UniLayout: Taming Unified Sequence-to-Sequence Transformers for Graphic Layout Generation

Zhaoyun Jiang\textsuperscript{1}, Huayu Deng\textsuperscript{2}, Zhongkai Wu\textsuperscript{3}, Jiaqi Guo\textsuperscript{1}, Shizhao Sun\textsuperscript{1}, Vuksan Mijovic\textsuperscript{5}, Zijiang Yang\textsuperscript{1}, Jian-Guang Lou\textsuperscript{4}, Dongmei Zhang\textsuperscript{1}
\textsuperscript{1}Xi’an Jiaotong University, \textsuperscript{2}Shanghai Jiaotong University, \textsuperscript{3}Beihang University, \textsuperscript{4}Microsoft Research Asia, \textsuperscript{5}Microsoft

Abstract

To satisfy various user needs, different subtasks of graphic layout generation have been explored intensively in recent years. Existing studies usually propose task-specific methods with diverse input-output formats, dedicated model architectures, and different learning methods. However, those specialized approaches make the adaption to unseen subtasks difficult, hinder the knowledge sharing between different subtasks, and are contrary to the trend of devising general-purpose models. In this work, we propose UniLayout, which handles different subtasks for graphic layout generation in a unified manner. First, we uniformly represent diverse inputs and outputs of subtasks as the sequences of tokens. Then, based on the unified sequence format, we naturally leverage an identical encoder-decoder architecture with Transformers for different subtasks. Moreover, based on the above two kinds of unification, we further develop a single model that supports all subtasks concurrently. Experiments on two public datasets demonstrate that while simple, UniLayout significantly outperforms the previous task-specific methods.

1 Introduction

Graphic designs, which facilitate information communication through effective visual expression, appear everywhere in our daily life. During the creation of graphic design, the layout, i.e., positions and sizes of elements, plays a critical role. Figure 1(a) shows examples for graphic designs and their corresponding layouts. To assist layout design, technologies about automatic layout generation have begun to take shape recently. Different subtasks of layout generation are proposed to tackle diverse users’ needs, including unconstrained generation [18, 3, 9, 34, 10], generation conditioned on element types [17, 13, 16], generation conditioned on element types and sizes [19, 16], generation conditioned on element relationships [17, 13], refinement [26] and completion [9] (see Figure 1(b)).

Currently, most layout generation approaches are subtask-specialized (see Table 1). They adopt dramatically different input-output formats of models. For example, on generation conditioned on element relationships, layouts are formulated as graphs [17], while in completion, they are often formulated as sequences. In addition, existing approaches usually employ customized model architectures and learning methods for different subtasks. For example, on generation conditioned on element types, GNN [30] and VAE [15] are leveraged, while on completion, Transformer decoder [32] and maximum likelihood estimation are used. These subtask-specialized approaches, however, greatly hinder the rapid development of solutions for new subtasks and effective knowledge sharing between subtasks. First, the techniques in specialized approaches may be prone to overfitting the subtasks or datasets, thereby failing to generalize to new subtasks. Second, although different subtasks face their unique challenges, they all share the same fundamental challenge, i.e., how to generate aesthetic...
layouts. Regrettably, existing approaches set up tough barriers between subtasks, and consequently, prevent the effective sharing of knowledge about layout generation. On the other side, there is an emerging trend of devising general-purpose models \cite{12, 35, 38, 11, 37, 33} to reduce task-specific techniques and tackle diverse subtasks in a unified manner.

This makes us beg the question: can different layout generation subtasks be unified? We attempt to answer it from three perspectives: 1) how to design a unified input-output format for different subtasks, 2) whether there exists one model architecture that handles different subtasks effectively, and 3) whether it is possible to train a single model that supports all the subtasks simultaneously. To this end, we propose a simple yet effective approach, called UniLayout. First, all subtasks are formulated as sequence-to-sequence transformations (see Figure 2(a)). The intuition behind is two-fold. On the one hand, formulating layouts as sequences conforms to their characteristics. A layout is made up of a set of elements, each of which can be described by five attributes, including its element type, left and top coordinates, as well as its width and height. Thus, it is natural to represent an element by five tokens and a layout by the concatenation of elements. On the other hand, it enables layout generation for various kinds of users’ needs. While users’ needs vary in subtasks, they are all about the five attributes of elements. Thus, we also represent users’ needs as sequences of tokens. Second, considering the sequence-to-sequence formulation, we uniformly leverage a simple Transformer \cite{32} encoder-decoder for different subtasks. Third, the unification of input-output format and model architecture makes it possible to train a single model for all subtasks. To achieve it, we add a subtask prefix to the input sequence and then put the data for all the subtasks together for training.

We conduct experiments on two public datasets, i.e., RICO \cite{20} and PubLayNet \cite{40}, to compare UniLayout with existing approaches on six layout generation subtasks. First, we train a model for each subtask using the proposed input-output format and model architecture. Experiments show that our approach significantly outperforms those specialized approaches on all the subtasks. Furthermore, by putting all the subtasks together to train a single model, our approach still achieves better performance than existing approaches on most subtasks.

2 Related Work

**Graphic Layout Generation.** Graphic layout generation is an emerging research topic. It aims at generating aesthetic layouts to tackle users’ needs. As users’ needs are diverse in real-world applications, various subtasks of layout generation are proposed. To achieve good performance on each subtask, many task-specific approaches are studied. As summarized in Table 1, how those approaches are specifically designed can be viewed from at least three dimensions. The first one is the input-output format of the model, which includes the graph \cite{17}, tree \cite{23}, set \cite{18, 19, 26, 13, 16}, and sequence \cite{3, 10, 34, 9, 21}. The approaches using graphs and trees, predefine a fixed set of pairwise relationships between elements (e.g., on the top of or smaller than) and then jointly model them via graphs or trees. For the approaches using sets, the generation order of elements does not matter, while for those using sequences, the generation of latter elements should depend on preceding elements. The second dimension is the model architecture. Recursive network (RvNN) and graphic neural network (GNN) \cite{17, 23} are leveraged when input-output formats are trees and graphs, respectively. Transformer encoder \cite{16, 26, 13} is used to handle the set format, while Transformer encoder-
Table 1: Summarization for techniques considered by existing studies.

| Dimensions          | Techniques Considered by Existing Studies |
|---------------------|------------------------------------------|
| Input-output Format | Graph [17], Tree [23], Set [18, 19, 26, 13, 16], Sequence [3, 10, 34, 9, 21] |
| Model Architecture  | Recursive Neural Network [23], GNN [17], Transformer Encoder [16, 26, 13], Transformer Decoder [9], Transformer Encoder-Decoder [3, 10, 34] |
| Learning Method     | GAN [18, 19, 39, 13], VAE [17, 23, 3, 10, 34], Masking Strategy [16], Maximum Likelihood Estimation [9, 26], Constrained Optimization [13] |

decoder [3, 10, 34] and Transformer decoder [9] are considered when the input-output format is the sequence. The third dimension is the learning method. GAN [18, 19, 39, 13], VAE [17, 23, 3, 10, 34], and maximum likelihood estimation (MLE) [9, 26] are the ones used most frequently. Besides, inspired by the recent success of BERT [7], masking strategy is also explored [16]. Moreover, some subtasks are formulated as constrained optimization problems and solved by an augmented Lagrangian method [13].

The above summarization reveals that existing work often designed completely different approaches to deal with different subtasks. For example, for unconstrained generation, VTN [3] uses the sequence format, Transformer encoder-decoder, and VAE; for generation conditioned on types, BLT [16] leverages the set format, Transformer encoder, and masking strategy. This motivates us to rethink whether it is necessary to develop those specialized techniques for layout generation subtasks.

Unified Frameworks. There is an emerging trend of developing general-purpose models to tackle tasks in a unified manner. In natural language processing, a unified text-to-text framework has been widely adopted to address various subtasks of named entity recognition [35], entity linking [4], question answering [12], and structured knowledge grounding [33]. In the vision community, different subtasks of image generation [31, 28, 8, 27] and object detection [3] has also been resolved by a unified framework. Moreover, unifying vision and language tasks has attracted great attention recently [6, 36]. Inspired by the success of those studies, we propose the first approach to unifying different layout generation subtasks.

3 Approach

3.1 Input and Output Format

As the first step towards a unified approach, we process the diverse inputs and outputs of subtasks into a uniform format. We start from the output format since different subtasks have an identical output, i.e., the graphic layout. In essence, a layout is composed of elements, where each element can be represented by five attributes, including its type, left and top coordinate, as well as width and height. Thus, we naturally represent an element by five tokens, and then represent a layout as a sequence by concatenating the representation of all its elements in a certain order. Other formats of layouts, such as graph [17], tree [23], and set [16, 26, 13], have also been explored by previous work, but they are not flexible enough compared to the sequence. For example, when the number of elements is not given, it would be challenging to generate a layout in either graph or set formats.

Regarding the input format, we also choose the sequence given the following considerations. First, while different, the users’ needs are all about the five attributes of elements. Therefore, it is natural to think about representing them as sequences. Second, the sequence format is expressive and inclusive. For example, we can easily represent any meaning or serialize any structured information in the user’s need as a sequence of tokens. In the following, we introduce the details of the input and output sequences of different subtasks and discuss the order in which elements are concatenated.

Output Format. We denote the total number of elements in a graphic layout as $N$. Moreover, we denote the type, left coordinate, top coordinate, width and height of the $i$-th element as $c_i$, $x_i$, $y_i$, $w_i$ and $h_i$. The continuous attributes $x_i$, $y_i$, $w_i$, and $h_i$ of elements are quantized into $[0, 127]$, which is proven to be helpful for graphic layouts [9, 3, 26]. Thus, an element can be represented by five tokens. Then, we represent a layout by concatenating all the elements’ tokens, i.e.,
We formulate a relationship between two elements as

\[ S = \langle \text{sos} \rangle c_1 x_1 y_1 w_1 h_1 | c_2 x_2 y_2 w_2 h_2 | \ldots | c_N x_N y_N w_N h_N \langle \text{eos} \rangle \].

Here, \( \langle \text{sos} \rangle \) and \( \langle \text{eos} \rangle \) are special tokens indicating the start and end of a sequence, and \( | \) is the separator between two elements.

Subtasks and Their Input Formats. There are six typical subtasks of layout generation. Figure 1(b) visualizes these subtasks and Figure 2(a) provides examples for their input and output formats.

Unconstrained generation (UGen) aims at generating aesthetic graphic layouts without considering any user requirements. Thus, we formulate the input as an empty sequence with necessary special tokens, i.e., \( S_{\text{UGen}} = \{ \langle \text{sos} \rangle \langle \text{eos} \rangle \} \).

Generation conditioned on types (Gen-T) is to generate layouts conditioned on the element types specified by users. We formulate the input as the concatenation of element types, i.e., \( S_{\text{Gen-T}} = \{ \langle \text{sos} \rangle c_1 c_2 | \ldots | c_N \langle \text{eos} \rangle \} \).

Generation conditioned on types and sizes (Gen-TS) is to generate layouts when users not only specify the element types but also the sizes of these elements. We formulate the input as a sequence containing the type, the width and the height for each element, i.e., \( S_{\text{Gen-TS}} = \{ \langle \text{sos} \rangle c_1 w_1 h_1 | c_2 w_2 h_2 | \ldots | c_N w_N h_N \langle \text{eos} \rangle \} \).

Generation conditioned on relationships (Gen-R) generates layouts conditioned on element relationships specified by users. For example, a user could expect to put a large image at the top left and a small text box at the bottom right. Typically, there are five kinds of position relationships (i.e., above, bottom, left, right, and overlap) and three kinds of size relationships (i.e., smaller, larger and equal) [13]. We formulate a relationship between two elements as \( \{ c_{k_{2m-1}}, k_{2m-1} r_{k_{2m-1}} k_{2m}, c_{k_{2m}}, k_{2m} \} \), where \( c_{k_{2m-1}} \) and \( c_{k_{2m}} \) are the element types, \( k_{2m-1} \) and \( k_{2m} \) are the indexes for the elements, and \( r_{k_{2m-1}}, k_{2m} \) is an extra token denoting one kind of aforementioned relationships. Based on it, we formulate the input as the concatenation of element types and relationships, i.e., \( S_{\text{Gen-R}} = \{ \langle \text{sos} \rangle c_1 | \ldots | c_N | k_{k_1} r_{k_1} k_2 | k_3 | k_2 | \ldots | k_{M-1} r_{k_{M-1}} k_M | \langle \text{eos} \rangle \} \), where \( M \) is the number of relationships and \( | \) is a special token to separate element types and relationships.

Refinement applies local changes to the elements that need improvements while maintaining the original layout design. As the input is a layout draft from the user, we formulate it as \( S_{\text{Refinement}} = \{ \langle \text{sos} \rangle c_1 x_1 y_1 w_1 h_1 | c_2 x_2 y_2 w_2 h_2 | \ldots | c_N x_N y_N w_N h_N \langle \text{eos} \rangle \} \).

Completion aims at generating other elements when the user has specified a set of elements. Thus, we formulate the input as \( S_{\text{Completion}} = \{ \langle \text{sos} \rangle c_1 x_1 y_1 w_1 h_1 | c_2 x_2 y_2 w_2 h_2 | \ldots | c_P x_P y_P w_P h_P \langle \text{eos} \rangle \} \), where \( P \) is the number of known elements.

Sequence Order. When we construct an input or output sequence, the elements should be concatenated in a certain order. In this work, we consider two design choices for the sequence order. With the alphabetic order, the elements are sorted by the alphabetic order of element types. With the position order, the elements are sorted by their left coordinate \( x \) and the top coordinate \( y \). For input sequences, as there is no information about element positions in most subtasks, we can only choose alphabetic order. For output sequences, both the alphabetic order and the position order are applicable.

3.2 Model Architecture, Training and Inference

Architecture. Considering the sequence-to-sequence formulation of subtasks, we adopt the encoder-decoder architecture with Transformers [32] to tackle subtasks uniformly (see Figure 2(b)). Our bidirectional encoder takes users’ needs as input and outputs their contextualized representations.
Then, the autoregressive decoder iteratively attends to previously generated tokens (which constitute a part of the target layout) as well as the contextualized representations (which indicate users’ needs), and predicts the probabilities of future tokens.

Besides Transformer encoder-decoder, Transformer decoder only is also a popular choice for sequence-to-sequence problems [2][24]. The intuition behind our choice of Transformer encoder-decoder is that the bidirectional Transformer encoder is superior in modeling complex sequences and thus can better understand diverse and complicated user needs in different subtasks. For example, on Gen-R, the model is required to have a deep understanding of the specified pairwise relationships between elements. We also explore and compare these two architecture choices by experiments.

**Single-task Training.** We first consider how to train a model for each subtask. For ease of reference, we denote it as UniLayout-S. It is trained to minimize the negative log-likelihood of layout tokens conditioned on the preceding layout tokens and the sequence of users’ needs, i.e.,

\[
\text{minimize } \sum_{t=1}^{|L|} - \log P_\theta(L_t|L_{<t}, S),
\]

where \(S\) is the input sequence representing users’ needs, \(L\) is the output layout sequence, \(\theta\) refers to the model parameters and \(|L|\) denotes the length of the layout sequence. We also apply the teacher-forcing strategy, feeding the ground-truth of preceding layout tokens to the decoder.

**Multi-task Training.** The unification of input-output format and model architecture opens up a chance for training a single model to serve all subtasks simultaneously, denoted as UniLayout-M. This is of great benefit. First, it enables deeper knowledge sharing between subtasks, which could boost the performance of some subtasks even further. Second, it saves a lot of deployment efforts in practice, since there is a single set of model weights for serving multiple subtasks. To achieve the goal, we explore a simple multi-task training method, inspired by its success in various NLP problems [25][2][29]. Specifically, we prepend a task indicator token to each input sequence, combine the training data of all subtasks with temperature mixing, and leverage Eqn. (1) to train UniLayout-M.

**Inference with Constrained Decoding Strategy.** Graphic layout generation requires at least two types of constraints. The format constraints state that a valid layout sequence should conform to the format defined in Section 3.1 otherwise, it cannot be parsed to a layout. The task-specific constraints refer to the constraints imposed by subtasks. Take the subtask of Gen-TS as an example. The number of elements as well as the type and size of each element are specified in the input. A valid layout for this subtask is required to meet all these specifications.

To make the generated layouts less violate these constraints, we introduce the constrained decoding strategy into the inference process. The basic idea is to guide the decoder to generate sequences that satisfy as many constraints as possible by pruning infeasible tokens at each decoding step. Specifically, inspired by [1], we frame the format and task-specific constraints as a finite-state machine (FSM). At each decoding step, the FSM computes a set of feasible tokens based on its current state and transits to the next state according to the decoder’s prediction on the set. Take the subtask of UGen as an example (see Figure 3 for its FSM). In the beginning, the FSM stays in the start state \(s_0\), and the feasible tokens for \(s_0\) are the set of element types \(C = \{\text{text}, \text{image}\}\). The decoder is then constrained to predict a token from \(C\). After that, the FSM transits to the next state \(s_1\) and returns all feasible tokens for the left coordinate, i.e., \(NUM = [0, 127]\). When the FSM enters the state \(s_0\), it produces two feasible tokens: \((\text{eos})\) and the separator \(\vert\). If the decoder predicts the separator, the FSM enters state \(s_6\) and continues to guide the decoder to generate more elements; otherwise, the FSM enters the final state and the decoding process finishes. The resulting layout sequence is guaranteed to have at least one element and satisfy the format constraints. The FSMs for other subtasks work similarly.\(^2\)

\(^2\)More FSMs for all subtasks are available in the supplemental material.
Table 2: Quantitative comparisons with baselines.

| Subtasks     | Methods       | RICO                  | PubLayNet             |
|--------------|---------------|-----------------------|-----------------------|
|              | mIoU ↑ | FID ↓ | Align. ↓ | Overlap ↓ | mIoU ↑ | FID ↓ | Align. ↓ | Overlap ↓ |
| UGen         | LayoutTransformer | 0.439 | 22.884 | 0.052 | 0.471 | 0.062 | 36.304 | 0.031 | 0.0009 |
|              | VTN            | 0.686 | 76.064 | 0.461 | 0.694 | 0.210 | 103.373 | 0.205 | 0.211  |
|              | Coarse2Fine   | 0.360 | 46.483 | 0.128 | 0.676 | 0.361 | 50.854 | 0.221 | 0.142  |
|              | UniLayout-S   | 0.742 | 19.688 | 0.047 | 0.547 | 0.417 | 46.522 | 0.029 | 0.0009 |
|              | UniLayout-M   | 0.734 | 11.667 | 0.058 | 0.463 | 0.430 | 30.161 | 0.029 | 0.0008 |
| Gen-T        | NDN-none      | 0.35  | 13.76  | 0.56  | 0.55  | 0.31  | 35.67  | 0.35  | 0.17   |
|              | LayoutGAN++   | 0.298 | 5.954  | 0.261 | 0.620 | 0.297 | 14.875 | 0.124 | 0.148  |
|              | BLT           | 0.216 | 25.633 | 0.150 | 0.983 | 0.140 | 38.684 | 0.036 | 0.196  |
|              | UniLayout-S   | 0.424 | 1.534  | 0.124 | 0.540 | 0.342 | 9.789  | 0.025 | 0.009  |
|              | UniLayout-M   | 0.396 | 2.101  | 0.161 | 0.586 | 0.352 | 10.62  | 0.021 | 0.018  |
| Gen-TS       | BLT           | 0.604 | 0.951  | 0.181 | 0.660 | 0.428 | 7.914  | 0.021 | 0.419  |
|              | UniLayout-S   | 0.618 | 0.775  | 0.151 | 0.547 | 0.472 | 1.045  | 0.027 | 0.037  |
|              | UniLayout-M   | 0.577 | 1.392  | 0.179 | 0.567 | 0.463 | 2.097  | 0.026 | 0.041  |
| Gen-R        | NDN           | 0.36  | -      | 0.56  | -     | 0.31  | -      | 0.36  | -      |
|              | CLG-LO        | 0.286 | 8.898  | 0.311 | 0.615 | 0.277 | 19.738 | 0.123 | 0.200  |
|              | UniLayout-S   | 0.420 | 6.293  | 0.109 | 0.547 | 0.328 | 7.486  | 0.032 | 0.086  |
|              | UniLayout-M   | 0.372 | 11.026 | 0.122 | 0.593 | 0.318 | 11.694 | 0.024 | 0.123  |
| Refinement   | RUITE         | 0.811 | 0.107  | 0.133 | 0.483 | 0.781 | 0.061  | 0.029 | 0.020  |
|              | UniLayout-S   | 0.816 | 0.032  | 0.123 | 0.489 | 0.785 | 0.086  | 0.024 | 0.0055 |
|              | UniLayout-M   | 0.786 | 0.084  | 0.135 | 0.495 | 0.773 | 0.094  | 0.022 | 0.0065 |
| Completion   | LayoutTransformer | 0.363 | 6.679  | 0.194 | 0.478 | 0.077 | 14.769 | 0.019 | 0.0013 |
|              | UniLayout-S   | 0.732 | 4.574  | 0.077 | 0.487 | 0.471 | 10.251 | 0.020 | 0.0022 |
|              | UniLayout-M   | 0.731 | 4.104  | 0.074 | 0.472 | 0.475 | 8.304  | 0.023 | 0.0016 |

4 Experiments

4.1 Setups

Datasets. We compare UniLayout to existing approaches on two public datasets for graphic layouts. RICO [20] is a dataset of mobile app UI that contains 66K+ UI layouts with 25 element types. PubLayNet [40] contains 360K+ document layouts with 5 element types. On both datasets, there are a few layouts with quite a lot of elements, which easily leads to an out-of-memory problem and largely impacts the training speed. Existing studies use multifarious rules to filter out these layouts. In our experiments, we simply filter out the layouts with more than 20 elements. Besides, we use 90%, 5% and 5% of data for training, validation and testing on both datasets.

Baselines. We try our best to reproduce all the existing approaches. Even so, we regret to not include some approaches for the following reasons. First, due to the missing implementation details and hyperparameter settings, we fail to reproduce some approaches [18, 19, 23, 21]. Second, a few approaches consider data-specific attributes and are difficult to be applied to arbitrary datasets about graphic layouts [39, 34]. Ultimately, we compare UniLayout against 1) VTN [3], Coarse2Fine [10] and LayoutTransformer [9] on UGen, 2) NDN-none [17], LayoutGAN++ [13] and BLT [16] on Gen-T, 3) BLT [16] on Gen-TS, 4) NDN [17] and CLG-LO [13] on Gen-R, 5) RUITE [26] on refinement and 6) LayoutTransformer [9] on completion.

Implementation Details. We implement UniLayout by PyTorch [22]. The model is trained using the Adam optimizer [14] with NVIDIA V100 GPUs. For Transformer blocks, we use 8 layers, 8 heads for multi-attention, 512 embedding dimensions and 2048 feed-forward dimensions. Other hyper-parameters, e.g., the batch size, the learning rate, and the task weights in multi-task training, are tuned to achieve the best performance on the validation set. Besides, for Gen-R, following the practice of CLG-LO [13], we randomly sample 10% element relationships as the input. For refinement, following the practice of RUITE [26], we synthesize the input by adding random noise to the position and size of each element, where the noise is sampled from a normal distribution with the mean 0 and the standard variance 0.01. At the inference time, on refinement, we simply use greedy sampling, and on the other subtasks, we use top-k sampling with k = 10 and temperature τ = 0.5 to generate multiple layouts.
4.2 Quantitative Comparison

Evaluation Metrics. To evaluate generation performance comprehensively, we adopt a set of metrics proposed by existing work, each of which measures one aspect of perceptual quality and diversity.

**Maximum Intersection over Union** (mIoU) measures the similarity between the generated layouts and the real layouts, where the similarity is based on the averaged IoU of bounding boxes. We use the same implementation as the previous work [13].

**Alignment** (Align.) measures whether the elements in a generated layout are well-aligned. We modify the original metric [19] by normalizing it by the number of elements.

**Overlap** measures the abnormal overlap area between elements in a generated layout. We improve the metric from previous work [19] by ignoring the elements that serve as a background or padding, e.g., list item, card, background and modal on RICO.

**Frechet Inception Distance** (FID) describes the distribution difference between real and generated layouts. Following the practice of previous work [13], we train a neural network to convert the layouts into representative features and then calculate the distribution difference based on learned features.

Results and Analysis. Table 2 shows the results. As introduced in Section 3.2, we have two versions of UniLayout, where UniLayout-S is trained to tackle each subtask separately and UniLayout-M is trained to serve all subtasks simultaneously. A larger value for mIoU indicates better performance, while smaller values for Align. Overlap and FID indicate better performance. We underline the
We have the following observations from Table 2. First, on both two datasets and all the six subtasks, UniLayout-S and UniLayout-M achieve significantly better performance than the baselines on most metrics (see the underlined results). This demonstrates the advantage of our unified approach over the existing subtask-specific approaches. Second, as for UniLayout-S and UniLayout-M, we find that while UniLayout-S achieves better performance than UniLayout-M on more subtasks and more metrics, UniLayout-M has the advantage over UniLayout-S on UGen and completion (see the bold results). We think it makes sense. On the one hand, handling all the subtasks simultaneously is much more difficult than tackling one subtask. On the other hand, knowledge sharing among different subtasks could boost the performance of some subtasks. We hypothesize that with more training data and larger model sizes, the performance of UniLayout-M could be further improved.

4.3 Qualitative Comparison

Figure 4 and Figure 5 show qualitative comparisons on RICO and PubLayNet. For UGen and completion, each group shows generated layouts from baselines and UniLayout-S. For completion, layouts in the same group are generated by giving the same element as the input. We can observe that while all the methods generate plausible layouts, UniLayout-S performs better at details. For Gen-T, Gen-TS and Gen-R, in each group, the first column shows constraints from the user, and the other columns show the generated layouts from baselines and UniLayout-S. We can find that compared to baselines, UniLayout-S generates layouts with better spacing, less misalignment and

---

More qualitative results about UniLayout-S and UniLayout-M are available in the supplemental material.
fewer unreasonable overlaps. For refinement, in each group, the first column shows a layout draft from users, and the other columns show refined layouts from RUITE and UniLayout-S. The results indicate that UniLayout-S can better improve the layout draft while keeping the overall layout design.

### 4.4 Ablation Studies

In Table 2, we consider the model using Transformer encoder-decoder architecture and the alphabetic order for both the input and output sequences. We denote it as EncDec-A. As discussed in Section 3, there are other model variants. The first one uses position order for the output sequences, denoted as EncDec-P. The second one leverages Transformer decoder only architecture, denoted as Dec-A.

Table 3 compares their performance on RICO. We underline the results where EncDec-A outperforms Dec-A. It is observed that EncDec-A has advantages over Dec-A on refinement, Gen-T, Gen-TS and Gen-R, and performs closely or a little worse than Dec-A on UGen and completion. Compared to UGen and completion, refinement, Gen-T, Gen-TS and Gen-R have more complex user needs and thus more complicated input sequences. This is consistent with our intuition that the bidirectional Transformer encoder is superior in modeling complex sequences. Besides, we use italic to denote better results between EncDec-A and EncDec-P. We find that while EncDec-P outperforms EncDec-A on more metrics, their performance is quite similar most of the time.

Moreover, we study the effectiveness of the constrained decoding strategy on Gen-R. We count how many relationships specified in the input are violated and use it as the metric. Table 4 show the results, where UniLayout and UniLayout-Naive denote the versions with and without constrained decoding strategy. It is shown that the violation rate is reduced significantly by leveraging the constrained decoding strategy. Besides, we also compare UniLayout with baselines. The results show that UniLayout outperforms all the baselines on PubLayNet but has a worse violation rate than CLG-LO on RICO. To figure out the reason, we revisit the quantitative and qualitative performance of CLG-LO on RICO in Table 2 and Figure 4. We find that CLG-LO does not make a good trade-off: the aesthetic quality of layouts generated by CLG-LO is significantly worse than that of UniLayout.

### 5 Conclusion

This paper introduces UniLayout, a simple yet effective approach to tackle various subtasks of graphic layout generation in a unified manner. UniLayout casts subtasks as sequence-to-sequence transformations, and adopts the Transformer encoder-decoder architecture to solve them. While simple, UniLayout significantly outperforms the previous subtask-specific approaches. By prepending a task prefix to input sequences and combining all subtasks’ data for training, UniLayout can serve all the subtasks simultaneously and boost the performance of some subtasks even further. Besides, UniLayout makes it possible to explore developing an efficient and powerful pre-trained model for graphic layout generation, which we leave as an important future work. Moreover, we also plan to investigate more practical user requirements for layout design and adopt UniLayout to tackle them.
Acknowledgments and Disclosure of Funding

References

[1] P. Anderson, B. Fernando, M. Johnson, and S. Gould. Guided open vocabulary image captioning with constrained beam search. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 936–945, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1098. URL https://aclanthology.org/D17-1098.

[2] V. Aribandi, Y. Tay, T. Schuster, J. Rao, H. S. Zheng, S. V. Mehta, H. Zhuang, V. Q. Tran, D. Bahri, J. Ni, J. Gupta, K. Hui, S. Ruder, and D. Metzler. Ext5: Towards extreme multi-task scaling for transfer learning. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=Vzh1BFUCiIX.

[3] D. M. Arroyo, J. Postels, and F. Tombari. Variational transformer networks for layout generation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 13642–13652, 2021.

[4] N. D. Cao, L. Wu, K. Popat, M. Artetxe, N. Goyal, M. Plekhanov, L. Zettlemoyer, N. Cancedda, S. Riedel, and F. Petroni. Multilingual autoregressive entity linking. In arXiv pre-print 2103.12528, 2021. URL https://arxiv.org/abs/2103.12528.

[5] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. Hinton. Pix2seq: A language modeling framework for object detection. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=e42KbIw6Wb.

[6] J. Cho, J. Lei, H. Tan, and M. Bansal. Unifying vision-and-language tasks via text generation. In International Conference on Machine Learning, pages 1931–1942. PMLR, 2021.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[8] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12873–12883, 2021.

[9] K. Gupta, J. Lazarow, A. Achille, L. S. Davis, V. Mahadevan, and A. Shrivastava. Layouttransformer: Layout generation and completion with self-attention. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 1004–1014, 2021.

[10] Z. Jiang, S. Sun, J. Zhu, J.-G. Lou, and D. Zhang. Coarse-to-fine generative modeling for graphic layouts. In AAAI’22, February 2022. URL https://www.microsoft.com/en-us/research/publication/coarse-to-fine-generative-modeling-for-graphic-layouts/.

[11] M. Josifoski, N. De Cao, M. Peyrard, and R. West. Genie: Generative information extraction. arXiv preprint arXiv:2112.08340, 2021.

[12] D. Khashabi, S. Min, T. Khot, A. Sabharwal, O. Tafjord, P. Clark, and H. Hajishirzi. UNIFIEDQA: Crossing format boundaries with a single QA system. In Findings of the Association for Computational Linguistics: EMNLP 2020, pages 1896–1907, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.171. URL https://aclanthology.org/2020.findings-emnlp.171.

[13] K. Kikuchi, E. Simo-Serra, M. Otani, and K. Yamaguchi. Constrained graphic layout generation via latent optimization. In Proceedings of the 29th ACM International Conference on Multimedia, pages 88–96, 2021.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In ICLR, 2015. URL http://arxiv.org/abs/1412.6980.

[15] D. P. Kingma and M. Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.
[16] X. Kong, L. Jiang, H. Chang, H. Zhang, Y. Hao, H. Gong, and I. Essa. Blt: Bidirectional layout transformer for controllable layout generation. arXiv preprint arXiv:2112.05112, 2021.

[17] H.-Y. Lee, L. Jiang, I. Essa, P. B. Le, H. Gong, M.-H. Yang, and W. Yang. Neural design network: Graphic layout generation with constraints. In European Conference on Computer Vision, pages 491–506. Springer, 2020.

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

[19] J. Li, J. Yang, J. Zhang, C. Liu, C. Wang, and T. Xu. Attribute-conditioned layout gan for automatic graphic design. IEEE Transactions on Visualization and Computer Graphics, 27(10): 4039–4048, 2020.

[20] T. F. Liu, M. Craft, J. Situ, E. Yumer, R. Mech, and R. Kumar. Learning design semantics for mobile apps. In The 31st Annual ACM Symposium on User Interface Software and Technology, UIST ’18, pages 569–579, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5948-1. doi: 10.1145/3242587.3242650. URL http://doi.acm.org/10.1145/3242587.3242650.

[21] D. D. Nguyen, S. Nepal, and S. S. Kanhere. Diverse multimedia layout generation with multi choice learning. In Proceedings of the 29th ACM International Conference on Multimedia, pages 218–226, 2021.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, F. Fieremans, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

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

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[25] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

[26] S. Rahman, V. P. Sermuga Pandian, and M. Jarke. Ruite: Refining ui layout aesthetics using transformer encoder. In 26th International Conference on Intelligent User Interfaces-Companion, pages 81–83, 2021.

[27] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In International Conference on Machine Learning, pages 8821–8831. PMLR, 2021.

[28] A. Razavi, A. Van den Oord, and O. Vinyals. Generating diverse high-fidelity images with vq-vae-2. Advances in neural information processing systems, 32, 2019.

[29] V. Sanh, A. Webson, C. Raffel, S. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, A. Raja, M. Dey, M. S. Bari, C. Xu, U. Thakker, S. S. Sharma, E. Szczesnica, T. Kim, G. Chhablani, N. Nayak, D. Datta, J. Chang, M. T.-J. Jiang, H. Wang, M. Manica, S. Shen, Z. X. Yong, H. Pandey, R. Bawden, T. Wang, T. Neeraj, J. Rozen, A. Sharma, A. Santilli, T. Fevry, J. A. Fries, R. Teehan, T. L. Scao, S. Biderman, L. Gao, T. Wolf, and A. M. Rush. Multi-task prompted training enables zero-shot task generalization. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=9VrbSW0W1A
[30] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[31] A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[33] T. Xie, C. H. Wu, P. Shi, R. Zhong, T. Scholak, M. Yasunaga, C.-S. Wu, M. Zhong, P. Yin, S. I. Wang, et al. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*, 2022.

[34] K. Yamaguchi. Canvasvae: Learning to generate vector graphic documents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5481–5489, 2021.

[35] H. Yan, T. Gui, J. Dai, Q. Guo, Z. Zhang, and X. Qiu. A unified generative framework for various NER subtasks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5808–5822, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.451. URL [https://aclanthology.org/2021.acl-long.451](https://aclanthology.org/2021.acl-long.451).

[36] Z. Yang, Z. Gan, J. Wang, X. Hu, F. Ahmed, Z. Liu, Y. Lu, and L. Wang. Crossing the format boundary of text and boxes: Towards unified vision-language modeling. *arXiv preprint arXiv:2111.12083*, 2021.

[37] Z. Zhang, J. Ma, C. Zhou, R. Men, Z. Li, M. Ding, J. Tang, J. Zhou, and H. Yang. UFC-BERT: Unifying multi-modal controls for conditional image synthesis. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL [https://openreview.net/forum?id=iEEAPq3TUEZ](https://openreview.net/forum?id=iEEAPq3TUEZ).

[38] X. Zhao, B. He, Y. Wang, Y. Li, F. Mi, Y. Liu, X. Jiang, Q. Liu, and H. Chen. Unids: A unified dialogue system for chit-chat and task-oriented dialogues. *arXiv preprint arXiv:2110.08032*, 2021.

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

[40] X. Zhong, J. Tang, and A. J. Yepes. Publaynet: largest dataset ever for document layout analysis. *arXiv preprint arXiv:1908.07836*, 2019.

**Checklist**

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We do not have theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [N/A] The assets we used are public.

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [N/A] The assets we used are public.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We do not use crowdsourcing or conducted research with human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]
A Visualization of Similarity among Coordinate Tokens

To formulate the layout as token sequence, we quantize the element coordinates $x_i, y_i, w_i$ and $h_i$ into $[0, 127]$, and then encode them as discrete tokens. Inspired by [5], we explore the numerical relations among the coordinate tokens via their embeddings.

We compute the cosine similarity of coordinate embeddings of UniLayout-M on PubLayNet. The results are shown in Figure 6. The points on the diagonal represent the similarities that the token embeddings compute with themselves, with the lightest color indicating the similarity of 1. The color darkens along the orthogonal direction of the diagonal, which means that the coordinate embeddings which represent the closer number values have higher similarities than the embeddings which represent far number values.

Figure 6: (a) Cosine similarity among the embeddings of all the 128 coordinate tokens of UniLayout-M on PubLayNet. (b) is only for the first 32 tokens, and (c) is only for the first five tokens, which demonstrates more clearly that the closer coordinate embeddings have higher similarities.

B FSMs in Constrained Decoding Strategy for All Subtasks

Here we introduce the finite-stage machines (FSMs) in the constrained decoding strategy for all the subtasks (see Figure 7). For convenience, we use specific instances to illustrate the FSMs.

The FSM for UGen and Completion is shown in Figure 7(a), which we have introduced in Section 3.2. Completion is similar to UGen since they both have no special constraints and only need to make sure to predict element, coordinate, or special token in the correct state.

An example of FSM for Gen-T and Refinement is shown in Figure 7(b). Taking the FSM for UGen as the basis, Gen-T and Refinement further require that the predicted element types in the generated layout comply with the type constraints. In this example, the constrained types are text and image. When the FSM enters the states $s_0$ and $s_6$, the constrained types specify the tokens.

An example of FSM for Gen-TS is shown in Figure 7(c). Beyond Gen-T, Gen-TS specifies the width and height of each element. As shown in this example, the tokens are directly specified by the given size constraints when the FSM enters the $s_3$, $s_4$ and $s_9$, $s_{10}$.

Gen-R has the constraints of the element types and the relationships between elements. The type constraints are handled the same way as the FSM for Gen-T, Refinement, and Gen-TS. The relationship constraints require the relative position and size relationships between the elements, so the feasible range for the coordinate values $x, y, w, h$ of the element that is currently being predicted will possibly be restrained by the prior predicted elements. As shown by the example in Figure 7(d) when it stays in the state $s_8$, the FSM computes the feasible tokens for the top position $y'$ of the element "image" depends on the bottom position $(y + h)$ of "text", which has been predicted in the previous states, to meet the relationship constraint "image at the bottom of text".
The training epochs and batch sizes of UniLayout-M and UniLayout-S for each subtask are shown in Table 5. On both RICO and PubLayNet, we use optimizer Adafactor with a learning rate of 0.0001. We use the learning rate warmup, and the numbers of the warmup steps for each model are also shown in Table 5. For UniLayout-M, the data sample weights of each subtask for constructing multitask dataset are shown in Table 6, which are the same on the two datasets. In practice, we find that slightly increasing the weights of Refinement and Gen-TS can better balance the performance for all subtasks. The subtask loss weights in UniLayout-M are all set as 1.

Table 5: Hyperparameters for training UniLayout.

| Model       | Task      | RICO     | PubLayNet |
|-------------|-----------|----------|-----------|
|             |           | Epoch    | Batch Size| Warmup Steps| Epoch | Batch Size| Warmup Steps|
| UniLayout-S | UGen      | 100      | 32        | 1000        | 100   | 100       | 1000        |
|             | Gen-T     | 100      | 32        | 1000        | 100   | 100       | 1000        |
|             | Gen-TS    | 100      | 32        | 1000        | 100   | 100       | 4000        |
|             | Gen-R     | 100      | 32        | 1000        | 64    | 3000      | 3000        |
|             | Completion| 100      | 32        | 1000        | 100   | 64        | 3000        |
|             | Refinement| 100      | 32        | 1000        | 100   | 4000      | 2000        |
| UniLayout-M | Multitasks| 300      | 32        | 4000        | 100   | 64        | 3000        |

Table 6: Subtask sample weights in UniLayout-M.

| Subtasks | UGen | Gen-T | Gen-TS | Gen-R | Refinement | Completion |
|----------|------|-------|--------|-------|------------|------------|
| Weights  | \(\frac{1}{12}\) | \(\frac{1}{12}\) | \(\frac{1}{12}\) | \(\frac{1}{12}\) | \(\frac{1}{12}\) | \(\frac{1}{12}\) |
D Qualitative Results

We show more generated layouts here to better demonstrate the outstanding performance of UniLayout. For all the six subtasks, we present the generated layouts by UniLayout-S and UniLayout-M on both RICO and PubLayNet.

D.1 Unconstrained Generation

The generated layouts of UGen on RICO and PubLayNet are shown in Figure 8 and Figure 9. As we introduced in Section 3.1, the input sequence of UGen only contains \{⟨sos⟩⟨eos⟩\}, since it considers no constraints. To ensure the diversity of generated layouts from identical input, we leverage top-k sampling for UniLayout during predicting the tokens.

Figure 8: Qualitative results of UGen on Rico.
D.2 Generation Conditioned on Element Types

The generated layouts of Gen-T are shown in Figure 10 and Figure 11. For UniLayout-S and UniLayout-M, we show six groups of layouts, respectively. Each group contains two layouts generated from the same element type constraints below. It can be seen that UniLayout can generate diverse layouts while satisfying the same constraints by leveraging top-k sampling.

D.3 Generation Conditioned on Element Types and Sizes

The generated layouts of Gen-TS are shown in Figure 12 and Figure 13. Similar with Section D.2, each group contains two layouts generated from the same element type and size constraints in the below table.

Figure 9: Qualitative results of UGen on PubLayNet.
Figure 10: Qualitative results of Gen-T on Rico. The element type constraints are in the table.
Figure 11: Qualitative results of Gen-T on PubLayNet. The element type constraints are in the table.
Figure 12: Qualitative results of Gen-TS on Rico. Tables show the element type and size constraints.
Figure 13: Qualitative results of Gen-TS on PubLayNet. Tables show the element type and size constraints.
### D.4 Generation Conditioned on Element Relationships

| Element Relationship Constraints |
|----------------------------------|
| (1) Text*2, Pager Indicator, Image, Text Button , Image 1 top Text 2, Text Button 1 bottom Image 1 |
| (2) Text*2, Multi-Tab, Icon, List Item*3, Text Button*3, Toolbar, Text Button 1 top canvas, List Item 1 larger Text 2, List Item 2 center Text 2, Text Button 3 top Text 2, Icon 1 smaller Multi-Tab 1, List Item 2 larger Multi-Tab 1, List Item 3 center List Item 1, Text Button 1 top List Item 3, Text Button 2 top List Item 3, Text Button 3 smaller List Item 3, Text Button 3 right Text Button 1, Toolbar 1 top Text Button 1 |
| (3) Text*3, Image*4, Icon*4, Text Button, Input*3, Text Button 1 larger Text 1, Icon 2 larger Text 2, Icon 3 smaller Text 2, Icon 4 smaller Text 2, Icon 4 bottom Text 2, Input 1 bottom Text 2, Image 3 smaller Text 3, Icon 3 smaller Text 3, Icon 4 smaller Text 3, Input 2 smaller Text 3, Icon 3 smaller Image 1, Text Button 1 larger Image 1, Icon 1 larger Image 2, Icon 3 bottom Image 2, Icon 2 larger Image 3, Icon 2 right Image 3, Icon 3 smaller Image 3, Icon 2 larger Image 4, Icon 2 smaller Icon 1, Input 3 larger Icon 1, Input 3 bottom Icon 1, Icon 4 equal Icon 3, Input 1 top Icon 3 |
| (4) Text*6, Multi-Tab, Image, Icon*3, Text Button*2, Image 1 bottom canvas, Text Button 2 bottom canvas, Text 5 smaller Text 1, Text 6 bottom Text 1, Multi-Tab 1 center Text 1, Text 5 smaller Text 2, Icon 1 larger Text 2, Icon 3 smaller Text 2, Text Button 2 larger Text 2, Icon 3 bottom Text 3, Text 6 smaller Text 4, Text Button 2 right Text 6, Text Button 2 bottom Multi-Tab 1, Icon 1 larger Image 1, Icon 2 bottom Image 1, Icon 3 equal Icon 2, Text Button 1 larger Icon 2, Text Button 1 left Icon 2 |
| (5) Text, Pager Indicator*2, Image*4, Icon*9, Text Button, Toolbar, Text 1 bottom canvas, Image 7 bottom canvas, Toolbar 1 top canvas, Text Button 1 larger Text 1, Icon 2 larger Pager Indicator 1, Toolbar 1 center Pager Indicator 1, Image 2 bottom Pager Indicator 2, Image 3 bottom Pager Indicator 2, Icon 6 center Pager Indicator 2, Image 3 bottom Image 1, Icon 3 smaller Image 1, Icon 6 center Image 1, Icon 8 smaller Image 1, Icon 9 smaller Image 1, Icon 3 equal Image 2, Icon 4 top Image 2, Icon 8 equal Image 2, Icon 2 top Image 3, Icon 1 larger Image 4, Icon 3 larger Image 4, Icon 7 left Image 4, Icon 8 larger Image 4, Icon 3 right Icon 1, Icon 6 smaller Icon 1, Icon 7 bottom Icon 1, Text Button 1 larger Icon 2, Toolbar 1 center Icon 2, Icon 6 bottom Icon 4, Icon 9 larger Icon 4, Icon 8 equal Icon 7, Icon 9 left Icon 8, Toolbar 1 top Text Button 1 |
| (6) Text*2, Icon, Text Button, Toolbar, Input, Text 2 bottom Text 1, Text Button 1 bottom Text 2, Text Button 1 larger Icon 1 |

(a) UniLayout-S

| Element Relationship Constraints |
|----------------------------------|
| (1) Text*2, Icon, Text Button*2, Toolbar, Input, Text Button 1 equal Text 2, Text Button 1 larger Icon 1, Input 1 larger Icon 1, Input 1 smaller Toolbar 1 |
| (2) Text, Advertisement, Image*2, Icon, Text Button*4, Web View*2, Input, Text Button 2 bottom canvas, Text Button 3 bottom canvas, Input 1 top canvas, Advertisement 1 bottom Text 1, Text Button 3 bottom Text 1, Image 1 smaller Advertisement 1, Text Button 4 bottom Advertisement 1, Icon 1 larger Web View 1, Web View 1 larger Image 1, Text Button 2 bottom Icon 1, Text Button 3 smaller Text Button 1, Web View 1 bottom Text Button 2, Web View 1 larger Text Button 4, Web View 2 equal Web View 1 |
| (3) Text*2, Image*2, Text Button*3, Image 2 smaller Text 1, Text Button 1 bottom Image 2, Text Button 3 equal Text Button 2, Text Button 3 right Text Button 2 |
| (4) Text*2, Pager Indicator, Icon, Text Button*2, Toolbar, Input*4, Text Button 2 bottom Text 1, Input 3 smaller Text 1, Input 4 bottom Text 1, Icon 1 top Text 2, Input 3 larger Text 2, Text Button 1 larger Pager Indicator 1, Text Button 2 larger Pager Indicator 1, Toolbar 1 larger Pager Indicator 1, Text Button 2 larger icon 1, Toolbar 1 center Icon 1, Input 3 bottom Text Button 1, Input 4 larger Text Button 1, Input 1 smaller Toolbar 1 |
| (5) Text, Background Image, Icon, Text Button*3, Input*2, Icon 1 top canvas, Icon 1 bottom Text 1, Text Button 3 bottom Text 1, Text Button 2 smaller Text Button 1, Text Button 3 equal Text Button 2, Input 1 larger Text Button 2, Input 2 larger Text Button 3 |
| (6) Text*3, Image*4, Icon*4, Text Button, Input*3, Text Button 1 larger Text 1, Icon 2 larger Text 2, Icon 3 smaller Text 2, Icon 4 smaller Text 2, Icon 4 bottom Text 2, Input 1 bottom Text 2, Image 3 smaller Text 3, Icon 3 smaller Text 3, Icon 4 smaller Text 3, Icon 2 smaller Text 3, Icon 3 smaller Image 1, Text Button 1 larger Image 1, Icon 1 larger Image 2, Icon 3 bottom Image 2, Icon 2 larger Image 3, Icon 2 right Image 3, Icon 3 smaller Image 3, Icon 2 larger Image 4, Icon 2 smaller Icon 1, Input 3 larger Icon 1, Input 3 bottom Icon 1, Icon 4 equal Icon 3, Input 1 top Icon 3 |

(b) UniLayout-M

Figure 14: Qualitative results of Gen-R on Rico. Tables show the element relationship constraints.
The generated layouts of Gen-R are shown in Figure 14 and Figure 15. We list the relationship constraints that the layouts generated from in the below table.

| Element Relationship Constraints                                                                                     |
|----------------------------------------------------------------------------------------------------------------------|
| (1) text*9, title, table, text 2 bottom text 1, title 1 bottom text 1, text 3 smaller text 2, text 4 smaller text 2, text 4 top text 2, text 8 smaller text 2, text 8 top text 2, text 1 left text 2, text 6 top text 5, text 9 bottom text 5, table 1 top text 6 |
| (2) text*9, title*2, text 6 right text 1, text 7 larger text 1, text 8 larger text 1, text 9 bottom text 1, text 6 top text 2, text 7 larger text 2, text 8 left text 3, text 7 bottom text 4, text 9 bottom text 4, text 9 left text 5, text 7 equal text 6, text 8 top text 7 |
| (3) text*2, figure*2, figure 1 top canvas, figure 1 top text 2                                                      |
| (4) text*8, title*4, table, text 6 center canvas, text 3 larger text 1, text 7 top text 1, title 4 smaller text 1, text 3 larger text 2, table 1 bottom text 2, text 6 larger text 3, title 3 smaller text 4, text 4 smaller text 4, text 7 smaller text 5, title 1 top text 5, title 2 smaller text 5, title 3 smaller text 5, table 1 smaller text 5, title 1 top text 6, title 4 smaller title 1, title 4 larger title 3, title 4 top title 3 |
| (5) text*7, title*3, list*2, figure, text 4 center text 1, text 5 smaller text 2, title 2 smaller text 2, list 2 larger text 2, figure 1 top text 2, list 2 left text 3, list 2 larger text 4, title 2 smaller text 5, figure 1 larger text 5, title 1 smaller text 6, title 1 bottom text 7, list 1 smaller text 7, list 1 top text 7, list 1 top title 1, figure 1 larger title 3 |
| (6) text*6, figure, text 2 top text 1, text 3 left text 2, text 5 larger text 4                                       |

(a) UniLayout-S

| Element Relationship Constraints                                                                                     |
|----------------------------------------------------------------------------------------------------------------------|
| (1) text*8, figure, text 5 center canvas, figure 1 top text 1, text 4 smaller text 2, text 5 bottom text 3, text 6 smaller text 4, text 7 equal text 4, text 7 smaller text 5, figure 1 larger text 7 |
| (2) text*5, title*2, table, text 3 bottom canvas, title 1 smaller text 1, text 3 left text 2, text 5 larger text 3, title 2 right text 5, title 2 bottom title 1 |
| (3) text*9, title*2, text 6 right text 1, text 7 larger text 1, text 8 larger text 1, text 9 bottom text 1, text 6 top text 2, text 7 larger text 2, text 8 left text 3, text 7 bottom text 4, text 9 bottom text 4, text 9 left text 5, text 7 equal text 6, text 8 top text 7 |
| (4) text*10, title, list*2, table, text 1 center canvas, text 2 top canvas, text 3 bottom canvas, text 3 larger text 1, list 2 larger text 1, text 4 larger text 2, text 7 larger text 2, title 1 bottom text 2, text 10 top text 3, table 1 top text 3, text 1 larger text 4, text 7 equal text 5, text 8 larger text 6, list 2 larger text 6, table 1 top text 6, text 8 bottom text 7, list 2 right text 8, title 1 top text 9, list 1 smaller text 9 |
| (5) text*9, list*2, table*2, text 9 center canvas, list 2 bottom canvas, text 2 smaller text 1, text 2 bottom text 1, text 7 larger text 1, list 1 bottom text 1, list 1 larger text 1, text 4 larger text 2, text 9 bottom text 2, list 2 larger text 2, list 2 larger text 2, text 9 smaller text 4, list 1 smaller text 4, list 2 larger text 4, text 8 top text 7, list 2 top text 7, table 2 top text 7, table 2 larger list 2 |
| (6) text*6, figure, text 2 top text 1, text 3 left text 2, text 5 larger text 4                                       |

(b) UniLayout-M

Figure 15: Qualitative results of Gen-R on PubLayNet. Tables show the element relationship constraints.
D.5 Refinement

The generated layouts of Refinement are shown in Figure 16 and Figure 17. For UniLayout-S and UniLayout-M, we show six groups of layouts, respectively. Each group contains two layouts. The left one is the noised layout that needs refinement, and the right one is the layout refined by UniLayout.

Figure 16: Qualitative results of Refinement on Rico.
Figure 17: Qualitative results of Refinement on PubLayNet.
D.6 Completion

The generated layouts of Completion are shown in Figure 18 and Figure 19. For UniLayout-S and UniLayout-M, we show four groups of layouts, respectively. Each group contains three layouts generated from the same first element. By leveraging top-k sampling, UniLayout can complete the partial layout into various final layouts.

Figure 18: Qualitative results of Completion on Rico.
Figure 19: Qualitative results of Completion on PubLayNet.
E Compare with finetuning on T5

Since we use the same model architecture with T5 [25], we compare our UniLayout with finetuning on the T5-large model of each layout generation subtask. The quantitative results are shown in Table 7 and we highlight the results of the method which outperforms another one. It can be found that although using a larger size model with pre-trained weights, finetuning on the T5 model does not achieve better performance on all the tasks and metrics, while bringing in larger training costs. The possible reason is that T5 is pre-trained on natural language data, while the layout sequence is quite different from the natural language. This makes pre-trained weights not bring too much help.

Table 7: Quantitative comparisons with finetuning on T5.

| Subtasks | Methods        | RICO  | PubLayNet |
|----------|----------------|-------|-----------|
|          | mIoU ↑ | FID ↓ | Align. ↓ | Overlap ↓ | mIoU ↑ | FID ↓ | Align. ↓ | Overlap ↓ |
| UGen     | UniLayout   | 0.742 | 19.688   | 0.047    | 0.547  | 0.417 | 46.522   | 0.029    | 0.0009   |
|          | finetuned-T5| 0.740 | 12.675   | 0.062    | 0.687  | 0.475 | 15.062   | 0.044    | 0.0007   |
| Gen-T    | UniLayout   | 0.424 | 15.34    | 0.124    | 0.540  | 0.342 | 9.789    | 0.025    | 0.0000   |
|          | finetuned-T5| 0.440 | 9.113    | 0.110    | 0.513  | 0.239 | 16.638   | 0.034    | 0.0074   |
| Gen-TS   | UniLayout   | 0.618 | 0.775    | 0.151    | 0.547  | 0.472 | 1.045    | 0.027    | 0.037    |
|          | finetuned-T5| 0.644 | 0.839    | 0.153    | 0.543  | 0.477 | 2.769    | 0.025    | 0.033    |
| Gen-R    | UniLayout   | 0.420 | 6.293    | 0.109    | 0.547  | 0.328 | 7.486    | 0.032    | 0.086    |
|          | finetuned-T5| 0.489 | 2.532    | 0.109    | 0.490  | 0.377 | 5.699    | 0.019    | 0.002    |
| Refinement | UniLayout    | 0.816 | 0.032    | 0.123    | 0.489  | 0.785 | 0.086    | 0.024    | 0.0055   |
|          | finetuned-T5| 0.833 | 0.035    | 0.115    | 0.477  | 0.782 | 0.082    | 0.025    | 0.0011   |
| Completion | UniLayout    | 0.732 | 4.574    | 0.077    | 0.487  | 0.471 | 10.251   | 0.020    | 0.0022   |
|          | finetuned-T5| 0.739 | 3.002    | 0.124    | 0.664  | 0.479 | 7.044    | 0.027    | 0.0013   |

28