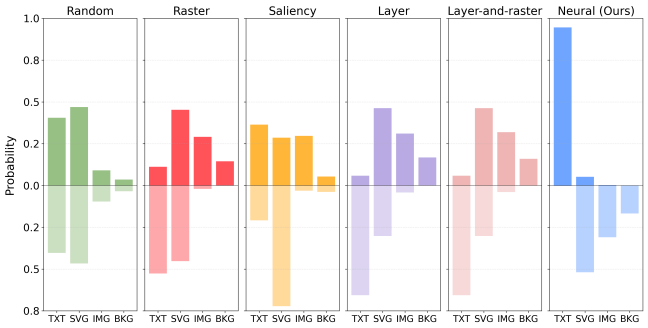


Fig. 7. Distributions of elements categories near the top and bottom of a sequence. Four main categories on Crello are considered: text (TXT), scalable vector graphics (SVG), image (IMG), background (BKG). For each order, we show the distributions of element categories in the top 25% (top view) and bottom 25% ((bottom view)) of the sequence. Our neural order prefers to place text element before other kinds of elements.

5.3.2 Adjacency between Element Categories. We also analyze the temporal behaviors of different orders along a sequence, by studying what is the most likely succeeding element category for a given element category in a sequence. For each order, we compute a $N \times N$ adjacency matrix, where N is the number of element categories and the entry M_{ij} represents the probability of category i being followed by category j in a sequence. The results are shown in Figure 8. We observe that adjacency matrix of the neural order has higher values on the diagonal than those of the other orders. This implies that our neural order prefers to place elements of one category before placing elements of another category along a sequence, causing higher adjacency probabilities between elements of the same category.

5.3.3 Beyond Category-based Ordering. The findings in Section 5.3.1 and Section 5.3.2 show that our neural order has strong patterns of ordering element categories. This suggests element categories contain important information for the ordering decisions of the neural order. Therefore, we are interested in studying whether our neural order have learned a more sophisticated ordering strategy beyond only the simple category-based sorting as found in in Section 5.3.1 and Section 5.3.2. For this, we consider a *text-first* baseline, which 1) places all text elements before non-text elements, and 2) groups the non-text elements by their subcategories (e.g., SVG and image), with the order of the groups randomized. We emphasize that *such text-first order is enabled through empirical study on our learned neural order*, which has not been explored by any existing literature.

We compare our neural order with the text-first order quantitatively on Crello, and report the results in Table 4. With each order, we train our AR generator and the diffusion generator (Section 5.2.3). We find that our neural order outperforms the text-first baseline on

Table 3. Quantitative results of diffusion generators trained with different orders on the Crello dataset. The best results are in **bold**; the second best results are underlined.

Dataset	Order	SeqFID ↓	SeqDensity ↑	SeqCoverage ↑	VisFID ↓	VisDensity ↑	VisCoverage ↑
Crello	Random	2.92	1.28	0.950	4.48	0.92	0.910
	Raster	2.50	1.39	0.969	4.33	0.97	0.915
	Saliency	2.52	1.35	0.966	3.75	0.96	0.911
	Layer	2.43	1.38	0.965	3.14	0.94	0.914
	Layer-and-raster	2.51	1.35	0.957	3.57	0.95	0.909
	Neural (Ours)	**2.40**	**1.40**	**0.970**	**2.88**	**0.99**	**0.923**

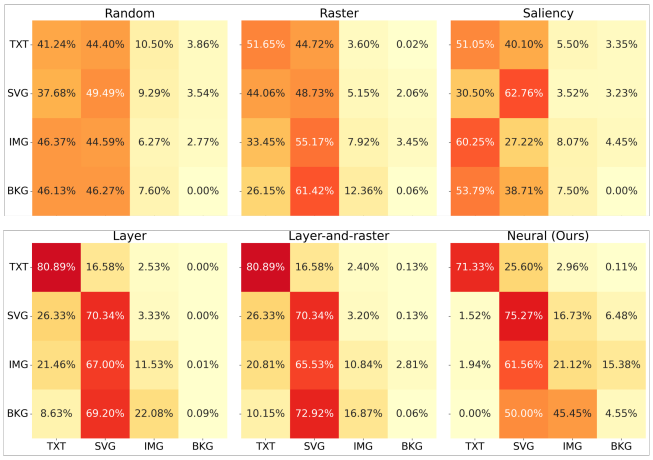


Fig. 8. Adjacency matrices of different orders, specifying the probabilities of one element category being followed by another in a sequence. Our neural order shows larger values along the diagonal, with a strong tendency to group the elements of the same category together in a sequence.

Table 4. Performance comparison of text-first order and neural order on the Crello dataset. For each order, an autoregressive generator (AR) and a diffusion generator (DM) are trained.

	AR		DM	
	Text-first	**Neural**	Text-first	**Neural**
SeqFID ↓	3.31	**3.25**	2.75	**2.40**
SeqDensity ↑	1.52	**1.53**	1.36	**1.40**
SeqCoverage ↑	0.951	**0.953**	0.958	**0.970**
VisFID ↓	2.13	**1.79**	3.90	**2.88**
VisDensity ↑	1.27	**1.28**	0.97	**0.99**
VisCoverage ↑	0.968	**0.969**	0.913	**0.923**

all the metrics as well as across all the generative approaches. These results demonstrate that our neural order goes beyond merely sorting elements based on their categories, encompassing more complex ordering rules based on other element attributes, which contributes to the performance improvements over the text-first order. This will be investigated in detail in Section 5.3.4.

5.3.4 Content-aware Ordering. We next study how the content of elements influences the decisions of neural order on the Crello dataset.

Graphical Content vs. Order. We first analyze the correlation between the content of non-text elements and their positions in sequences sorted by our neural order. We focus our analysis primarily on SVG elements, for two reasons: first, they take up a majority (78.58 %) of the non-text elements in Crello; second, they span a large variety of visual forms (e.g., shapes, icons, decorative patterns, illustrations), with rich semantic information.

For this study, we use the quantization on the SVG elements (which is used for the AR generator training) into 2048 bins, each of which is a cluster of visually and semantically similar SVG elements. For each bin, we compute a distribution of positions of the bin elements. Since our neural order tends to put text elements before non-text elements and group elements of similar categories together, we consider the position of a SVG element in the subsequence of all the SVG elements. This allows us to reduce the effect of categories (to avoid category information overwhelming content information that we are interested in) and focus more on content-induced ordering patterns. Moreover, to accommodate variable sequence length, we compute a *normalized position* for each SVG element as: its position in SVG-only subsequence divided by the subsequence length.

These position distributions provide an intuitive and quantitative way to study where different graphical contents are positioned in a sequence. For each of the candidate orders (neural order, random order, raster order, saliency order, text-first order), per-bin position distributions are calculated. To quantify the difference between two orders in terms of correlation between graphical content and order, we compute Wasserstein distance [Arjovsky et al. 2017] between the two distributions for each bin, and average across all the bins.

For the following analysis, we compare neural order against text-first order. There are two main reasons for this. First, when considering the order of elements that are only from the same category (e.g., SVG), text-first order degenerates to a random ordering approach (as it uses randomized ordering within each category). Comparison with such random content-agnostic baseline allows us to understand whether neural order have captured any meaningful correlation between the content and order of SVG element. Second, as observed in Section 5.3.3, text-first order is closer to neural order compared to other orders; however, it underperforms neural order. Hence, the comparison here can also help reveal why the neural order performs better than text-first order.

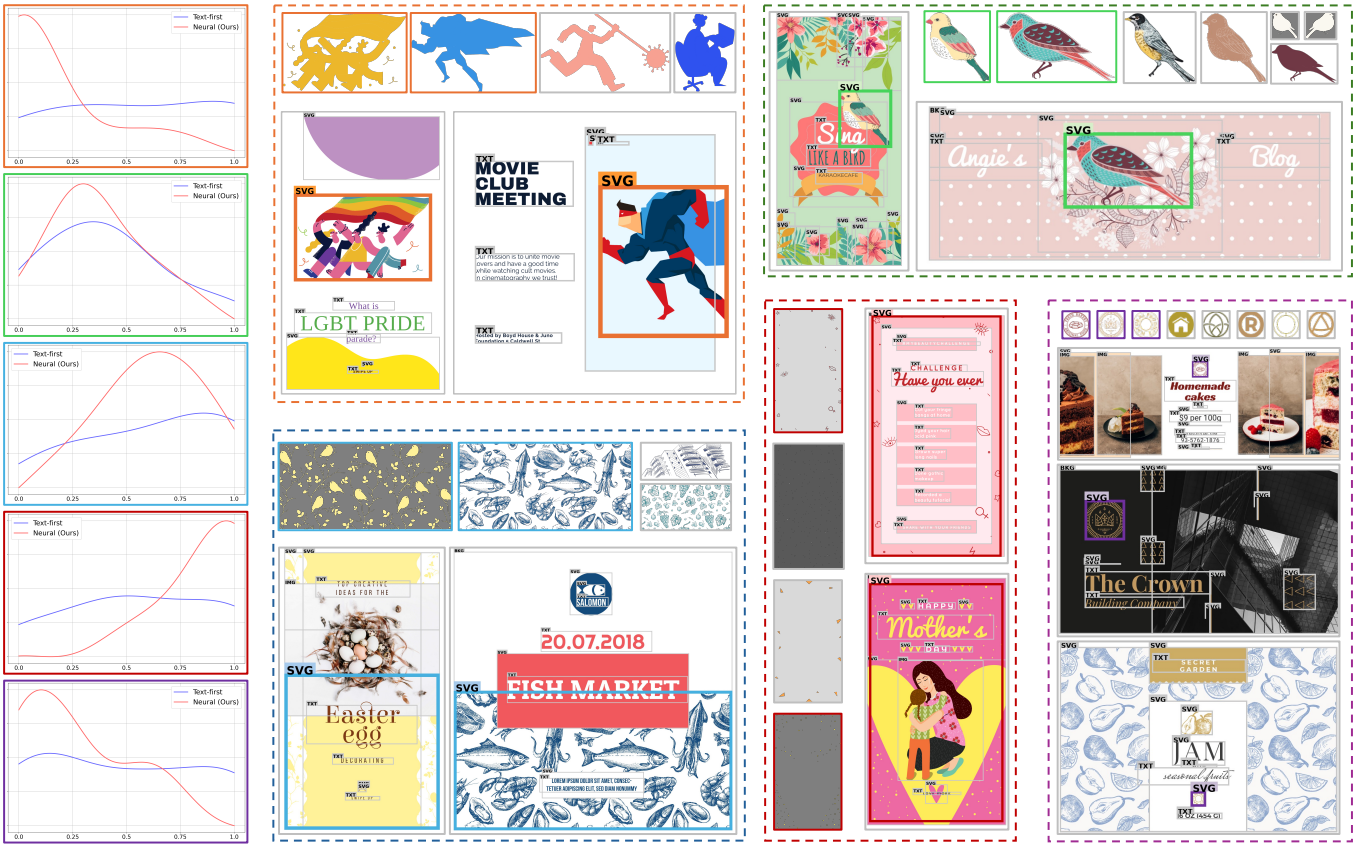


Fig. 9. Visualization of five representative SVG bins, where the differences between position distributions of neural order and text-first order are large. For each bin, the position distributions of the two orders are shown (on the leftmost). Furthermore, some representative SVG elements for each bin are shown, together with several example designs that some of the elements belong to, in order to provide the context of the elements. In each design, the representative element are highlighted with a thicker boarder. Each bin is indicated by a differently colored border. The designs are from VistaCreate (© Crello).

Figure 9 shows some representative SVG bins, on which the discrepancies between the position distributions of neural order and text-first order are large. For each bin, in addition to the two distributions, we show some representative SVG elements from the bin, along with some example designs that use the elements. The text-first order show relatively uniform distributions across all the cases, partially due to its random ordering within each type of elements. In contrast, our neural order exhibits easily-recognizable, distinct ordering patterns for each bin, depending on its semantic content. For example, we observe several intriguing properties of our neural order: 1) in case of the presence of people (orange), the elements has a strong tendency of being put near the beginning; 2) decorative patterns (blue) are more likely to be placed in the bottom half ; 3) birds (green) usually appear around the middle.

These observations suggest that our neural order can make sense of the visual content of graphical elements to decide their positions in the sequences. Furthermore, we note that similar SVG elements in a bin can be used in varying contexts. Hence, the neural order is able to understand not only the semantics of individual elements, but also their contextual relationships to other elements in a design, in order to sort the elements properly.

Font Family vs. Order.

Text elements have three typographic properties that may impact the decisions of our neural order: font family, font size and font color. There exists numerous simple rules of thumb on how to set font size and color—e.g., use a larger font size and a bright font color for important information to make it more eye-catching and memorable. However, how to choose the right font families is much more challenging task, which is not well studied and has received great interest from the research communities [Campbell and Kautz 2014; O'Donovan et al. 2014]. Motivated by this, our study is focused on providing an understanding of the dependency of our neural order upon font family.

For this study, we quantize all the font families in Crello into 64 bins, using their token embeddings extracted from the embedding layer of our GOL. As in Section 5.3.4, we compare our neural order with text-first order, and for each order, we follow the same procedure in Section 5.3.4 to compute position distributions for each bin.

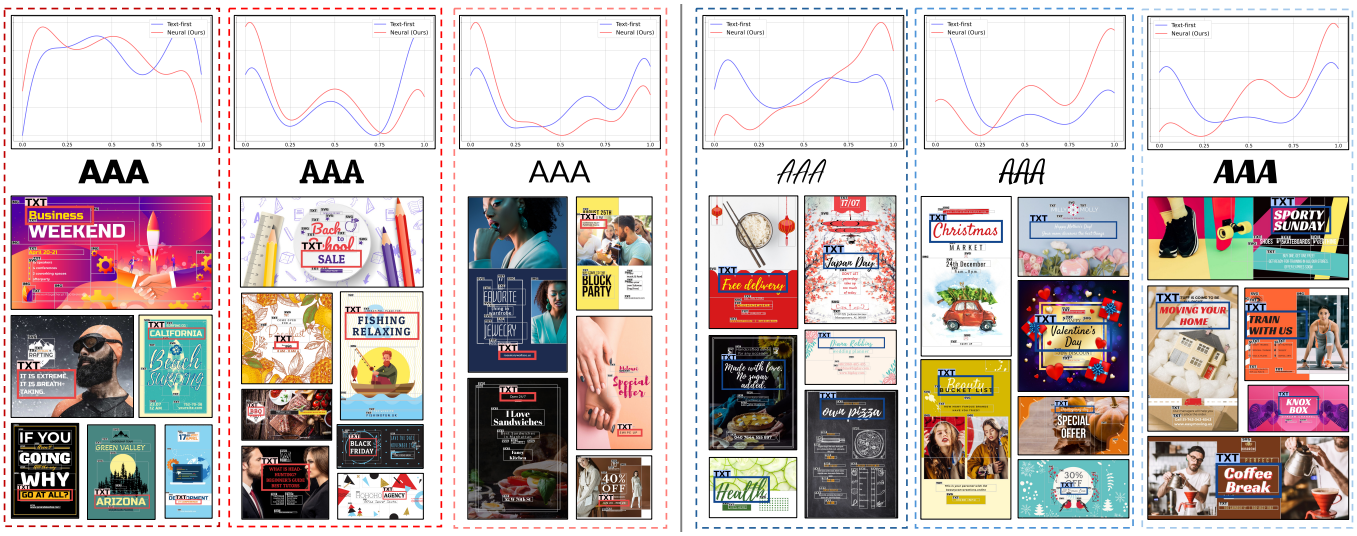


Fig. 10. Visualization of six representative font family bins, where the differences between position distributions of neural order and text-first order are large. For each bin, the position distributions of the two orders are plotted (on the top). We also show a representative font, and some example designs, where the elements using font families from the bin are highlighted with thicker boarders. Each bin is indicated by a differently colored border. The designs are from VistaCreate (© Crello).

Figure 10 visualizes some representative font family bins, with large differences between the position distributions of neural order and text-first order. For each bin, we also show a representative font and example designs whose elements use font family from the bin. We observe that our neural order can determine the positions of text elements based on their font families: font families placed near the front appears to be more formal and regular, while some artistic font families are placed near the end. We conjecture this may be related to the practical process of placing text elements during design composition, where designers may first determine how to place important text that is often rendered with formal fonts for better legibility, and then add more artistic text to increase visual attractiveness of the design.

5.4 Can neural order help design generators scale up?

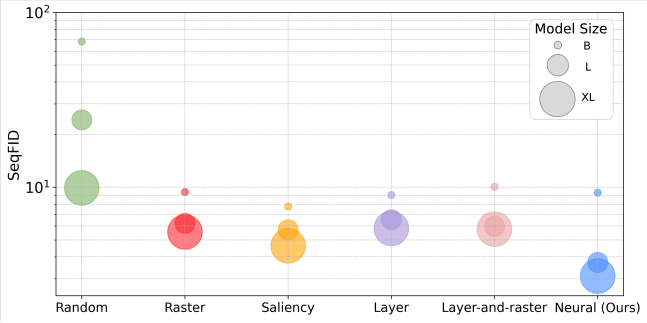


Fig. 11. SeqFID of design generators of different model sizes trained with each order for 24K iterations.

Table 5. Configurations of our autoregressive design generators (ARDG). We consider Base (B), Large (L) and XLarge (XL) variants, with increasing model capacities and compute.

Model	Layer #	Emb. Dim.	Head #	Gflops
ARDG-B	6	512	8	2.11
ARDG-L	12	1024	16	15.83
ARDG-XL	24	1024	16	31.32

Since our design generator is based on the standard Transformer decoder blocks, it inherits the excellent scaling properties of the Transformers. We investigate how well our neural order can help our design generator scale up. We still conduct this scaling experiment on the Crello dataset, where we learn a neural order and compare it with the other orders. For each order, we train three variants of our design generator with different configurations (Table 5): ARDG-B, ARDG-L, ARDG-XL, which span a range of model sizes and computational complexities. Due to the memory constraint, we use a batch size of 64 and a learning rate of 2.5e-5 for all the variants. Figure 11 show the SeqFID of different models at 24K training step. For smaller models (ARDG-B), all the orders except random order perform closely; with larger models (ARDG-L, ARDG-XL), the performance gaps between neural order and the other orders become larger. Our neural order can help the design generator scale up easily, showing substantially improved performance in SeqFID compared to the other orders with larger models. We additionally show the SeqFID of different models over training in Figure 12. All the models start to saturate after 24K training steps.

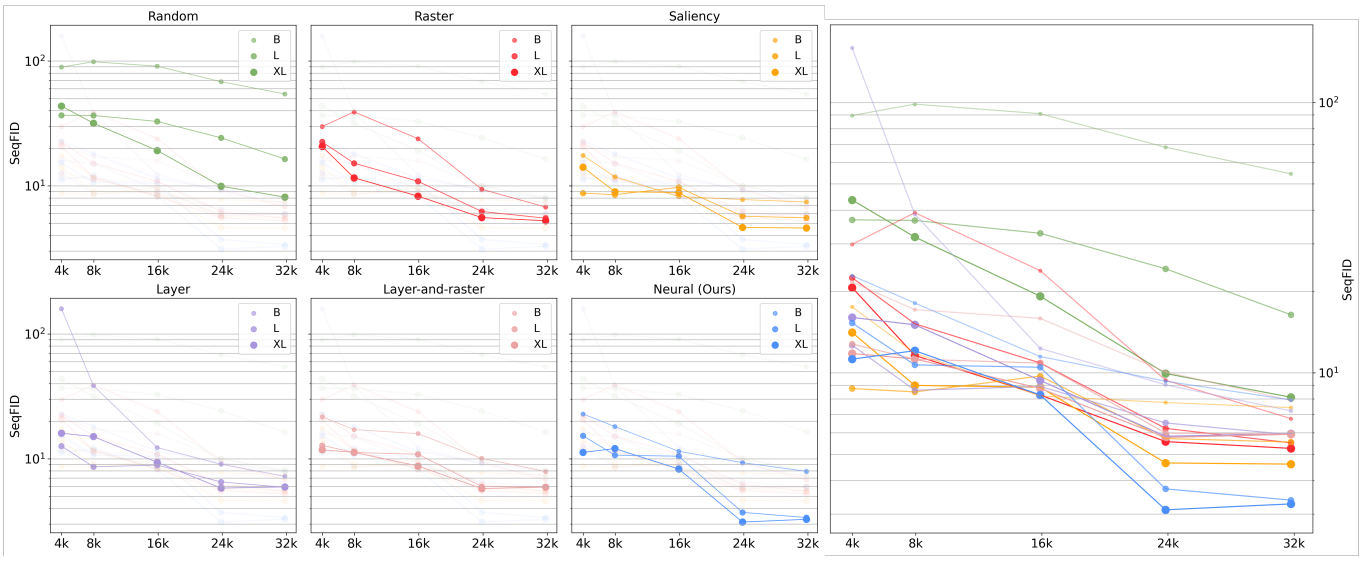


Fig. 12. SeqFID of design generators over training. For each order, three models of different capacities are trained.

6 CONCLUSION AND DISCUSSION

Generative modeling of graphic designs has recently emerged as a promising area of research. Despite great progress achieved by learning distributions of design sequences through powerful generative models, the approach of ordering elements in a design sequence is still under-explored. In this paper, we show element ordering approaches can critically impact generation quality, taking a step in learning a neural order of elements to improve the performance of generative models of graphic designs. With the proposed generative order learning framework, we are able to learn the neural order on graphic design data without need for ordering annotations. We have demonstrated that our neural order can improve training efficiency and lead to dramatically improved design generative models compared with other commonly used ordering approaches across several datasets. We also comprehensively analyze the neural order to reveal its intriguing behaviors, and show that it can enable design generators to scale up excellently, obtaining considerable quality gain by simply making the models deeper and wider. We believe that our neural order can lay a solid foundation and provide inspiration for future work in developing more capable design generation models.

We discuss several interesting avenues to explore in the future. *Alignment with human decisions.* The neural order has been shown to be beneficial to generation quality. It would be interesting to investigate whether it agrees with how human designers decide the order of elements when creating graphic designs. For this end, one possibility is to collect a benchmark of human ordering decisions on graphic designs, and evaluate how well our neural order aligns with the human decisions. *Synthesizing novel images.* In our current method, images on rendered designs are retrieved from a library of pre-existing images in the training dataset. This inevitably limits the visual diversity of rendered designs. To address this, one way is to train an additional diffusion-based image decoder to synthesize novel and diverse images from the embeddings of predicted image tokens (similar to the decoder in DALL-E2 [Ramesh et al. 2022]. *Conditional generation.* We now only evaluate the neural order under the unconditional generation setting. Our design generator can be flexibly extended to handle various conditional inputs (such as text or sketches), e.g., by adding cross-attention layers into the Transformer decoder. We leave evaluating the effectiveness of the neural order for conditional generators as future work.

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning*. 214–223.

Diego Martin Arroyo, Janis Postels, and Federico Tombari. 2021. Variational Transformer Networks for Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13642–13652.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. Structured Denoising Diffusion Models in Discrete State-Spaces. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*. 17981–17993.

Guillaume Ayoub, Simon Sapin, ErikOnBike, aziesemer, Lucie Anglade, Babil G. Sarwar, Miroslav Šedivý, Matthew Scroggs, SilverCardioid, João Ricardo Lhullier Lugão, Neil Freeman, Yotam Gingold, Jon Carr, Mounier Florian, Bart Massey, AdCoder, Werner Thie, Thibaud Gaillard, no, Ricardo Tomasi, Rian McGuire, Raphael Gaschignard, Ralph Bean, Ming Aldrich-Gan, Mark Mollineaux, Malthe Borch, clément plasse, Joseph Kocherhans, Hugo Delahousse, and Guillaume Wenzek. 2024. *Kozea/CairoSVG*. https://github.com/Kozea/CairoSVG

Ayan Kumar Bhunia, Ayan Das, Umar Riaz Muhammad, Yongxin Yang, Timothy M Hospedales, Tao Xiang, Yulia Gryaditskaya, and Yi-Zhe Song. 2020. Pixelor: A Competitive Sketching AI Agent. So you think you can sketch? *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–15.

Andrew Brock. 2018. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv preprint arXiv:1809.11096* (2018).

Neill DF Campbell and Jan Kautz. 2014. Learning a Manifold of Fonts. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–11.

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

Chin-Yi Cheng, Forrest Huang, Gang Li, and Yang Li. 2023. PLay: Parametrically Conditioned Layout Generation using Latent Diffusion. In *Proceedings of the 40th International Conference on Machine Learning*. 5449–5471.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). The MIT Press.

Prafulla Dhariwal and Alex Nichol. 2021. Diffusion Models Beat GANs on Image Synthesis. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*. 8780–8794.

Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. 2023. LayoutGPT: Compositional Visual Planning and Generation with Large Language Models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 18225–18250.

Stephanie Fu, Netanel Y. Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. 2023. DreamSim: Learning New Dimensions of Human Visual Similarity using Synthetic Data. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 50742–50768.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. 2019. Stochastic Optimization of Sorting Networks via Continuous Relaxations. In *International Conference on Learning Representations*.

Shunan Guo, Zhuochen Jin, Fuling Sun, Jingwen Li, Zhaorui Li, Yang Shi, and Nan Cao. 2021. Vinci: An Intelligent Graphic Design System for Generating Advertising Posters. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Article 577, 17 pages.

Kamal Gupta, Justin Lazarow, Alessandro Achille, Larry S Davis, Vijay Mahadevan, and Abhinav Shrivastava. 2021. LayoutTransformer: Layout Generation and Completion with Self-attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1004–1014.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked Autoencoders Are Scalable Vision Learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16000–16009.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 6629–6640.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*. 6840–6851.

Daichi Horita, Naoto Inoue, Kotaro Kikuchi, Kota Yamaguchi, and Kiyoharu Aizawa. 2024. Retrieval-Augmented Layout Transformer for Content-Aware Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 67–76.

Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. 2023a. LayoutDM: Discrete Diffusion Model for Controllable Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10167–10176.

Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. 2023b. Towards Flexible Multi-modal Document Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14287–14296.

Naoto Inoue, Kento Masui, Wataru Shimoda, and Kota Yamaguchi. 2024. OpenCOLE: Towards Reproducible Automatic Graphic Design Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8131–8135.

Peidong Jia, Chenxuan Li, Zeyu Liu, Yichao Shen, Xingru Chen, Yuhui Yuan, Yinglin Zheng, Dong Chen, Ji Li, Xiaodong Xie, et al. 2023. COLE: A Hierarchical Generation Framework for Multi-Layered and Editable Graphic Design. *arXiv preprint arXiv:2311.16974* (2023).

Yue Jiang, Luis A. Leiva, Hamed Rezazadegan Tavakoli, Paul R. B. Houssel, Julia Kylmälä, and Antti Oulasvirta. 2023b. UEyes: Understanding Visual Saliency across User Interface Types. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Article 285, 21 pages.

Zhaoyun Jiang, Jiaqi Guo, Shizhao Sun, Huayu Deng, Zhongkai Wu, Vuksan Mijovic, Zijiang James Yang, Jian-Guang Lou, and Dongmei Zhang. 2023a. LayoutFormer++: Conditional Graphic Layout Generation via Constraint Serialization and Decoding Space Restriction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18403–18412.

Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. 2019. LayoutVAE: Stochastic Scene Layout Generation From a Label Set. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9895–9904.

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

Xiang Kong, Lu Jiang, Huiwen Chang, Han Zhang, Yuan Hao, Haifeng Gong, and Irfan Essa. 2022. BLT: Bidirectional Layout Transformer for Controllable Layout Generation. In *Proceedings of the 17th European Conference on Computer Vision*. 474–490.

Hsin-Ying Lee, Lu Jiang, Irfan Essa, Phuong B. Le, Haifeng Gong, Ming-Hsuan Yang, and Weilong Yang. 2020. Neural Design Network: Graphic Layout Generation with Constraints. In *Proceedings of the 16th European Conference on Computer Vision*. 491–506.

Gang Li, Gilles Baechler, Manuel Tragut, and Yang Li. 2022. Learning to Denoise Raw Mobile UI Layouts for Improving Datasets at Scale. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. Article 67, 13 pages.

Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. 2019. LayoutGAN: Generating Graphic Layouts with Wireframe Discriminators. In *International Conference on Learning Representations*.

Jiawei Lin, Jiaqi Guo, Shizhao Sun, Zijiang James Yang, Jian-Guang Lou, and Dongmei Zhang. 2023a. LayoutPrompter: Awaken the Design Ability of Large Language Models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 43852–43879.

Jinpeng Lin, Min Zhou, Ye Ma, Yifan Gao, Chenxi Fei, Yangjian Chen, Zhang Yu, and Tiezheng Ge. 2023b. AutoPoster: A Highly Automatic and Content-aware Design System for Advertising Poster Generation. In *Proceedings of the 31st ACM International Conference on Multimedia*. 1250–1260.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101* (2017).

James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. 281–297.

Muhammad Ferjad Naeem, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo. 2020. Reliable Fidelity and Diversity Metrics for Generative Models. In *Proceedings of the 37th International Conference on Machine Learning*. 7176–7185.

Peter O'Donovan, Jānis Lībeks, Aseem Agarwala, and Aaron Hertzmann. 2014. Exploratory Font Selection Using Crowdsourced Attributes. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–9.

Akshay Gadi Patil, Omri Ben-Eliezer, Or Perel, and Hadar Averbuch-Elor. 2020. READ: Recursive Autoencoders for Document Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 544–545.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. 2023. SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis. *arXiv preprint arXiv:2307.01952* (2023).

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical Text-Conditional Image Generation with CLIP Latents. *arXiv preprint arXiv:2204.06125* (2022).

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10684–10695.

Jaejung Seol, Seojun Kim, and Jaejun Yoo. 2024. PosterLlama: Bridging Design Ability of Language Model to Content-Aware Layout Generation. In *Proceedings of the 18th European Conference on Computer Vision*. 451–468.

Zecheng Tang, Chenfei Wu, Juntao Li, and Nan Duan. 2023. LayoutNUWA: Revealing the Hidden Layout Expertise of Large Language Models. *arXiv preprint arXiv:2309.09506* (2023).

Praneetha Vaddamanu, Vinay Aggarwal, Bhanu Prakash Reddy Guda, Balaji Vasan Srinivasan, and Niyati Chhaya. 2022. Harmonized Banner Creation from Multimodal Design Assets. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. Article 217, 7 pages.

Kota Yamaguchi. 2021. CanvasVAE: Learning To Generate Vector Graphic Documents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5481–5489.

Xuyong Yang, Tao Mei, Ying-Qing Xu, Yong Rui, and Shipeng Li. 2016. Automatic Generation of Visual-Textual Presentation Layout. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 12, 2 (2016), 1–22.

Junyi Zhang, Jiaqi Guo, Shizhao Sun, Jian-Guang Lou, and Dongmei Zhang. 2023. LayoutDiffusion: Improving Graphic Layout Generation by Discrete Diffusion Probabilistic Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7226–7236.

Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson WH Lau. 2019. Content-aware Generative Modeling of Graphic Design Layouts. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.

Min Zhou, Chenchen Xu, Ye Ma, Tiezheng Ge, Yuning Jiang, and Weiwei Xu. 2022. Composition-aware Graphic Layout GAN for Visual-Textual Presentation Designs. In *Proceedings of the 31th International Joint Conference on Artificial Intelligence*. 4995–5001.