BLT: Bidirectional Layout Transformer for Controllable Layout Generation

Xiang Kong\textsuperscript{2*}, Lu Jiang\textsuperscript{1,2}, Huiwen Chang\textsuperscript{1}, Han Zhang\textsuperscript{1}, Yuan Hao\textsuperscript{1}, Haifeng Gong\textsuperscript{1}, Irfan Essa\textsuperscript{1,3}  
\textsuperscript{1}Google \quad \textsuperscript{2}LTI, Carnegie Mellon University, \textsuperscript{3}Georgia Institute of Technology  
xiangk@cs.cmu.edu \{lujiang, huiwenchang, zhanghan, yuanhao, hfgong, irfanessa\}@google.com

Abstract

Creating visual layouts is an important step in graphic design. Automatic generation of such layouts is important as we seek scale-able and diverse visual designs. Prior works on automatic layout generation focus on unconditional generation, in which the models generate layouts while neglecting user needs for specific problems. To advance conditional layout generation, we introduce BLT, a bidirectional layout transformer. BLT differs from autoregressive decoding as it first generates a draft layout that satisfies the user inputs and then refines the layout iteratively. We verify the proposed model on multiple benchmarks with various fidelity metrics. Our results demonstrate two key advances to the state-of-the-art layout transformer models. First, our model empowers layout transformers to fulfill controllable layout generation. Second, our model slashes the linear inference time in autoregressive decoding into a constant complexity, thereby achieving 4x-10x speedups in generating a layout at inference time.

1. Introduction

Creating visual layouts is important for any design task ranging from generating documents, adverts, posters to furniture layout. Layout dictates the placement and sizing of graphic components, playing a central role in how viewer interacts with the information provided [17].

Graphic layout generation is emerging as a new research direction for generating realistic and diverse layouts to facilitate design tasks. Recent works show promising methods of layout generation for applications such as graphic user interfaces [2], presentation slides [10], magazines [25], scientific publications [1], commercial advertisements [17,22], Computer-Aided Design (CAD) [24], indoor graphics scenes [4], etc.

Recent work explores neural models for layout generation using Generative Adversarial Networks (GANs) [8,18] and Variational Autoencoder (VAEs) [13,16,17,21]. State-of-the-art layout generation models are built on the Transformer architecture [23]. These transformers [1,11] represent a layout as a sequence of objects and an object as a (sub)sequence of attributes (See Fig. 1a). Layout transformers predict the attribute sequentially based on previously generated output (i.e. autoregressive decoding). By virtue of the powerful self-attention, transformers are capable of modeling complex and long-range object relations, and, compared to GANs or VAEs, achieve superior quality and diversity on common layout benchmarks.

However, existing transformers only tackle unconditional layout generation where layout elements are generated from random seeds without considering specific user requirements\textsuperscript{1}. In this process, the users have no controls of what object to be generated or how big the object is.

\textsuperscript{1}Although [11] showed layout completion, we consider it as uncontrollable generation as it provides users with little controls of the object attributes to be generated. This is because of the order of primitives that is an acknowledged limitation detailed in [11].
which is in direct contrast to the real-world scenario where
the designer may already have objects with partially known
attributes and hope to generate the missing attributes. As
shown in Fig. 1a, the user wants to place the icon and but-
ton with known sizes onto the canvas. This is a different
setting to unconditional generation that pays no attention to
the actual object category or the size at hand.

State-of-the-art transformers [1, 11] have difficulties in
conditional generation due to the following two limitations:
• **Immutable dependency chain**: Autoregressive transform-
ers follow a pre-defined generation order of object at-
tributes. For instance, the transformer in Fig. 1b must
generate attributes starting from the category \( c \), then \( x \)
and \( y \), followed by \( w \) and \( h \). The dependency chain is im-
mutable \( i.e. \) it cannot be changed at decoding time. There-
fore, autoregressive transformers fail to perform condi-
tional layout generation when the condition disagrees
with the pre-defined dependency. For example, it is im-
possible to generate the \( y \)-position conditioning on the
known width in Fig. 1b.
• **High latency in decoding**: Autoregressive decoding is not
parallelizable, and the decoding time quickly becomes a
bottleneck for the layout with a large number of objects.
This is an issue for conditional generation because the
model has no control of the number of objects the user
specifies. For example, it can take on average 3 seconds
to decode a layout with 20 objects on a CPU.

In this work, we introduce Bidirectional Layout Trans-
former (or BLT) for controllable layout generation. Differ-
ent from the existing transformer models [1, 11], BLT en-
ables controllable layout generation where every attribute in
the layout can be modified, with high flexibility, based on
the user inputs (cf. Fig. 1a). During training, BLT learns
to predict the masked attributes by attending to attributes
in two directions (cf. Fig. 1b). The bidirectional atten-
tion eliminates the immutable dependency, which allows the
model to fulfill conditional generation. At inference time,
we propose a parallel decoding algorithm, where BLT first
generates a draft layout based on the user inputs, then it-
eratively refines the low-confident attributes in the layout.
Compared to autoregressive decoding, it has constant time
complexity and hence reduces the latency in decoding.

We evaluate the proposed method on six layout datasets
under various metrics to analyze the visual quality. These
datasets cover representative design applications for graphic
user interface [2], magazines [25] and publications [26],
commercial ads [17], natural scenes [19] and home dec-
oration [6]. Experiments demonstrate two key benefits to
the state-of-the-art layout transformer models [1, 11]. First,
our model empowers transformers to fulfill controllable lay-
out generation. Even though our model is not designed for
unconditional layout generation, it achieves quality on-par
with the state-of-the-art. Second, our new method reduces
the linear inference time complexity in [1, 11] to a new con-
stant complexity, thereby achieving 4x-10x speedups in lay-
out generation. To summarize, we make the following con-
tributions:

1. Novel method that empowers transformer model to carry
out conditional and controllable layout generation.
2. Reduce the time complexity in layout transformer to a
better constant complexity.
3. Extensive experiments validate that our method per-
forms favorably against state-of-the-art models in terms
of realism, alignment, and semantic relevance on six lay-
out datasets.

2. Related Work

**Layout Synthesis**: Recently, automatic generation of
high-quality and realistic layouts has fueled increasing in-
terest. Data-driven methods rely on deep generative mod-
els such as GANs [8] and VAEs [16]. For example,
LayoutGAN [18] uses a GANs-based framework to syn-
thesizes semantic and geometric properties for scene ele-
ments. During inference time, LayoutGAN generates lay-
outs from the Gaussian noise. LayoutVAE [13] introduces
two conditional VAEs. The first aims to learn the distri-
bution of category counts which will be used during lay-
out generation. The second produces layouts conditioning
on the number and category of objects generated from the
first VAE or reference data. Due to limited model capac-
ity, it performs worse than the self-attention-based models.
READ [21] uses heuristics to model the relationships and
trains a RNNs-based VAE to generate document layouts.
Neural Design Networks (NDN) [17] is a state-of-the-art
VAEs-based model for conditional layout generation, which
focuses on modeling the asset relations and constraints by
graph convolution. Our work is different from NDN in
modeling the layout and user inputs by the transformer
which, as shown in Table 5, perform more favorably thanks
to the transformer architecture.

Currently, the state-of-the-art for layout generation is
held by the transformer models [23]. In particular, [11] em-
loys the standard autoregressive Transformer decoder with
unidirectional attention. They find out that self-attention is
able to explicitely learn relationships between objects in the
layout, resulting in superior performance compared to prior
works. Furthermore, to increase the diversity of generated
layout, [1] incorporates the standard autoregressive Trans-
former decoder into a VAE framework and [20] employs
multi-choice prediction and winner-takes-all loss. Despite
the superior performance, these models have difficulties in
conditional layout generation. Similar to [17], [14] allows
designers to add constraints through latent optimization.
Bidirectional Transformer: The classic Transformer [23] decoder uses the unidirectional self-attention mechanism to generate the sequence token-by-token from left to right, leaving the right-to-left contexts unexploited. Recently, people start to investigate generation tasks by bidirectional Transformers which allow representations to attend in both directions [3]. Our work is inspired by the success in the generative NLP tasks of language generation [9], text-to-speech generation [5]. Our novelty lies in the proposed new masking strategy and decoding algorithm which, as substantiated by our experiments, are essential for layout generation.

3. Problem Formulation

Following [11], we use 5 attributes to describe an object, i.e., (c, x, y, w, h), in which the first element c ∈ C is the object category such as the logo and button, and the remainder details the bounding box information i.e. the center location (x, y) ∈ R² and the width and height (w, h) ∈ R². Furthermore, float values in bounding box information is discretized using 8-bit uniform quantization. For instance, the x-coordinate after the quantization becomes \( \{x | x \in \mathbb{Z}, 0 \leq x \leq 31 \} \). A layout \( l \) of K assets is hence denoted as a flattened sequence of integer indices:

\[
l = [\langle \text{bos} \rangle, c_1, x_1, y_1, w_1, h_1, c_2, \ldots, h_K, \langle \text{eos} \rangle],
\]

(1)

where \( \langle \text{bos} \rangle \) and \( \langle \text{eos} \rangle \) are special tokens to denote the start and the end of sequence. We use a shared vocabulary and represent each element in \( l \) as an integer index or equivalently as a one-hot vector with the same length. For simplicity, the notation uses five attributes to represent an object but we explore higher-dimensional attributes to model 3D or complex layouts in our experiments.

Issues To train the model, prior work [1, 11] estimates the joint likelihood of observing a layout as:

\[
p(l) = \prod_{i=1}^{||l||} p(l_i | l_{1:i-1}).
\]

(2)

During training, an autoregressive Transformer model is learned to maximize the likelihood using ground-truth attribute as input (i.e., teacher forcing). At inference time, the transformer model predicts the attribute sequentially based on previously generated output (i.e., autoregressive decoding), starting from the begin-of-sequence or \( \langle \text{bos} \rangle \) token until yielding the end-of-sequence token \( \langle \text{eos} \rangle \). The generation must follow a fixed conditional dependency. For example, Eq. (1) defines an immutable generation order \( x \rightarrow y \rightarrow w \rightarrow h \). And in order to generate the height \( h \) for an object, one must know its \( x-y \) coordinates and width \( w \).

There are two issues with autoregressive decoding for conditional generation. First, it is infeasible to process user conditions that differ from the dependency order used in training. For instance, the model using Eq. (1) is not able to generate \( x-y \) coordinates from width and height i.e., \( w \rightarrow h \rightarrow x \rightarrow y \), which corresponds to a practical example of placing an object with given size. This issue is exacerbated by complex layouts that require more attributes to represent an object. Second, the autoregressive inference is not parallelizable, rendering it inefficient for the dense layout with a large number of objects or attributes.

4. Approach

Our goal is to design a transformer model for controllable layout generation. We propose a novel method to learn non-autoregressive transformer. Unlike existing layout transformers [1, 11], the new layout transformer is bidirectional and can generate all attributes simultaneously in parallel, which allows not only for flexible conditional generation but also more efficient inference. In this section, we first discuss the model and training objective; then we detail a new parallel decoding algorithm by iterative refinement.

4.1. Model and Training

The BLT backbone is the multi-layer bidirectional Transformer encoder [23] as shown in Fig. 2. We use the identical architecture as in the existing autoregressive layout transformers [1, 11] but the attention mechanism in our model is bidirectional in the sense that it can utilize richer contexts in two directions to predict the attribute.

Inspired by BERT [3], during training, we randomly select a subset of attributes in the input sequence, replace them with a special “[MASK]” token, and optimize the model to predict the masked attributes. For a layout sequence \( l \), let \( \mathcal{M} \) denote a set of masked positions. Replacing attributes in \( l \) with “[MASK]” at \( \mathcal{M} \) yields the masked sequence \( l^{\mathcal{M}} \).

Given a layout set \( \mathcal{D} \), the training objective is to minimize the negative log-likelihood of the masked attributes:

\[
\mathcal{L}_{\text{mask}} = -\mathbb{E}_{l \in \mathcal{D}} \left[ \sum_{i \in \mathcal{M}} \log p(l_i | l^{\mathcal{M}}) \right],
\]

(3)

While, we can technically predict all attributes, we do not compute the loss for the unmasked attributes as this simple copy-paste task does not constitute a learning task.

The masking strategy greatly affects the quality of masked language model [3]. BERT [3] applies random masking with a fixed ratio where a constant 15% masks are randomly generated for each input. Similarly, we find masking strategy is important for layout generation, but the random masking used in BERT does not work well. We propose to use a hierarchical sampling policy. To do so, we first divide the attributes of an object into several semantic groups, e.g., Fig. 2 shows 3 groups: category, position, and size. In the first step, we randomly select a semantic group.
Next, we dynamically sample the number of masked tokens from a uniform distribution between one and the number of attributes belonging to the chosen group, and then randomly mask that number of tokens in the selected group. As such, it is guaranteed that the model only predicts attributes of the same semantic meaning each time.

4.2. Parallel Decoding by Iterative Refinement

In BLT, all attributes in the layout are generated simultaneously in parallel. Yet, generating layouts in a single pass is quite challenging. To tackle this, we introduce a new parallel decoding algorithm by iterative attribute refinement. The core idea is to generate a layout in a small constant number of steps where parallel decoding is applied at each.

Algorithm 1 Decoding by Iterative Attribute Refinement

Input: Sequence \( l \) with partially-known attributes. Constant \( T \) for the number of iterations.

1. for \( g \) in \([C, S, P]\) do  
   
2. for \( i \leftarrow 1 \) to \( T/3 \) do
3. \( p, l^i = \text{BLT}(l) \)  
4. \( \gamma_i = \frac{T - 3i}{T} \)  
5. \( n_i = \lceil \gamma_i \times |g| \rceil \)  
6. \( M = \text{arg}\top_{k=n_i} (-p) \)  
7. Obtain \( l \) by masking \( l^i \) with respect to \( M \)
8. end for
9. end for
10. return \( l \)

Algorithm 1 presents the proposed decoding algorithm. The procedure is also illustrated in Fig. 2b. The input to the decoding algorithm is a mixture sequence of known and unknown attributes, where the known attributes are given by the user inputs and the model aims at generating the unknown attributes decoded by the [MASK] token.

Like in training, we divide the object attributes into three semantic groups: category \((C)\), size \((S)\) and position \((P)\). For each iteration, only one group of attributes is generated. In Step 3 of Algorithm 1, the model makes parallel predictions for all unknown attributes, where \( p \) denotes the prediction scores. In Step 6, it selects the attributes that belong to the chosen group with the lowest prediction scores. Finally, it masks these low-confident attributes on which the model has doubts. These masked attributes correspond to the difficult cases that will be re-predicted in the next iteration of refinement conditioning on all other ascertained attributes so far. The masking ratio calculated in Step 4 decreases with the number of iteration. This process will repeat \( T \) times until all attributes of all objects are generated (cf. Fig. 2b). Note that the attributes given by the user are considered ground-truth and hence will not be masked during generation. Our algorithm is inspired by the bidirectional NMT model [7]. However, our algorithm is novel in layout generation for processing attribute groups.

Algorithm 1 can also be used for unconditional generation. In this case, the input is a layout sequence of only “[MASK]” tokens, and the same algorithm is used to generate all attributes in the layout. Unlike conditional generation, we need to know the sequence length in advance, i.e. the number of objects to be generated. Here, we can simply use the prior distribution obtained on the training dataset. During decoding, we obtain the number of objects through sampling from this prior distribution.

4.3. Complexity Analysis

This section compares the complexity between the proposed method and the autoregressive layout transformers [1, 11]. We focus on the time complexity when full parallelization is assumed. To generate a sequence of length \( N \), the autoregressive transformer needs \( N \) steps with the cost of \( \delta \) for generating each attribute. So the total cost amounts to \( O(N\delta) = O(N) \). By contrast, our model generates the layout in a constant number of \( T \) steps \((T \ll N)\), where the total cost equals \( O(T\delta) = O(1) \). Besides, both models have the same space complexity of \( O(N) \). The above theoretical analysis shows the constant time complexity of BLT and the finding is consistent with our empirical runtime comparison in Section 5.4.
In 3D-FRONT are represented by the 3D bounding boxes. Different from previous datasets, objects e.g., etc. contain 80 things and 91 stuff categories, after removing tagged as “iscrowd”. Its objects come from 5 categories: text, title, figure, list, backgrounds).

5. Experimental Results

This section verifies the proposed method on six layout benchmarks under four metrics to examine the quality. We compare to the state-of-the-art models on conditional and unconditional layout generation tasks. The results show our model performs favorably against the strong baselines and achieves 4x-10x speedups in layout generation.

5.1. Setsups

Datasets We employ six datasets that cover representative graphic design applications. RICO [2] is a dataset of user interface designs for mobile applications. It contains 91K entries with 27 object categories (button, toolbar, list item, etc.). PubLayNet [26] contains 330K examples of machine annotated scientific documents crawled from the Internet. Its objects come from 5 categories: text, title, figure, list, and table. Magazine [25] contains 4K images of magazine pages and 6 categories (texts, images, headlines, over-image texts, over-image headlines, backgrounds). Image Ads [17] is the commercial ads dataset with layout annotation detailed in [17]. COCO [19] contains ~100K images of natural scenes. We follow [1] to use the Stuff variant, which contains 80 things and 91 stuff categories, after removing small bounding boxes (≤ 2% image area), and instances tagged as “iscrowd”. 3D-FRONT [6] is a repository of professionally designed indoor layouts. It contains around 7K room layouts with objects belonging to 37 categories, e.g., the table and bed. Different from previous datasets, objects in 3D-FRONT are represented by the 3D bounding boxes.

Evaluation metrics Prior works use multiple metrics to assess the quality of generated layouts from various perspectives. We employ three common metrics in the literature and introduce a new metric for conditional generation. Specifically, IOU measures the intersection over union between the generated bounding boxes. We compute the IOU in the pixel space. Overlap [18] measures the total overlapping area between any pair of bounding boxes inside the layout. Alignment [17] computes an alignment loss with the intuition that objects in graphic design are often aligned either by center or edge (e.g., left- or right-aligned). For all IOU, Overlap, Alignment, the lower the better.

The above metrics do not consider user inputs, and we need a metric for conditional generation. We introduce Similarity which compares the generated layout with the real layout under the same input condition. We use DocSim [25] to calculate the similarity between two layouts.

Generation settings We examine three practical layout generation scenarios (2 conditional and 1 unconditional).

- Conditional on Category: only object categories are given by users. The model needs to predict the size and position for each object.
- Conditional on Category + Size: users specify the object category and size for each object, and the model needs to predict the position, i.e., placing the objects on the canvas.
- Unconditional Generation: no information is provided by users. Users have little control of the generation process. Prior works mainly focus on this setting.

In unconditional generation, the model generates 1K samples from the random seed. The test split of each dataset is used for conditional generation.

Implementation details The model is trained for five trials with random initialization and the averaged metrics with standard deviations are reported. All models including ours have the same configuration, i.e., 4 layers, 8 attention heads, 512 embedding dimensions and 2,048 hidden dimensions. Adam optimizer [15] with $\beta_1 = 0.9$ and $\beta_2 = 0.98$ is used. Models are trained on 2×2 TPU devices with batch size 64. For conditional generation, we randomly shuffle objects in the layout. For unconditional generation, to improve diversity, we use the nucleus sampling [12] with $p = 0.9$ for the

| Model | IOU↓ | Overlap↓ | Alignment↓ | Sim.↑ | Sim.↑ |
|-------|------|---------|-----------|-------|-------|
| Trans. | 0.24±0.2% | 0.33±0.4% | 0.30±0.4% | 0.20±0.1% | - |
| VTN | 0.22±0.1% | 0.30±0.3% | 0.32±0.9% | 0.20±0.1% | - |
| Ours | 0.22±0.4% | 0.23±0.3% | 0.20±1.1% | 0.21±0.2% | 0.30 |

Table 1. Category (+ Size) conditional layout generation performance on various benchmarks.

| Ads | Conditioned on Category | + Size |
|-----|-------------------------|-------|
| Model | IOU↓ | Overlap↓ | Alignment↓ | Sim.↑ | Sim.↑ |
| Trans. | 0.60±0.4% | 1.86±0.6% | 0.34±0.2% | 0.20±0.2% | - |
| VTN | 0.63±0.4% | 1.79±0.4% | 0.32±0.3% | 0.22±0.1% | - |
| Ours | 0.35±0.5% | 1.93±0.5% | 0.16±0.5% | 0.24±0.1% | 0.44 |

Table 2. Conditional generation performance on Image Ads.

| Magazine | Conditioned on Category | + Size |
|----------|-------------------------|-------|
| Model | IOU↓ | Overlap↓ | Alignment↓ | Sim.↑ | Sim.↑ |
| Trans. | 0.20±0.6% | 0.22±0.6% | 0.48±1.1% | 0.15±0.3% | - |
| VTN | 0.18±0.6% | 0.15±0.2% | 0.47±0.4% | 0.15±0.9% | - |
| Ours | 0.18±0.6% | 0.12±1.0% | 0.44±1.9% | 0.16±0.4% | 0.27 |

Table 3. Conditional generation performance on 3D-FRONT.
Although our model is not designed for such purpose, we compare it to recent models [1, 11, 13] on unconditional layout generation. From Table 4, our model outperforms LayoutVAE [13] and achieves comparable performance with two autoregressive transformers (Trans. [11] and VTN [1]). Furthermore, we compare our method with a state-of-the-art VAE model (called NDN) [17] by their proposed metric “Alignment”. We demonstrate better results when the model is given no constraints (NDN-none) and all constraints (NDN-all).

5.3. Qualitative Result

We show some generated layouts, along with the rendered examples for visualization, in Fig. 4. The setting is conditional generation on category and size for three design applications including the mobile UI interface, scientific paper, and magazine. We observe that our method yields realistic layouts which facilitates generating high-quality output by rendering, suggesting that our model is capable of capturing position relationships between objects.

Next we explore the home design task on the 3D-Front dataset [6]. The goal is to place the furniture with the user-given category and length, height, and width. Examples are shown in Fig. 5. Unlike the previous tasks, here the model needs to predict the position for the 3D bounding box. The low similarity score on this dataset indicate that housing design layout is still a challenging task and we will explore this in future work.

Where to pay attention? To further understand what relationships between attributes BLT is able to learn, we visualize the patterns in how our model’s attention heads behave. We choose a simple layout with two objects and mask their positions (x, y). The model needs to predict these masked attributes from other known attributes. Examples of heads exhibiting these patterns are shown in Fig. 3. We use ⟨layer⟩-⟨head number⟩ to denote a particular attention head.

5.4. Ablation Study

Attribute order in the decoding algorithm In Algorithm 1, we prespecify an order of attribute groups, which is
Figure 3. Examples of attention heads exhibiting the patterns for masked tokens. The darkness of a line indicates the strength of the attention weight (some attention weights are so low they are invisible). We use ⟨layer⟩-⟨head number⟩ to denote a particular attention head.

Figure 4. Conditional layout generation for scientific papers, user interface, and magazine. The user inputs are the object category and their size (width, height). We also present the rendered examples constructed based on the generated layouts.

Category (C) → Size (S) → Position (P). Here, more orders are explored including the “No Order” in BERT [3] where attributes are individually selected without using group. The result is shown in Table 6. “No Order” performs clearly worse than the other orders, demonstrating the necessity of our algorithm. Moreover, among pre-defined orders, it seems better to first generate the category and afterward determine either location or size.

Decoding speed w.r.t. the sequence length We compare the inference speed of our model and the autoregressive transformer models [1, 11]. Specifically, all models generate 1,000 layouts with batch size 1 on a single GPU and the average decoding time in millisecond is reported. The result is shown in Fig.6, where the x-axis denotes the number of objects in the layout. It shows that autoregressive decoding time grows with #objects. On the contrary, the decoding speed of the proposed model appears not affecting by #objects. The speed advantage becomes more evident when producing dense layouts. For example, our fastest model obtains a 4x speed-up when generating around 10 objects and a 10x speed-up for 20 objects.

Iterative Refinement Process To understand the process of our iterative refinement algorithm, we show a sample of generated layouts at different number of iterations in Fig.8. At the first iteration, there are severe overlaps between objects, showing the difficulty to yield high-quality layouts.
Figure 5. 3D-FRONT sample layouts.

Figure 6. Decoding speed versus number of generated assets. ‘Autoregressive’ denote the autogressive Transformer-based model. ‘Iter-*’ shows the proposed model with various #iterations.

with just one pass. However, after iteratively refining low-confident attributes, the layouts become more realistic.

Quantitatively, IOU and Overlap metrics, where lower the better, are plotted in Fig. 7 along with #iterations. With more iterations, the quality metrics are getting improved and stable. This result is consistent with our qualitative observation above on the Magazine and PubLayNet datasets.

Figure 7. IOU and overlap scores with different number of iterations on Magazine and PubLayNet.

6. Conclusion and Future Work

We present BLT, a bidirectional layout transformer capable of generating layout attributes in parallel. We design an iterative refinement algorithm to generate high-quality layouts in a few rounds. Compared to the autoregressive Transformer, our model is suitable for flexible conditional layout generation where every attribute is controlled by the users. Experimental results on several benchmarks demonstrate the effectiveness and flexibility of BLT.

| Order     | IOU  | Overlap | Alignment |
|----------|------|---------|-----------|
| C→S→P   | 0.127| 0.102   | 0.342     |
| C→P→S   | 0.129| 0.107   | 0.344     |
| S→C→P   | 0.147| 0.109   | 0.351     |
| S→P→C   | 0.162| 0.121   | 0.357     |
| No Order | 0.208| 0.193   | 0.374     |

Table 6. Layout generation results with different iteration group orders on the RICO dataset. C, S and P denote category, size and position attribute groups, respectively.

Negative Societal Impact and Limitation BLT may be applied to questionable and potentially harmful applications. A limitation of our work is content-agnostic conditional generation. We leave this out to have a fair comparison to our baselines which do not use visual information either. In the future, we will explore using rich visual information.

References

[1] Diego Martin Arroyo, Janis Postels, and Federico Tombari. Variational transformer networks for layout generation. In CVPR, 2021.
[2] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hirschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In UIST, 2017. 1, 2, 5

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In NAACL, 2019. 3, 7

[4] Xinhan Di and Pengqian Yu. Multi-agent reinforcement learning of 3d furniture layout simulation in indoor graphics scenes. arXiv preprint arXiv:2102.09137, 2021. 1

[5] Jeff Donahue, Sander Dieleman, Mikolaj Binkowski, Erich Elsen, and Karen Simonyan. End-to-end adversarial text-to-speech. In ICLR, 2020. 3

[6] Huan Fu, Bowen Cai, Lin Gao, Lingxiao Zhang, Cao Li, Qixun Qian, Yongyi Chen, Yu Zheng, Yi Liu, Peng Liu, Lin Ma, Le Weng, Xiaohang Hu, Xin Ma, Qian Qian, Rongfei Jia, Binqiang Zhao, and Hao Zhang. 3dfront: 3d furnished rooms with layouts and semantics. arXiv preprint arXiv:2011.09127, 2020. 1, 5, 6

[7] Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In EMNLP-IJCNLP, 2019. 4

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. NeurIPS, 27, 2014. 1, 2

[9] Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. Non-autoregressive neural machine translation. In ICLR, 2018. 3

[10] Mengxi Guo, Dangqing Huang, and Xiaodong Xie. The layout generation algorithm of graphic design based on transformer-cvae. arXiv preprint arXiv:2110.06794, 2021. 1

[11] Kamal Gupta, Justin Lazarow, Alessandro Achille, Larry S Davis, Vijay Mahadevan, and Abhinav Shrivastava. Layouttransformer: Layout generation and completion with self-attention. In ICCV, 2021. 1, 2, 3, 4, 6, 7

[12] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In ICLR, 2019. 5

[13] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. Layoutvae: Stochastic scene layout generation from a label set. In CVPR, 2019. 1, 2, 6

[14] Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. Constrained graphic layout generation via latent optimization. In MM, 2021. 2

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014. 5

[16] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013. 1, 2

[17] Hsin-Ying Lee, Lu Jiang, Irfan Essa, Phuong B Le, Haifeng Gong, Ming-Hsuan Yang, and Weilong Yang. Neural design network: Graphic layout generation with constraints. In ECCV, 2020. 1, 2, 5, 6

[18] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. In ICLR, 2018. 1, 2, 5

[19] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In ECCV, 2014. 2, 5

[20] David D Nguyen, Surya Nepal, and Salil S Kanhere. Diverse multimedia layout generation with multi choice learning. In MM, 2021. 2

[21] Akshay Gadi Patil, Omri Ben-Eliezer, Or Perel, and Hadar Averbuch-Elor. Read: Recursive autoencoders for document layout generation. In CVPR Workshops, 2020. 1, 2

[22] Chunyao Qian, Shizhao Sun, Weiwei Cui, Jian-Guang Lou, Haidong Zhang, and Dongmei Zhang. Retrieve-then-Adapt: Example-based automatic generation for proportion-related infographics. IEEE Transactions on Visualization and Computer Graphics, 27(2):443–452, 2020. 1

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In NeurIPS, 2017. 1, 2, 3

[24] Karl DD Willis, Pradeep Kumar Jayaraman, Joseph G Lambourne, Hang Chu, and Yewen Pu. Engineering sketch generation for computer-aided design. In CVPR, 2021. 1

[25] Xiru Zheng, Xiaotian Qiao, Ying Cao, and Rynson WH Lau. Content-aware generative modeling of graphic design layouts. ACM Transactions on Graphics (TOG), 38(4):1–15, 2019. 1, 2, 5

[26] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. Publayout: largest dataset ever for document layout analysis. In IEEE ICDAR, 2019. 2, 5
Appendix: BLT: Bidirectional Layout Transformer for Controllable Layout Generation

A. Training Details

To find out the optimal hyperparameters for each task, we use a simple grid search for the following ranges of possible values, learning rate in \( \{1 \times 10^{-3}, 3 \times 10^{-3}, 5 \times 10^{-3}\} \), dropout and attention dropout in \( \{0.0, 0.3\} \). The data preprocessing procedure discussed in [1] is used. In particular, for RICO dataset, [1] removes layouts with more than 100 objects, ignoring 0.03% of the data. Since our model is trained on TPU with stricter memory constraints, we omit layouts with more than 50 objects, in total removing around 0.5% of the data.

B. Evaluation Metrics

In [1], the author calculate the IOU scores between all pairs of overlapped objects and average them. In our work, we compute the IOU score by projecting the layouts into the discrete space \((32 \times 32)\), which is the overlapped area divided by the union area of all objects. We show the difference via a toy example in Fig. 9. The areas of objects \(A\), \(B\) and \(C\) are 5, 1, 1, the overlapped area of \(B\) and \(C\) are 0.5. Based on their method, since they just care about overlapped objects, only the IOU of objects \(B\) and \(C\) are computed which is \(\frac{0.5}{1.5} = \frac{1}{3}\). On the contrary, in our method, the overlapped area of \(B\) and \(C\) will be divided the union area of all objects, hence, the IOU of this layout is \(\frac{0.5}{6.5} = \frac{1}{13}\) which is more reasonable than their result.

Figure 9. An toy layout sample for the IOU computation. The metrics used yields more reasonable IOU \(\frac{0.5}{6.5} = \frac{1}{13}\) than the IOU \(\frac{0.5}{1.5} = \frac{1}{3}\) used in [1]

C. Diverse Conditional Generation

In our main experiment, we use greedy search to find out the most likely candidate for each attribute at each iteration. Here, we generate layouts through sampling the top-k \((k = 10)\) from the likelihood distribution for category and size conditional generation. This leads to generation diverse layouts. Some examples are shown in Fig. 11.

D. Additional Visualization

D.1. More Attention Head Patterns

Patterns for other heads at different layers are listed in Fig. 10. We could find that for masked \(x\) position (head 1-1 and head 2-6, etc.), their heads will attend to width information of various objects for accurate prediction. And similar findings could be found for other heads.

D.2. Qualitative Results

We show more samples in Fig. 12 from conditional generation on category and size for four design applications including the mobile UI interface, scientific paper, magazine and natural scenes.

D.3. Iterative Refinement Process

We list more samples for iterative refinement process in Fig. 13. Severe overlaps between objects will be mitigated with more iterations. The quantitative quality metrics for the layout generated at each iteration is compared in Figure 7 of the main paper.

D.4. Failure Cases

Some undesired conditional generation results are shown in Fig 14. Similar to other layout generation models, there are some overlaps between objects in some generation results. Furthermore, some generated samples are largely different from the real layouts with low visual quality. For example, in the second sample on the Magazine, the alignment of the generated sample is worse than its corresponding real layout. We will explore these directions in the future work.
Figure 10. More examples of attention heads exhibiting the patterns for masked tokens. The darkness of a line indicates the strength of the attention weight (some attention weights are so low they are invisible). We use ⟨layer⟩-⟨head number⟩ to denote a particular attention head.

Figure 11. Diverse conditional generation via top-k sampling method.
Figure 12. Conditional layout generation for scientific papers, user interface, and magazine. The user inputs are the object category and their size (width, height). We compare the generated layout and the real layout with the same input in the dataset.
Figure 13. More layouts refinement process. Layouts generated at different iterations ($t$) are shown on three datasets.
Figure 14. Failure cases for layout generation using the propose method. We compare the generated layout and the real layout with the same input in the dataset. See Section D.4 for more discussion.