StructLayoutFormer: Conditional Structured Layout Generation via Structure Serialization and Disentanglement

Xin Hu, Pengfei Xu, Jin Zhou, Hongbo Fu, and Hui Huang

Abstract—Structured layouts are preferable in many 2D visual contents (e.g., GUIs, webpages) since the structural information allows convenient layout editing. Computational frameworks can help create structured layouts but require heavy labor input. Existing data-driven approaches are effective in automatically generating fixed layouts but fail to produce layout structures. We present *StructLayoutFormer*, a novel Transformer-based approach for conditional structured layout generation. We use a structure serialization scheme to represent structured layouts as sequences. To better control the structures of generated layouts, we disentangle the structural information from the element placements. Our approach is the first data-driven approach that achieves conditional structured layout generation and produces realistic layout structures explicitly. We compare our approach with existing data-driven layout generation approaches by including post-processing for structure extraction. Extensive experiments have shown that our approach exceeds these baselines in conditional structured layout generation. We also demonstrate that our approach is effective in extracting and transferring layout structures. The code is publicly available at https://github.com/Teagrus/StructLayoutFormer.

Index Terms—Transformer, Conditional layout generation, Structured layout generation

1 Introduction

Graphic layout plays an important role in graphic design. Traditionally, when designing 2D visual contents, e.g., posters, webpages, or GUIs, their layouts are created manually with interactive tools [1], [2] or semi-automatically with computational frameworks [3]–[5]. With the rise of learning techniques, this layout generation task can be achieved automatically in a data-driven manner. Existing data-driven approaches [6]–[15] have adopted GAN [16], VAE [17], GNN [18], Transformer [19], Diffusion model [20], etc., for the automatic layout generation task and obtained remarkable progress.

Compared with the computational methods for semi-automatic layout creation, data-driven approaches are more efficient in generating large numbers of layouts. However, existing data-driven approaches focus on producing fixed layouts represented as sets of bounding boxes, which is not suitable for further layout adjustment. In contrast, computational methods can help create structured layouts containing relations among elements. This structural information enables structure-preserving manipulation of layouts. For example, the GUI layouts created with existing computational frameworks [2] contain relations among GUI elements. With such structural information, these GUI layouts can automatically adapt to different screen sizes [21] without manual input (see Figure 1). Nevertheless, these computational methods often require heavy user labor input. It would be meaningful to exploit learning techniques to generate structured layouts automatically.

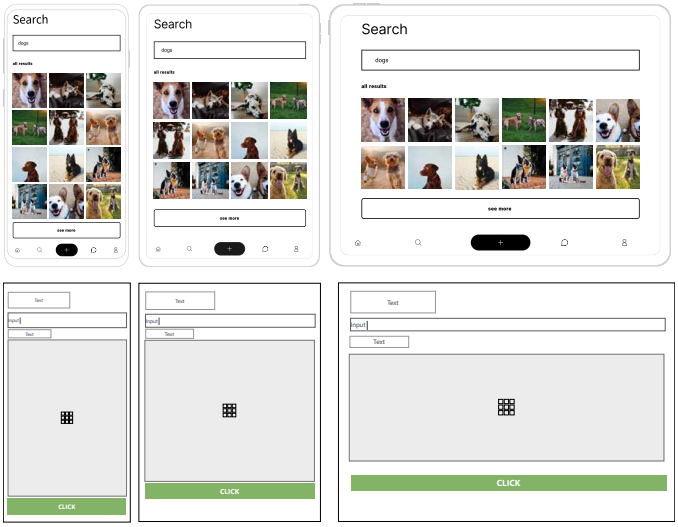


Fig. 1. GUI layouts (top) and their major internal structures (below). In these examples, all images are under grid nodes, and their arrangements can be automatically adjusted to different screen sizes.

There exist several structured layout datasets, e.g., RICO [22] and WebForest [23], in which layouts are represented as trees. A node of a layout tree represents a graphic element, which could be visible, e.g., a text box, or invisible, e.g., a linear arrangement. An internal node acts as a container of its children nodes. For such structured data, recursive neural networks (RvNNs) [24] are often adopted for the automatic generation task. They have been successfully

- *Corresponding author: Pengfei Xu.*
- *X. Hu (qzlyhx@hotmail.com), P. Xu (xupengfei.cg@gmail.com), J. Zhou (doudin2618@gmail.com), and H. Huang (hhzhiyan@gmail.com) are with the College of Computer Science and Software Engineering, Shenzhen University.*
- *H. Fu (fuplus@gmail.com) is with the Hong Kong University of Science and Technology.*

used for generating structured 3D shapes [25]–[27] and indoor scenes [28]. However, RvNNs have the following limitations that prevent them from applying to the structured layout generation problem. First, RvNNs are inefficient in training since they require a bottom-up training procedure, which is not parallel for each sample. This is critical since a layout often contains many elements and has a complicated structure compared with structured 3D shapes or indoor scenes. Second, RvNNs prohibit messages from passing between different branches of a tree structure, restricting their learning capacity. Third, the existing RvNN-based methods are unsuitable for conditional generation, thus limiting the application scenarios. Compared with RvNN, Transformer is more potent in learning the arrangement patterns of graphic elements and can easily achieve conditional layout generation, as confirmed by existing approaches [6], [8], [14], [29]. However, exploiting a Transformer architecture for the structured layout generation problem is nontrivial.

We propose *StructLayoutFormer*, a novel Transformer-based approach for conditional structured layout generation. In this work, a structured layout is represented as a layout tree. This representation has been adopted by existing layout datasets [22], [23] and computational methods [5], [11], [23], [30]–[32]. To adapt the Transformer architecture, we use a structure serialization scheme to map a layout tree to a sequence of tokens (see Figure 2). This sequence contains all the structural information and can faithfully recover the layout tree. Our model produces such sequences autoregressively as the generated structured layouts. To better control the structure of the generated layouts, we disentangle the structural information of a layout from its element placement by constructing a latent space that embeds high-level layout structures. With this disentanglement, we can use a structure code as a condition for layout generation. It also enables our approach to support existing conditional layout generation tasks without considering a structured representation as input. Structure codes also bring more generation variety; our approach can produce different results under the same input conditions. Without structure codes, generating diverse structures can only be achieved through probabilistic sampling, which is less controllable.

We have extensively tested our approach on two structured layout datasets, i.e., RICO [22] and WebForest [23]. The experiments include several conditional layout generation tasks. To better examine the effectiveness of our approach in structured layout generation, we compare our approach with the state-of-the-art layout generation approaches, including LayoutFormer++ [29], BLT [14], LayoutDM [10], and LayoutTransformer [8]. Different from our approach, these approaches cannot produce layout structures explicitly. However, they can find the arrangement patterns existing in the datasets. Therefore, we treat the internal nodes of layout trees as additional elements and use these approaches to produce layout structures implicitly. We also compare our approach with GTLayout [33], an RvNN-based method that can produce layout structures. To quantitatively compare these approaches, besides the frequently-used metrics for measuring the quality of the element arrangement, we introduce additional metrics that measure the quality of the produced layout structures. The experiments show that our approach outperforms these approaches in the adopted metrics. We also demonstrate that our approach can extract and transfer layout structures.

2 Related work

Structured layout creation. Many approaches have been proposed for structured layout creation due to the wide application of such layouts. Xu et al. [1] proposed a framework to create structured layouts interactively. The Auckland layout editor [2] was another interactive framework for structured GUI layout creation. Some computational approaches reduced the labor input for layout creation. O'Donovan et al. [32] presented an optimization method for generating structured grid layouts based on design principles. Kikuchi et al. [23] proposed a method to transfer structures between existing webpage layouts. Girds [3] was a computational framework for structured GUI layout generation from heuristic rules. Scout [4] was a system that helped designers explore structured GUI layouts. Xu et al. [5] proposed a method to create novel structured layouts via layout blending. Although these works help create structured layouts, they more or less require manual input or specification from people. In contrast, our approach is purely data-driven and may not require human intervention.

Learning-based layout generation. With the rise of learning techniques, many data-driven approaches have been proposed for layout generation. LayoutGAN [34] was an early work exploiting GAN [16] for layout generation. LayoutGAN++ [13] also adopted GAN and improved the quality of generated layouts. LayoutVAE [12] adopted two VAEs [17] to generate layouts. LayoutTransformer [8] and VTN [6] exploited Transformer [19] for layout generation. Jiang et al. [11] also exploited Transformer for layout generation. They realized the importance of layout structures. NDN [35] adopted a GNN [18] for layout generation. Recently, more works have focused on conditional layout generation. BLT [14] extended BERT [36] for conditional layout generation. LayoutFormer++ [29] could use geometric relations among elements as conditions for layout generation. The diffusion model [20] was also extended to the layout generation task, e.g., LayoutDMs [7], [10] and LDGM [9] achieving controllable layout generation with Diffusion models. These recent works have obtained remarkable progress in layout generation. However, none of them can produce layout structures explicitly. READ [37] exploited a binary tree structure for layout generation but could not be extended to general structure generation. GTLayout [33] adopted RvNN to achieve structured layout generation. However, it was designed for specific layout structures, including three types of element arrangements, making it difficult to adapt to real application scenarios that involve diverse structures. Jiang et al. [11] proposed a VAE-based method for generating a two-layer structured layout. In contrast, our data-driven approach explicitly achieves conditional generation of layouts with realistic structures.

Large language models (LLMs) have been extensively applied to layout generation tasks in recent studies. These approaches typically represent layouts in textual formats and design appropriate prompt instructions to leverage the reasoning capabilities of LLMs for generating layout samples. Most existing studies have not considered structured

layout generation tasks [38]–[41]. LayoutGPT [38] converts text inputs to image and indoor scene layouts. LayoutNUWA [39] uses the SVG format to represent layouts and set unknown values as a mask token to achieve conditional layout generation. PosterLLaVa [40] combines background image and textual requirements to generate satisfied layouts. LLplace [41] focuses on the 3D indoor scene layout generation and editing. Lin et al. [42] introduced Parse-Then-Place, a text-to-layout method that takes a rough description of the target layout, including the target's structure and contents, as input to generate a structured layout. This text-to-layout method cannot take precise constraints, e.g., element geometry, as conditions. LLMs also demonstrate the capability of converting a webpage screenshot into a structured webpage represented as HTML code [43]. However, no LLM-based methods are currently capable of generating general structured layouts. The conversion of general structured layouts into textual representations for LLMs remains unexplored. The intermediate representation proposed in Parse-Then-Place [42] can describe structured layouts in text. However, This representation is primarily employed to extract information from input text and lacks details to represent a complete layout. An additional Transformer decoder is required to transform the intermediate representation into specific layouts.

Learning-based structure generation. Several works have achieved structured data generation. Most of them adopted RvNN [24] for their tasks. Li et al. [25] presented GRASS for structured 3D shape synthesis. Zhu et al. [27] presented SCORES for structured 3D shape composition. In these two works, shape structures are represented as binary trees. StructureNet [26] exploited general trees for structured 3D shape generation. Li et al. [28] presented GRAINS, an RvNN-based method for indoor scene synthesis. As discussed earlier, RvNN has several limitations and cannot be applied to conditional structured layout generation. Li et al. [44] proposed a Transformer-based tree decoder on user interface completion task but did not achieve conditional generation. Compared with these methods, our approach uses a Transformer architecture and succeeds in conditional structured layout generation.

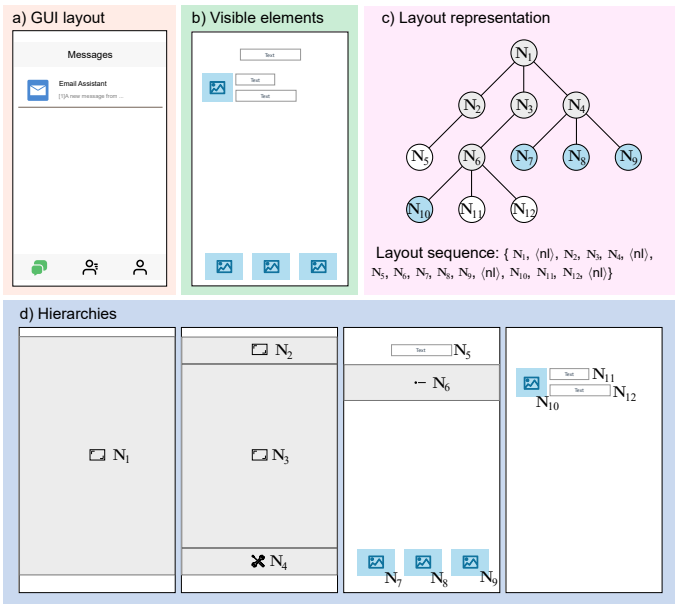


Fig. 2. An illustration of layout representation and serialization. a) An example GUI layout. b) The layout's visible elements. c) The layout structure and the corresponding layout sequence. d) The visualization of the layout hierarchies.

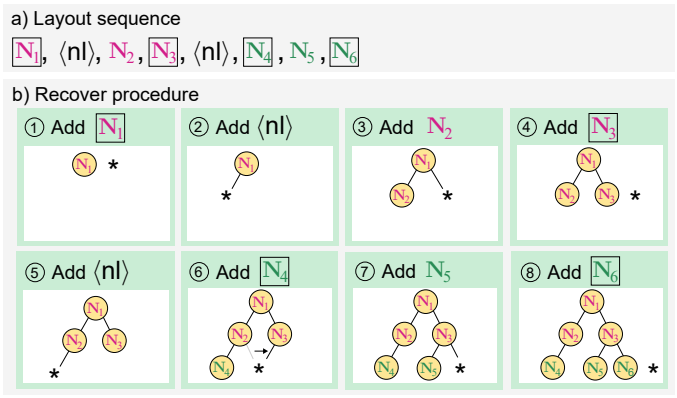


Fig. 3. An example of recovering a layout sequence to a layout tree. a) In the layout sequence, $\mathbf{N}_1$, $\mathbf{N}_2$, and $\mathbf{N}_3$ (in magenta) are internal nodes; $\mathbf{N}_4$, $\mathbf{N}_5$, and $\mathbf{N}_6$ (in green) are leaf nodes; $\mathbf{N}_1$, $\mathbf{N}_3$, $\mathbf{N}_4$, and $\mathbf{N}_6$ (with frames) are the last children of their parents. All this information is stored in the nodes. b) Detailed recovery procedure. The * symbol represents the position where the next predicted element will be placed.

3 Approach

Our approach adopts a Transformer architecture for structured layout generation. To adapt the Transformer architecture to our task, we use a structure serialization scheme to represent structured layouts as sequences. We also introduce a latent space to embed high-level layout structures. This latent space helps disentangle the structural information of layouts to achieve structure-conditioned layout generation. In the following, we first reiterate the layout representation adopted in our approach and introduce our serialization scheme that maps a layout tree to a sequence (Section 3.1). We then explain our model architecture that helps achieve conditional structured layout generation (Section 3.2). Finally, we describe the training objective and training details of our model (Section 3.3).

3.1 Layout representation and serialization

Representation. Most existing data-driven approaches [6]–[14] consider a layout as a set of bounding boxes. This representation is sufficient for unstructured layouts. However, for structured layouts, a more appropriate representation is the layout tree, which has been adopted by existing structured layout datasets [22], [23] and computational frameworks [5], [11], [23], [30]–[32]. Our approach adopts this representation for conditional structured layout generation. Specifically, a structured layout is represented as $\mathcal{T} = \{\mathbf{N}_i\}$, where $\mathcal{T}$ denotes a layout tree and $\mathbf{N}_i$ is a node of this layout tree. $\mathbf{N}_i$ contains the geometric and structural information of this node: $\mathbf{N}_i = [x_i, y_i, w_i, h_i, t_i, \{\mathbf{N}_j\}_i]$. x_i and y_i are the left and top coordinates of the node's bounding box. w_i and h_i are the bounding box's width and height. Following the

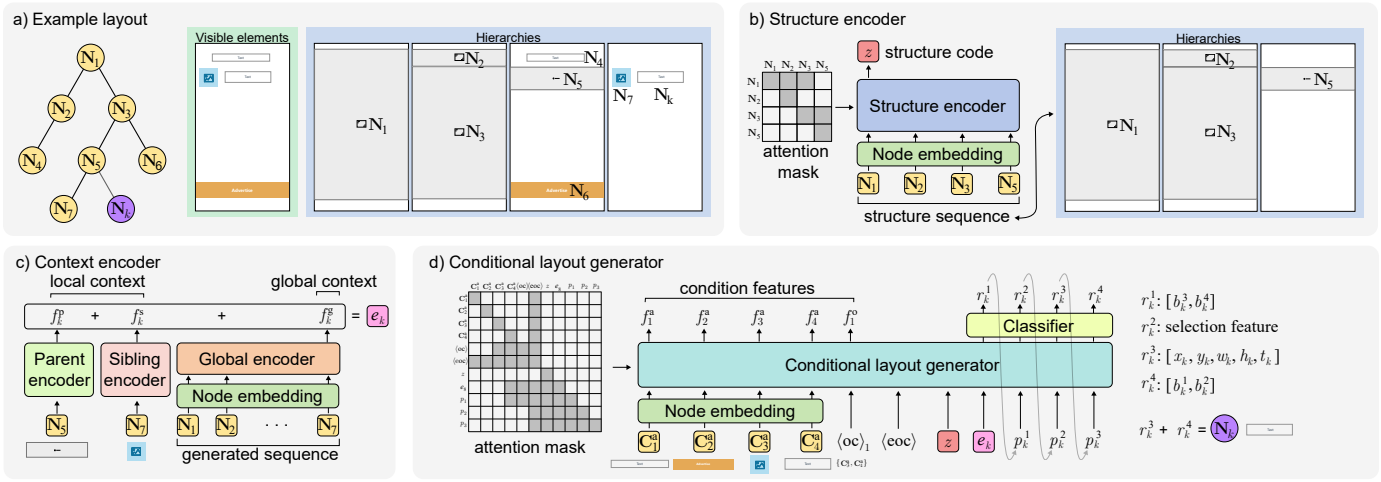


Fig. 4. An overview of our model. a) An example of layout generation conditioned on element types and sizes. The purple node is the target of the current round of node prediction. b) Structure encoder is a VAE that encodes a structure sequence into a latent structure code z. c) Context encoder encodes the context information of $\mathbf{N}_k$(the node model predicts now) to a context code e_k. The context information includes its parent, sibling, and the predicted nodes. d) Conditional layout generator selects the proper condition according to structure z and the context code e_k then predicts attributes of the new element $\mathbf{N}_k$.

previous works [6], [8], [14], [29], these four attributes are quantized. t_i indicates the node's type, which can be a leaf node's semantic label or an internal node's organization type; $\{\mathbf{N}_j\}_i$ is a set containing this node's children. It is empty if this node is a leaf.

Serialization. The layout tree representation cannot be directly used in a Transformer architecture. We adopt the following serialization scheme to represent a tree structure with a sequence. As illustrated in Figure 2, given a layout tree $\mathcal{T} = \{\mathbf{N}_i\}$, we first compose several sub-sequences, each of which consists of the nodes at the same level of the layout tree. In these sub-sequences, a node $\mathbf{N}_i$ does not contain its children set $\{\mathbf{N}_j\}_i$. This avoids the recursive representation problem. To retain the structural information, we add two binary variables b_i^1 and b_i^2, and then $\mathbf{N}_i = [x_i, y_i, w_i, h_i, t_i, b_i^1, b_i^2]$. b_i^1 indicates whether $\mathbf{N}_i$ is a leaf node or an internal node. b_i^2 indicates whether $\mathbf{N}_i$ is the last child node of its parent. With these two binary variables, we can recover the relations among the nodes in different sub-sequences. To represent the structured layout completely, we concatenate these sub-sequences in the order of their levels in the layout tree. To retain the level information, we add an extra token $\langle \mathrm{nl} \rangle$ between the adjacent sub-sequences. Figure 3 shows an example of recovering a layout sequence back to a layout tree.

3.2 Model architecture

Figure 4 illustrates our model architecture. Our model produces a layout sequence autoregressively to generate a structured layout. It has three components: a conditional layout generator, a structure encoder, and a context encoder. The input of the conditional layout generator consists of three parts. The first part is an element condition sequence. It contains the constraints on the elements of the generated layout, e.g., the element types or sizes. The second part is a structure code. It determines the structure of the generated layout. During training, this code is obtained by encoding the high-level structure of a sample with the structure encoder. The third part is a context code. This code changes dynamically in the autoregressive generation procedure. For each generation step, this code is updated by feeding the already generated layout sequence to the context encoder. In the following, we describe our model in detail.

Element condition. The element condition contains the constraints on the elements of the generated layout. Our approach supports two types of conditions: attribute conditions and organization conditions. An attribute condition specifies an element's attributes, i.e., position, size, and type. It is defined as $\mathbf{C}^{\mathrm{a}} = [x, y, w, h, t]$. Note that an attribute condition may only specify a subset of an element's attributes. We thus introduce a mask token $\langle \mathrm{m} \rangle$ to replace the unspecified attributes. For each attribute condition, we convert it into a token via an embedding network, which will be described later. Then, a list of attribute conditions becomes a list of tokens.

An organization condition constrains a set of nodes to be siblings in the generated structured layout. It is difficult to embed such a constraint in a token. Instead, we use a fixed token $\langle \mathrm{oc} \rangle$ to indicate an organization condition and achieve its constraint via an attention mask. This attention mask only allows the condition tokens constrained by this organization condition to pass messages to this token $\langle \mathrm{oc} \rangle$. We will explain the details later when introducing the conditional layout generator. Combining all the attribute and organization conditions results in an element condition sequence. We append an extra $\langle \mathrm{eoc} \rangle$ token to indicate the end of conditions.

Structure encoder. We introduce the structure encoder to disentangle the structural information of a layout from its element placement. For a structured layout represented as a layout tree, its internal nodes indicate how the graphic elements are hierarchically organized, and its leaf nodes reflect the element placement. In addition, a higher-level internal node, i.e., the one closer to the root, contributes more to the structure, and a lower-level internal node influences more on the element placement. We thus extract the structural in-

formation of the layout from its internal nodes. Specifically, we remove the leaf nodes in the layout sequence to obtain a structure sequence. This structure sequence is then fed to the structure encoder to get a structure code. The structure encoder adopts a Transformer-VAE architecture, similar to VTN [6]. To better capture the structural information, we introduce an attention mask in this structure encoder. This attention mask only allows the message of a node to pass to its parent node and itself, implicitly enhancing the influence of the high-level nodes.

Context encoder. Our model produces a layout sequence autoregressively. Each time when generating a new node $\mathbf{N}_k$ of the layout sequence, the previously generated partial sequence is fed to the context encoder to obtain a context code. Since the generated sequence contains the structural information (see Section 3.1), we can determine the relation between $\mathbf{N}_k$ and the previously generated nodes. Based on this structural information, we define local and global contexts. The local context is estimated by feeding the parent node and the most recently generated sibling node of $\mathbf{N}_k$ to two separate FC layers and then adding the extracted features f_k^{p} and f_k^{s}. The global context f_k^{g} is estimated by feeding the generated nodes to a Transformer block. We then add these two context features to obtain the final context code.

Conditional layout generator. The conditional layout generator is a Transformer block. In each autoregressive generation step, it consumes an element condition sequence, a structure code, and a context code to generate a new node $\mathbf{N}_k$. During the generation, the element conditions will be satisfied gradually. As the conditions change, we do not modify the element condition sequence. Instead, we apply an attention mask to indicate the updated conditions. For example, if a condition is already satisfied, the message of the corresponding token cannot pass to other tokens. This attention mask also provides other restrictions. We describe this attention mask as follows: (a) an element condition token only accepts messages from itself and the $\langle\mathrm{eoc}\rangle$ token; (b) an organization condition token accepts messages from itself, the element tokens constrained by this organization condition, and the $\langle\mathrm{eoc}\rangle$ token; (c) the $\langle\mathrm{eoc}\rangle$ token accepts messages from all tokens in the element condition sequence; (d) the structure code token only accepts messages from itself; (e) the context code token accepts messages from unsatisfied condition tokens, the $\langle\mathrm{eoc}\rangle$ token, the structure code token, and itself.

Generating a new node $\mathbf{N}_k$ is achieved in four sub-steps whose outputs are r_k^1, r_k^2, r_k^3, and r_k^4 respectively (Figure 4). The input sequences in these steps have a common part $\{\mathbf{C}_1^{\mathrm{a}}, ..., \mathbf{C}_n^{\mathrm{a}}, \langle\mathrm{oc}\rangle_1, ..., \langle\mathrm{oc}\rangle_m, \langle\mathrm{eoc}\rangle, z, e_k\}$ which contains n attribute conditions, m organization conditions, an $\langle\mathrm{eoc}\rangle$ mark, a structure code z, and a context code e_k. We use $\mathbf{S}_{\mathrm{c}}$ to represent this sequence in the following descriptions. In *Substep 1*, the conditional layout generator takes $\mathbf{S}_{\mathrm{c}}$ as input to generate a feature r_k^1, which is then converted into two binaries b_k^3 and b_k^4 through an MLP classifier. The first binary indicates whether $\mathbf{N}_k$ is an $\langle\mathrm{nl}\rangle$, and the second indicates whether $\mathbf{N}_k$ should satisfy a condition. If $\mathbf{N}_k$ is an $\langle\mathrm{nl}\rangle$, then this round of generation stops. Otherwise, we check if $\mathbf{N}_k$ should satisfy a condition. If yes, we continue *Substep 2*; if no, we go to *Substep 3*.

In *Substep 2*, we convert r_k^1 into a token p_k^1 with an MLP and append it to the input sequence to obtain $\{\mathbf{S}_{\mathrm{c}}, p_k^1\}$. The generator produces a selection feature r_k^2 and calculates its distances to condition features including $f_1^{\mathrm{a}}...f_n^{\mathrm{a}}$ and $f_1^{\mathrm{o}}...f_m^{\mathrm{o}}$. The condition features are the outputs of the conditional layout generator at the corresponding positions of the input conditions. We then aggregate all distances to form a selection categorical probability distribution through a softmax function and finally sample a target condition feature. We force the predicted node in this step to match the selected condition.

In *Substep 3*, we define a new token p_k^2 as the selected condition token or a zero token if no condition is selected. We then append it to the input sequence. Then the input sequence becomes $\{\mathbf{S}_{\mathrm{c}}, p_k^1, p_k^2\}$. The generator takes this sequence and produces a feature r_k^3. Finally we use feature extraction MLPs on r_k^3 to get $[x_k, y_k, w_k, h_k, t_k]$ of $\mathbf{N}_k$.

In *Substep 4*, we convert $[x_k, y_k, w_k, h_k, t_k]$ into a new token p_k^3 and append p_k^3 to the input sequence to obtain $\{\mathbf{S}_{\mathrm{c}}, p_k^1, p_k^2, p_k^3\}$. The generator consumes this sequence to produce a feature r_k^4. We then extract two binaries b_k^1 and b_k^2. b_k^1 decides whether $\mathbf{N}_k$ is a leaf node and b_k^2 indicates whether $\mathbf{N}_k$ is the last child of its parent. After this substep, we finish the generation of the new node $\mathbf{N}_k = [x_k, y_k, w_k, h_k, t_k, b_k^1, b_k^2]$.

Node embedding. In our model, we treat a node of a layout tree as one single token. Such a token is obtained by feeding a node $\mathbf{N} = [x, y, w, h, t, b^1, b^2]$ to an FC layer. This helps reduce the sequence lengths. This node embedding is adopted in the structure encoder and the context encoder. When preparing the element condition sequence, an attribution condition $\mathbf{C}^{\mathrm{a}} = [x, y, w, h, t]$ is converted into a token via this embedding layer.

3.3 Training

We train our model on structured layout datasets and adopt the teacher-forcing training technique. The structure encoder, the context encoder, and the conditional layout generator are trained together. Our generator produces a node $\mathbf{N}_k$ or a next-level token $\langle\mathrm{nl}\rangle$ in each generation step. Since the attributes of this node are all quantized, we adopt cross-entropy losses for training. In addition, since the structure encoder adopts a Transformer-VAE architecture, we include a KL-divergence loss.

4 Experiments

4.1 Setups

Datasets. We adopt two structured layout datasets in our experiments: RICO [22] and WebForest [23]. RICO contains more than 66K mobile GUI layouts with well-defined structures. This dataset has been adopted for the experiments in existing works [6], [8], [10], [14], [29]. However, all these works focus on the element placement only and neglect the layout structures. In contrast, we test our approach on this dataset for structured GUI layout generation. WebForest contains 4.5K webpage layouts with structures. This dataset was proposed to evaluate a computational framework [23] for creating structured layouts. We test our approach on this dataset for structured webpage layout generation. Table 1

Datasets	Element Types
RICO	View, LinearLayout, RelativeLayout, FrameLayout, ViewPager, ListView, GridView, Toolbar, Card, ListItem, Drawer, RecyclerView, WebView, Advertisement, TextButton, ButtonBar, Icon, DatePicker, Modal, Text, Image, Video, Checkbox, Input, BackgroundImage, NumberStepper, MapView, OnOffSwitch, Slider, RadioButton, PagerIndicator, MultiTab, BottomNavigation
WebForest	Root, Container, Image, Text, Button, Graphic, Input

TABLE 1
The element types in two datasets.

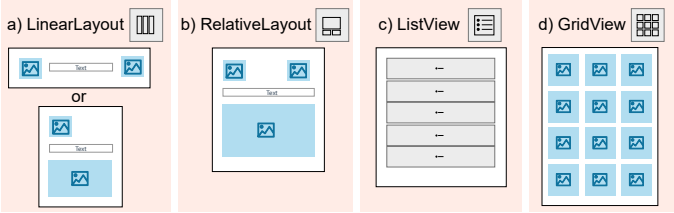


Fig. 5. Examples of four typical internal nodes. a) LinearLayout. All elements are arranged linearly in the horizontal or vertical direction. b) RelativeLayout. Elements can be represented flexibly in a non-linear manner. c) ListView. Elements are arranged in a list and are usually of equal size. d) GridView. Elements are arranged in a grid and are usually of equal size.

shows the element types in these two datasets. Figure 5 shows four typical internal nodes in RICO. Google's Android widget reference[1] gives a more detailed explanation for internal nodes. The training/testing ratio for both datasets is 9:1.

Baselines. To the best of our knowledge, no existing data-driven approach achieves general conditional structured layout generation. READ [37] adopts RvNN for layout generation. However, it considers a layout structure as a binary tree and cannot generate general layout structures. GTLayout [33] also adopts RvNN and can produce layout structures. However, the produced structure is limited to three types of arrangements, i.e., vertical arrangement, horizontal arrangement, and stack arrangement. Parse-Then-Place [42] is another approach capable of generating structured layouts. It takes text descriptions, which contain explicit structural information, as input and has a distinct setting to our approach. On the other hand, the existing approaches can find the arrangement patterns in the datasets. For a structured layout represented as a layout tree, a parent node and its child node often follow certain arrangement patterns, i.e., this parent node often includes its child node. If we treat the internal nodes of layout trees as additional elements and assign them corresponding labels, the existing approaches may produce layout structures implicitly.

Specifically, we achieve structured layout generation in three steps with these approaches. First, we focus on the leaves of layout trees and perform layout generation with these approaches. This is the same as the traditional layout generation task for unstructured layouts. Second, we use the leaves as the condition to predict the internal nodes of layout trees. Note that, in the first and second steps, we train two separate models for each approach. After obtaining the leaves and internal nodes, we determine the tree structure with the following procedure: for each element, its parent is defined as the one that has an internal label, has a larger size than this element, and covers this element most.

We thus compare our approach with the following state-of-the-art layout generation approaches, including LayoutFormer++ [29], BLT [14], LayoutDM [10], and LayoutTransformer [8]. We also tried VTN [6], but its training failed due to memory limit. GTLayout [33] can also produce structured layouts but can not adapt to general layout structures. We compare our approach with GTLayout in its structure setting in a separate experiment (Section 4.3).

Evaluation metrics. Our approach generates structured layouts represented as layout trees. To evaluate the quality of the generated layouts, we adopt two types of metrics, which we term **element metrics** and **structure metrics**. The **element metrics** measures the quality of element arrangements. We adopt the alignment score (**Align**) [29], the overlap score (**Overlap**) [45], the Wasserstein distance for the label distribution (**W Label**) [6], and the Wasserstein distance for the bounding box distribution (**W Box**) [6], as the **element metrics**. Since they are about element arrangements, we use the visible graphic elements in layouts for their computations. The **structure metrics** measures the quality of layout structures. We find it difficult to measure the quality of global structures. However, we observe that the quality of local structures reflects the quality of global structures. We thus define the following metrics as the **structure metrics**.

S-Align. This metric measures the alignment of local structures. Here, a local structure can be a set of sibling nodes. Given a structured layout, we first collect all its local structures, i.e., all sibling sets. We then compute the alignment score [29] of each sibling set and define **S-Align** of a structured layout as the average of these scores.

S-Overlap. This metric measures the overlap of local structures. A local structure is also defined as a sibling set. Then **S-Overlap** of a structured layout is defined as the average overlap score [45] of its sibling sets.

S-Inclusion. This metric measures the inclusion of local structures. Here, a local structure is defined as a pair of a parent node and a child node. Since a parent node often serves as the container of its child nodes, the child nodes should be included in the parent node. We thus define the inclusion score of a parent-child pair as the intersection of the parent node and child node over the child node. Then, **S-Inclusion** of a structured layout is defined as the average inclusion score of all its parent-child pairs.

W S-Label and **W S-Box**. These two metrics reflect whether the generated layout structures are similar to those in the datasets. We use parent-child pairs as local structures to define these two metrics. Specifically, we consider the

1. developer.android.com/reference/android/widget/package-summary

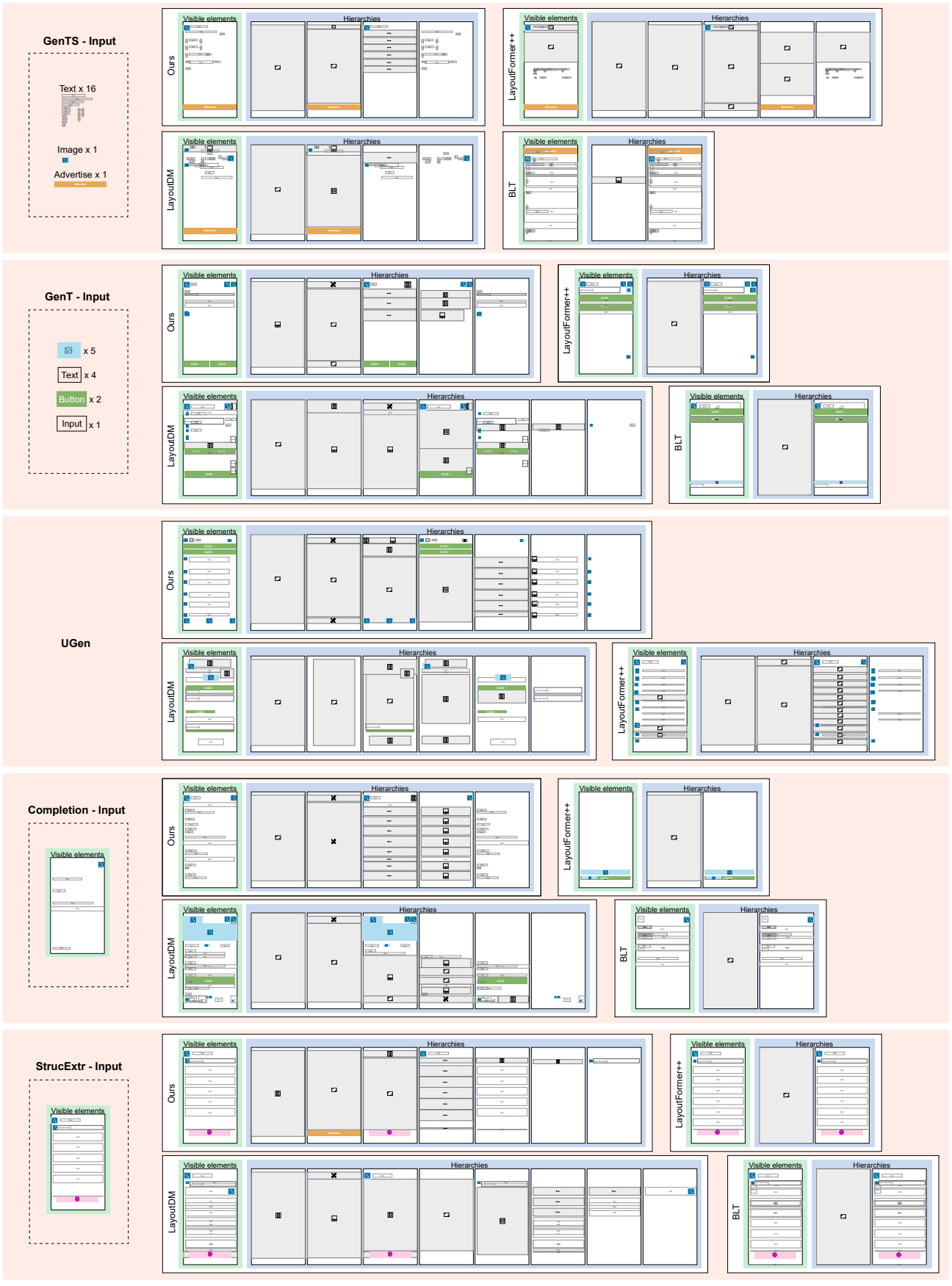


Fig. 6. Representative results generated by the compared approaches in the tasks of **GenTS**, **GenT**, **UGen**, **Completion**, and **StructExtr** on RICO. For each sequence of results, the first one is the visible elements, and the subsequent ones indicate the hierarchies. More results are included in the supplemental material.

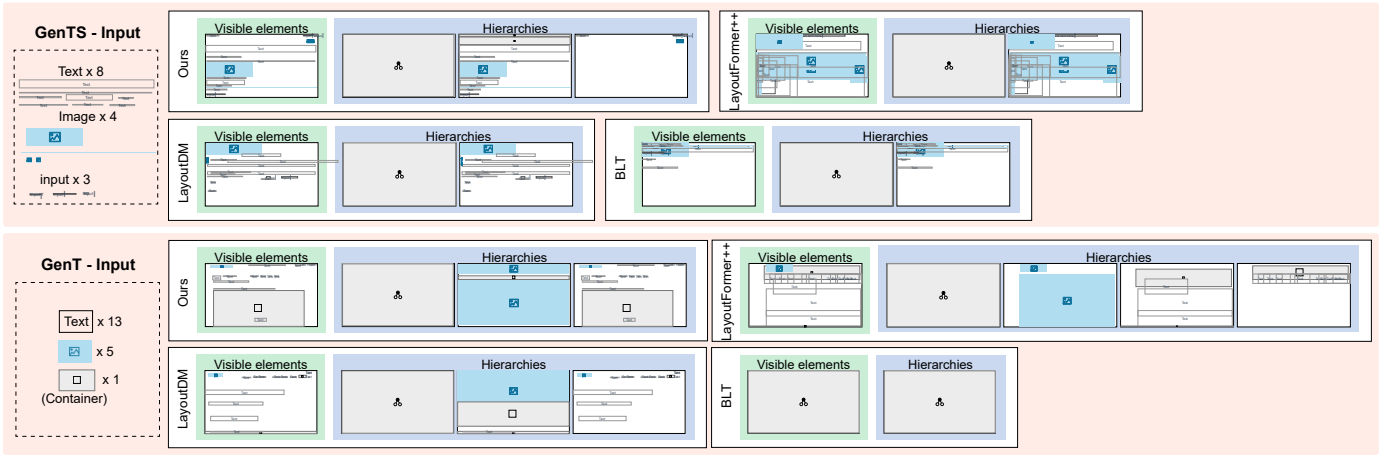


Fig. 7. Representative results generated by the compared approaches in the tasks of **GenTS** and **GenT** on WebForest.

Fig. 8. Results of our approach in the task of **StructTran**.

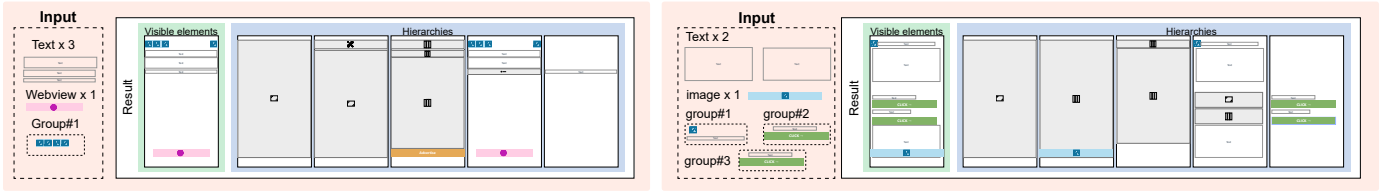


Fig. 9. Results of our approach in the task of **GenO**.

		Structure metrics					Element metrics			
Tasks	Approaches	W S-Label ↓	W S-Box ↓	S-Inclusion ↑	S-Align ↓	S-Overlap ↓	Align ↓	Overlap ↓	W Label ↓	W Box ↓
GenTS	Ours	**4.90**	**0.033**	0.927	0.0032	**0.049**	0.0019	**0.024**	0.083	0.026
	LayoutFormer++	54.12	0.088	0.888	0.0031	0.067	0.0014	0.032	0.272	0.053
	BLT	16.70	0.139	0.318	**0.0017**	0.207	**0.0007**	0.114	0.504	0.121
	LayoutDM	7.57	0.078	**0.939**	0.0042	0.114	0.0008	0.049	**0.025**	**0.013**
GenT	Ours	**4.63**	**0.038**	0.942	0.0023	0.052	0.0034	**0.022**	0.164	0.040
	LayoutFormer++	57.37	0.112	0.868	0.0022	**0.048**	0.0006	0.017	0.351	0.071
	BLT	13.87	0.198	0.358	**0.0005**	0.240	**0.0004**	0.141	0.402	0.232
	LayoutDM	6.01	0.085	**0.943**	0.0042	0.117	0.0009	0.047	**0.05**	**0.012**
UGen	Ours	11.10	**0.052**	0.908	**0.0018**	0.057	**0.0003**	0.019	0.764	0.042
	LayoutFormer++	58.18	0.134	0.890	0.0020	**0.038**	0.0005	**0.009**	0.234	0.079
	LayoutDM	**6.77**	0.079	**0.941**	0.0049	0.123	0.0009	0.051	**0.089**	**0.014**
Completion	Ours	**3.62**	**0.053**	0.863	0.0021	**0.026**	0.0070	**0.006**	0.205	0.023
	LayoutFormer++	38.73	0.096	**0.939**	0.0066	0.072	0.0028	0.033	0.592	0.060
	BLT	18.89	0.123	0.322	**0.0016**	0.213	0.0008	0.112	0.561	0.135
	LayoutDM	5.08	0.094	0.938	0.0040	0.115	**0.0007**	0.040	**0.091**	**0.017**
StructExtr	Ours	**2.35**	**0.041**	0.801	0.0047	0.050	-	-	-	-
	LayoutFormer++	57.73	0.114	0.836	0.0038	**0.046**	-	-	-	-
	BLT	17.41	0.120	0.311	**0.0030**	0.167	-	-	-	-
	LayoutDM	4.82	0.084	**0.942**	0.0041	0.105	-	-	-	-

TABLE 2
Quantitative comparisons of **GenT**, **GenTS**, **UGen**, **Completion** and **StructExtr** on RICO.

		Structure metrics					Element metrics			
Tasks	Approaches	W S-Label ↓	W S-Box ↓	S-Inclusion ↑	S-Align ↓	S-Overlap ↓	Align ↓	Overlap ↓	W Label ↓	W Box ↓
GenTS	Ours	**0.84**	**0.056**	0.847	0.0048	**0.052**	0.0022	**0.029**	**0.018**	0.037
	LayoutFormer++	1.09	0.095	**0.989**	0.0118	0.067	0.0053	**0.029**	0.084	0.027
	BLT	2.95	0.232	0.100	**0.0020**	0.067	**0.0006**	0.043	0.144	0.096
	LayoutDM	7.57	0.078	0.939	0.0042	0.114	0.0008	0.049	0.025	**0.013**
GenT	Ours	**0.92**	**0.047**	0.862	0.0050	0.048	0.0020	0.027	**0.025**	0.047
	LayoutFormer++	1.39	0.098	**0.988**	0.0092	0.058	0.0022	**0.016**	0.124	0.023
	BLT	3.24	0.239	0.085	**0.0009**	**0.038**	**0.0007**	0.026	0.093	0.13
	LayoutDM	6.01	0.085	0.943	0.0042	0.117	0.0009	0.047	0.053	**0.012**
UGen	Ours	**0.74**	**0.056**	0.827	**0.0043**	0.058	0.0010	0.017	0.110	0.035
	LayoutFormer++	1.04	0.108	**0.993**	0.0074	**0.042**	0.0018	**0.005**	0.141	0.029
	LayoutDM	6.77	0.079	0.941	0.0049	0.123	**0.0009**	0.051	**0.089**	**0.014**
Completion	Ours	**1.18**	**0.044**	0.783	0.0106	**0.012**	0.0024	**0.002**	**0.018**	0.027
	LayoutFormer++	1.56	0.111	**0.992**	0.0105	0.046	0.0031	0.010	0.077	0.039
	BLT	3.55	0.254	0.065	**0.0014**	0.029	0.0009	0.024	0.333	0.140
	LayoutDM	5.08	0.094	0.938	0.0040	0.115	**0.0007**	0.040	0.091	**0.017**
StructExtr	Ours	2.34	0.106	0.798	0.0105	**0.007**	-	-	-	-
	LayoutFormer++	**1.14**	**0.092**	**0.991**	0.0079	0.039	-	-	-	-
	BLT	2.59	0.247	0.083	**0.0021**	0.029	-	-	-	-
	LayoutDM	2.65	0.145	0.937	0.0029	0.044	-	-	-	-

TABLE 3
Quantitative comparisons of **GenT**, **GenTS**, **UGen**, **Completion** and **StructExtr** on WebForest.

distribution of label pairs (**W S-Label**) and bounding box pairs (**W S-Box**) to compute the Wasserstein distance [6] between real and generated layouts.

Implementation details. We implement our approach by PyTorch. The model is trained using the Adam optimizer [46] with NVIDIA RTX 3090 GPUs. The Transformer blocks have 512 embedding dimensions and 2048 feed-forward dimensions. For the conditional layout generator, the Transformer block has 6 layers. For the structure encoder and context encoder, the Transformer blocks have 4 layers.

4.2 Conditional structured layout generation

Tasks. Our approach supports the following existing layout generation tasks: generation conditioned on element types (**GenT**); generation conditioned on element types and sizes (**GenTS**); completion from given elements (**Completion**); and unconstrained generation (**UGen**). Since our approach considers layout structures explicitly, it also supports the following new tasks: structure extraction from given elements (**StructExtr**); layout generation conditioned on element organizations (**GenO**); and structure transfer between structured layouts (**StructTran**). Below, we briefly describe how our approach achieves these tasks.

GenT, **GenTS**, **Completion**, and **UGen**. These tasks do not require any structural conditions. When generating a structured layout, the element condition sequence only includes element attribute conditions. The structure code is randomly sampled from the constructed structure space. The randomly sampled structure codes ensure structure diversity in the generated layouts. On the other hand, the structured code can also be predetermined to achieve more controllable structured layout generation.

StructExtr. This task takes a complete set of elements as conditions. To extract its structure, we randomly sample a structure code for the structured layout generation. To obtain a more reasonable structure, we can estimate a new structure code from the generated structure and use this new code for a new round of structure generation. This procedure may iterate until a better structure is extracted.

GenO. This task requires element organizations as conditions. When generating a structured layout, the element condition sequence includes element attribute conditions

		Structure metrics					Element metrics			
Tasks	Approaches	**W S-Label** ↓	**W S-Box** ↓	**S-Inclusion** ↑	**S-Align** ↓	**S-Overlap** ↓	**Align** ↓	**Overlap** ↓	**W Label** ↓	**W Box** ↓
GenT	Ours	0.69	**0.018**	**0.943**	**0.0051**	0.189	**0.0015**	0.065	0.345	**0.027**
	GTLayout	**0.56**	0.057	0.938	0.0083	**0.152**	0.0018	**0.028**	**0.170**	0.045
GenTS	Ours	0.70	**0.017**	0.924	**0.0057**	0.181	**0.0011**	0.066	0.348	**0.022**
	GTLayout	**0.62**	0.071	**0.939**	0.0085	**0.146**	0.0020	**0.027**	**0.110**	0.049

TABLE 4
Quantitative comparisons between our approach and GTLayout in the tasks of **GenT** and **GenTS**.

and organization conditions. The structure code is also randomly sampled.

StructTran. This task transfers the structure of an existing structured layout to an unstructured layout. It is achieved by using the structured layout's structure code and the unstructured layout's elements as conditions.

Evaluation settings. The existing approaches basically support **GenT**, **GenTS**, **Completion**, and **UGen**. In addition, we notice that the second step when using the baselines is similar to **StructExtr**. We thus compare our approach with the baselines on these tasks. Specifically, we compare our approach with (a) LayoutFormer++ [29], BLT [14], and LayoutDM [10] on **GenT**, **GenTS**, **Completion**, and **StructExtr**; (b) LayoutFormer++ [29], and LayoutDM [10] on **UGen**. In these tasks, we randomly select 1000 layout samples from the testing set and mask specific attributes for each sample to create condition settings. For example, to create the condition settings for **GenT**, we mask all the attributes except for the element types for the selected 1000 layout samples. For each condition setting and each compared approach, we generate a layout. The generated layouts are used for quantitative and qualitative comparisons to show the effectiveness of our approach. Since the baselines can not take structures as input, our approach takes randomly sampled structure codes in these tasks for a fair comparison.

Since no existing approach supports **GenO**, we demonstrate qualitative results in this task. [23] is a computational approach supporting **StructTran**. However, this approach adopts different settings from ours and is not open-source.

Results and discussion. Tables 2 and 3 show the quantitative results of the compared approaches in the tasks of **GenT**, **GenTS**, **UGen**, **Completion**, and **StructExtr**. Our approach achieves the best performance in most **structure metrics**. The most representative metrics are **W S-Label** and **W S-Box**. These two metrics measure the quality of the generated structures, which is the target of our paper. Our approach achieves the best scores in almost all tasks, confirming the rationality of our generated structures. For **S-Align** and **S-Overlap**, our approach achieves good **S-Overlap** scores while keeping **S-Align** scores comparative to other baselines. this also confirms the high quality of our generated layouts. The baselines achieve better scores in **S-Inclusion**. This is reasonable since their layout structures are extracted by maximizing the inclusion scores. This metric would be more meaningful if all the compared approaches could produce structures explicitly.

The baselines generate structured layouts with three steps and the visible elements are produced in the first step. In contrast, our approach produces layouts' visible elements and structures simultaneously. However, our approach still produces layouts with comparable alignment quality of visible elements. This is confirmed by the **element metrics**.

Figure 6 shows the qualitative results of the compared approaches in the task of **GenTS**, **GenT**, **UGen**, **Completion**, and **StructExtr** on RICO. For each task, we provide one input setting and the layouts produced by the compared approaches. The results on WebForest have lower qualities since this dataset does not contain sufficient samples. We do not include these results in the paper. More results are included in the supplemental material. These visual results further confirm the superiority of our approach. Our approach can generate samples with high-quality hierarchical structures and visual elements. In contrast, the layout hierarchies generated by LayoutDM, LayoutFormer++, and BLT are less realistic or meaningful. For example, In the task of **StructExtr**, our approach generates reasonable hierarchies that appropriately organize the visual elements; LayoutDM produces specious hierarchies that are not functionally justifiable; LayoutFormer++ and BLT even fail to generate any multi-level hierarchies. Figure 7 shows the qualitative results of the compared approaches in the task of **GenTS** and **GenT** on WebForest. Due to the limited size of the dataset, the results produced by the compared methods are not satisfactory. Nevertheless, our approach still demonstrates superior performance over the baselines in terms of both visible elements and structure.

Figure 8 and Figure 9 show the results of our approach in the task of **StructTran** and **GenO**. **GenO** allows people to give high-level organizations of elements. Our model can find appropriate structures for such organizations. This function may help with structured layout design. **StructTran** helps quickly design structured layouts from existing ones or create a set of structured layouts with similar styles. Figure 8 demonstrates that the layouts generated by our approach faithfully capture the structure of the reference layouts. These functions further increase the application scenarios of our approach.

Our approach has the same training and inference time complexity as the other Transformer-based approaches, although our approach includes more attention mask operations. On average, our approach can generate a structured layout in 1.51 seconds. In comparison, to generate a structured layout, LayoutFormer++ costs 1.74 seconds, LayoutDM costs 0.03 seconds, and LayoutTransformer costs 0.11 seconds. As a future work, our approach can be further accelerated by parallelization, which is adopted by LayoutTransformer.

4.3 Comparison with GTLayout

GTLayout [33] adopts RvNN for structured layout generation. However, it primarily focuses on layout construction

Fig. 10. Representative results of our approach and GTLayout in the tasks of **GenT** and **GenTS**.

and interpolation without extensively addressing conditional layout generation. We first extend GTLayout to support conditional generation and then compare our approach with GTLayout under various conditional settings.

The GTLayout pipeline consists of a VAE encoder and decoder, which encodes structured layouts into latent space vectors and generates layouts from vectors sampled from this latent space. Inspired by CVAE [47], we introduce a conditional encoder before the original VAE decoder. This encoder is a single-layer transformer that integrates the latent code and conditional inputs into a feature vector, which then serves as the input to the VAE decoder. GTLayout uses a dataset that differs from ours and includes distinct internal node types, i.e., vertical arrangement, horizontal arrangement, and stack arrangement. The design of these internal node types is tied to GTLayout's model architecture, making it challenging for the GTLayout pipeline to adapt to our dataset. We thus compare our method with GTLayout using their RICO dataset.

Results and discussion. Table 4 shows the quantitative results. These quantitative results demonstrate that our method performs comparatively to GTLayout, confirming the generation ability of both methods. GTLayout even achieves better **Overlap**, **S-Overlap**, **W Label**, and **W S-Label** scores. Although GTLayout is effective in generating high-quality structured layouts, it often fails to satisfy the input conditions and deviates from the goal of conditional generation. Figure 10 displays results produced by our method and GTLayout. GTLayout struggles with correctly handling the number and size of input elements, often failing to predict the exact number of nodes and generating unnecessary elements. GTLayout employs relative bounding box representations, which prevent it from accommodating global size constraints. Furthermore, GTLayout is limited to three specific internal nodes: vertical arrangement, horizontal arrangement, and stack arrangement. In contrast, our method can handle datasets with custom internal nodes.

4.4 Ablation study

We conduct an ablation study to demonstrate the necessity of the local context and the global context. In this study, we remove the local context and the global context from the complete model, respectively, and adopt the **GenT** task for the evaluation. Table 5 shows the quantitative results. The results confirm that the complete model outperforms the other configurations. Without the global context, the model achieves a better **W S-Box** score. It is reasonable since the model predicts the current node based on the parent and the sibling, which is learned from the dataset's distribution of parent-child pairs. Without the local context, the model achieves a better **Align** score since this score is computed based on global alignment, which may be affected by the local context. Figure 11 shows some qualitative results. These results further confirm that the complete model produces layouts with higher quality. Without the local context, the produced layout has defects in local element arrangement; without the global context, the global arrangement of the elements in the produced layout is severely affected.

We also conduct an experiment on the conditional layout generator. In this experiment, we remove the attention mask, and the model fails to produce reasonable results. For

Configurations	Structure metrics					Element metrics			
	W S-Label ↓	W S-Box ↓	S-Inclusion ↑	S-Align ↓	S-Overlap ↓	Align ↓	Overlap ↓	W Label ↓	W Box ↓
Complete model	**3.33**	0.064	0.938	**0.0017**	**0.056**	0.0037	**0.020**	**0.091**	**0.045**
W/o local context	3.96	0.063	**0.943**	0.0031	0.064	**0.0033**	0.026	0.115	**0.045**
W/o global context	6.31	**0.032**	0.923	0.0031	0.066	0.0037	0.040	0.097	0.069

TABLE 5
Quantitative results of the ablation study.

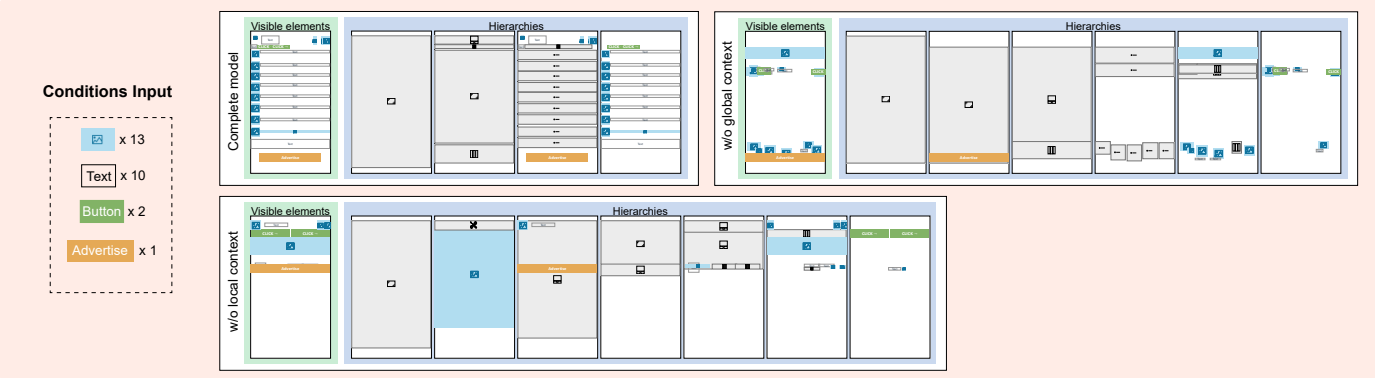


Fig. 11. Representative results of the ablation study.

the other component of the conditional layout generator, e.g., the structure code z and the organization token $\langle \text{oc} \rangle$, can not be removed since they are explicitly designed for the specific functions of our approach. For example, without the structure code, our approach can not achieve the **StructTran** task; without the organization token, our approach can not support organization conditions.

4.5 Structure for layout optimization

In this section, we demonstrate that the structured layouts generated by our approach can be used for layout optimization, which is crucial for adapting graphic layouts to different display configurations.

The structured layouts generated by our approach contain detailed hierarchical organizations of visible graphic elements. Such hierarchical organizations can not be used for layout geometry optimization directly. However, they can help determine the alignment relations among the elements. In a structured layout, an internal node specifies the arrangement types of its children nodes. For example, if an internal node has the type of linear arrangement, then its children elements should be arranged in a sequence. The arrangement direction and order can be determined by the initial geometry of the children elements. With this information, we can extract the precise alignment relations among the elements and use the existing approaches [1] for layout optimization.

The structural information also enables easy manipulation of graphic layouts, which is important in graphic layout design. Figure 12 demonstrates layout manipulation by editing visible and internal elements. For example, dragging a visible element can easily modify its relative position in the layout; resizing an internal element helps automatically re-arrange its children elements, adapting to different display configurations; modifying the visibility of an internal element eases the manipulation of the layout containing overlapped elements.

5 Conclusion

In this paper, we have presented *StructLayoutFormer*, a novel Transformer-based approach for conditional structured layout generation. We use a structure serialization scheme to represent structured layouts as sequences. We also disentangle the structural information of layouts from element arrangements, thus achieving better control of layout structures. The experiments have confirmed that our approach is more effective in generating structured layouts with conditions compared with the baselines. We have also demonstrated that our approach can achieve layout structure extraction and transfer and discussed the potential applications. To the best of our knowledge, our approach is the first data-driven approach that achieves conditional structured layout generation.

Our approach still has limitations. Given a specific set of conditions for layout generation, the randomly sampled structure code may not always be suitable. For example, with a structure code corresponding to a simple structure and a set of conditions specifying a large number of elements, the generated layout may contain an unexpected element overlay. Although we propose an iterative generation strategy for this problem, it is still necessary to devise a mapping between the conditions and the structures.

In future work, we plan to extend the model for other structured data generation tasks, e.g., 3D shapes and indoor scenes. Our approach requires layouts to be completely structured. It would be meaningful to learn structural patterns from partially structured layouts. It would also be interesting to exploit Diffusion models for structured layout generation.

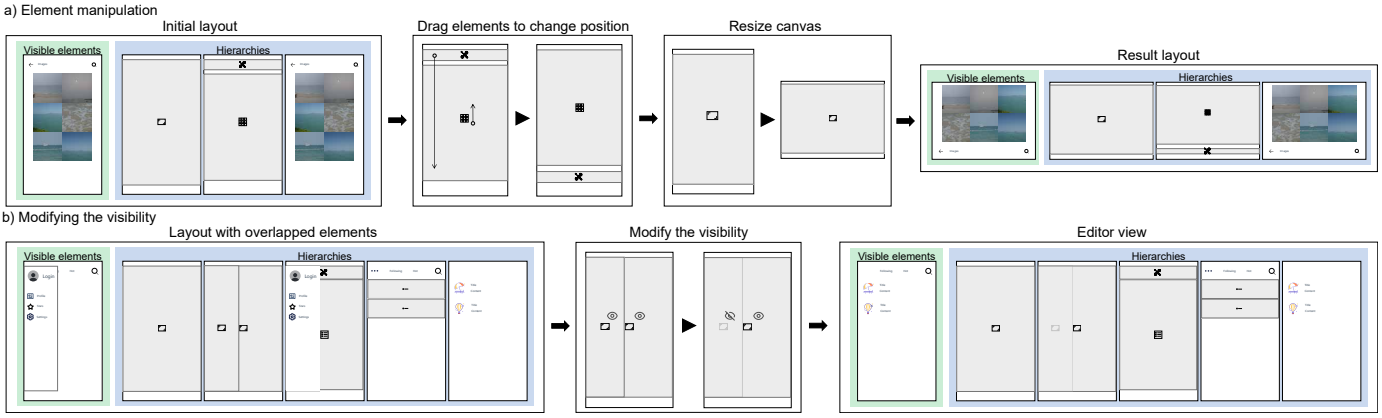


Fig. 12. The structural information enables easy manipulation of graphic layouts. a) Dragging an element can easily modify its relative position in the layout; resizing an element helps automatically re-arrange its children elements, adapting to different display configurations. b) Modifying the visibility of an internal element eases the manipulation of the layout containing overlapped elements.

ACKNOWLEDGEMENTS

We thank the reviewers for their insightful comments. This work was partially supported by grants from NSFC (62472287, 62072316, U21B2023), Guangdong Basic and Applied Basic Research Foundation (2023A1515011297, 2023B1515120026), DEGP Innovation Team (2022KCXTD025), and Scientific Development Funds from Shenzhen University.

REFERENCES

[1] P. Xu, G. Yan, H. Fu, T. Igarashi, C.-L. Tai, and H. Huang, "Global beautification of 2d and 3d layouts with interactive ambiguity resolution," *IEEE transactions on visualization and computer graphics*, vol. 27, no. 4, pp. 2355–2368, 2019.

[2] C. Zeidler, C. Lutteroth, W. Sturzlinger, and G. Weber, "The auckland layout editor: An improved gui layout specification process," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*, 2013, pp. 343–352.

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

[4] A. Swearngin, C. Wang, A. Oleson, J. Fogarty, and A. J. Ko, "Scout: Rapid exploration of interface layout alternatives through high-level design constraints," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[5] P. Xu, Y. Li, Z. Yang, W. Shi, H. Fu, and H. Huang, "Hierarchical layout blending with recursive optimal correspondence," *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)*, vol. 41, no. 6, pp. 249:1–249:15, 2022.

[6] 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*, 2021, pp. 13 642–13 652.

[7] S. Chai, L. Zhuang, and F. Yan, "Layoutdm: Transformer-based diffusion model for layout generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 18 349–18 358.

[8] 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*, 2021, pp. 1004–1014.

[9] M. Hui, Z. Zhang, X. Zhang, W. Xie, Y. Wang, and Y. Lu, "Unifying layout generation with a decoupled diffusion model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 1942–1951.

[10] N. Inoue, K. Kikuchi, E. Simo-Serra, M. Otani, and K. Yamaguchi, "LayoutDM: Discrete Diffusion Model for Controllable Layout Generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 10 167–10 176.

[11] 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.

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

[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*, 2021, pp. 88–96.

[14] X. Kong, L. Jiang, H. Chang, H. Zhang, Y. Hao, H. Gong, and I. Essa, "Blt: bidirectional layout transformer for controllable layout generation," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVII*. Springer, 2022, pp. 474–490.

[15] X. Zheng, X. Qiao, Y. Cao, and R. W. Lau, "Content-aware generative modeling of graphic design layouts," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–15, 2019.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[17] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations*, 2013.

[18] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

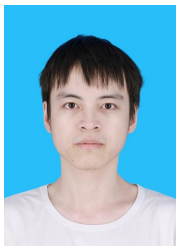
[20] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[21] Y. Jiang, R. Du, C. Lutteroth, and W. Stuerzlinger, "Orc layout: Adaptive gui layout with or-constraints," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–12.

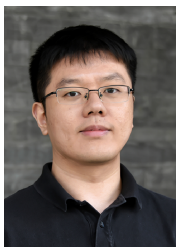
[22] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, 2017, pp. 845–854.

[23] K. Kikuchi, M. Otani, K. Yamaguchi, and E. Simo-Serra, "Modeling visual containment for web page layout optimization," in *Computer Graphics Forum*, vol. 40, no. 7. Wiley Online Library, 2021, pp. 33–44.

[24] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 129–136.
[25] J. Li, K. Xu, S. Chaudhuri, E. Yumer, H. Zhang, and L. Guibas, "Grass: Generative recursive autoencoders for shape structures," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–14, 2017.
[26] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. Mitra, and L. J. Guibas, "Structurenet: Hierarchical graph networks for 3d shape generation," *arXiv preprint arXiv:1908.00575*, 2019.
[27] C. Zhu, K. Xu, S. Chaudhuri, R. Yi, and H. Zhang, "Scores: Shape composition with recursive substructure priors," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pp. 1–14, 2018.
[28] M. Li, A. G. Patil, K. Xu, S. Chaudhuri, O. Khan, A. Shamir, C. Tu, B. Chen, D. Cohen-Or, and H. Zhang, "Grains: Generative recursive autoencoders for indoor scenes," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 2, pp. 1–16, 2019.
[29] Z. Jiang, J. Guo, S. Sun, H. Deng, Z. Wu, V. Mijovic, Z. J. Yang, J.-G. Lou, and D. Zhang, "Layoutformer++: Conditional graphic layout generation via constraint serialization and decoding space restriction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 18 403–18 412.
[30] M. Dixon, D. Leventhal, and J. Fogarty, "Content and hierarchy in pixel-based methods for reverse engineering interface structure," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 969–978.
[31] Y. Jiang, W. Stuerzlinger, and C. Lutteroth, "Reverseorc: Reverse engineering of resizable user interface layouts with or-constraints," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–18.
[32] P. O'Donovan, A. Agarwala, and A. Hertzmann, "Learning layouts for single-page graphic designs," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 8, pp. 1200–1213, 2014.
[33] P. Xu, W. Shi, X. Hu, H. Fu, and H. Huang, "Gtlayout: Learning general trees for structured grid layout generation," in *International Conference on Computational Visual Media*. Springer, 2024, pp. 131–153.
[34] J. Li, J. Yang, A. Hertzmann, J. Zhang, and T. Xu, "Layoutgan: Generating graphic layouts with wireframe discriminators," *arXiv preprint arXiv:1901.06767*, 2019.
[35] 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*. Springer, 2020, pp. 491–506.
[36] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.
[37] 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*, 2020, pp. 544–545.
[38] W. Feng, W. Zhu, T.-j. Fu, V. Jampani, A. Akula, X. He, S. Basu, X. E. Wang, and W. Y. Wang, "Layoutgpt: Compositional visual planning and generation with large language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 18 225–18 250, 2023.
[39] Z. Tang, C. Wu, J. Li, and N. Duan, "Layoutnuwa: Revealing the hidden layout expertise of large language models," *arXiv preprint arXiv:2309.09506*, 2023.
[40] T. Yang, Y. Luo, Z. Qi, Y. Wu, Y. Shan, and C. W. Chen, "Posterllava: Constructing a unified multi-modal layout generator with llm," *arXiv preprint arXiv:2406.02884*, 2024.
[41] Y. Yang, J. Lu, Z. Zhao, Z. Luo, J. J. Yu, V. Sanchez, and F. Zheng, "Llplace: The 3d indoor scene layout generation and editing via large language model," *arXiv preprint arXiv:2406.03866*, 2024.
[42] J. Lin, J. Guo, S. Sun, W. Xu, T. Liu, J.-G. Lou, and D. Zhang, "A parse-then-place approach for generating graphic layouts from textual descriptions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 23 622–23 631.
[43] H. Laurençon, L. Tronchon, and V. Sanh, "Unlocking the conversion of web screenshots into html code with the websight dataset," *arXiv preprint arXiv:2403.09029*, 2024.
[44] Y. Li, J. Amelot, X. Zhou, S. Bengio, and S. Si, "Auto completion of user interface layout design using transformer-based tree decoders," *arXiv preprint arXiv:2001.05308*, 2020.
[45] 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*, vol. 27, no. 10, pp. 4039–4048, 2020.
[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980
[47] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

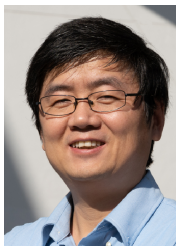


Xin Hu is an M.Sc. candidate in the Visual Computing Research Center at Shenzhen University, China. He received his bachelor's degree in computer science from Jilin University in 2022. His research interest is computer graphics.



Pengfei Xu is an Associate Professor of the College of Computer Science and Software Engineering at Shenzhen University. He received his Bachelor's degree in Math from Zhejiang University, China, in 2009 and his Ph.D. in Computer Science from the Hong Kong University of Science and Technology in 2015. His primary research lies in Human-Computer Interaction and Computer Graphics.

Jin Zhou is a Master's candidate at the Visual Computing Center, Shenzhen University, specializing in computer graphics. He earned his Bachelor's degree from Shanghai Maritime University in 2023. His research interest is computer graphics.



Hongbo Fu received a BS degree in information sciences from Peking University, China, in 2002 and a PhD degree in computer science from the Hong Kong University of Science and Technology in 2007. He is a Full Professor of the Division of Emerging Interdisciplinary Areas at the Hong Kong University of Science and Technology. His primary research interests fall in the fields of computer graphics and human-computer interaction. He has served as an Associate Editor of The Visual Computer, Computers & Graphics, and Computer Graphics Forum.

Hui Huang is a Distinguished TFA Professor at Shenzhen University, where she directs the Visual Computing Research Center. She received her Ph.D. degree in applied math from The University of British Columbia in 2008. Her research interests span computer graphics, vision, and visualization. She is currently a Senior Member of IEEE/ACM/CSIG, a Distinguished Member of CCF, and is on the editorial boards of ACM TOG and IEEE TVCG.