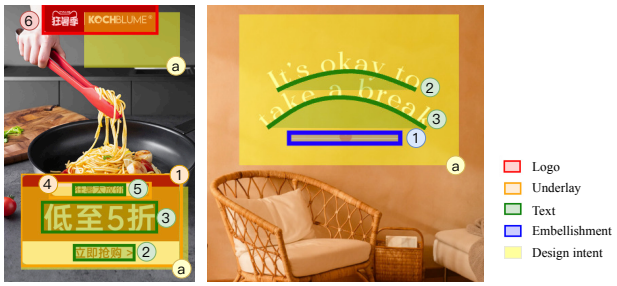
PosterO: Structuring Layout Trees to Enable Language Models in Generalized Content-Aware Layout Generation

Supplementary Material

This supplementary material provides additional information about the proposed approach, **PosterO**, and the new dataset, **PStylish7**. Sec. A presents examples of layout trees T and prompts P, Sec. B reports additional experimental results, and Sec. C provides more details and statistics about the PStylish7 dataset.

A. Examples of Layout Trees and Prompts

Layout trees T. Fig. A presents examples of layout trees structured as depicted in Sec. 3.1. In (a), two distinct areas ⓐ are detected for its design intent, and the double-nested underlay elements (*i.e.*, ①-⑤) contribute to a layout tree depth of three, based on the proposed hierarchical node representation. In (b), the curved texts (*i.e.*, ②,③) are outlined using `<path>`, following universal element vectorization.



(a) (b)

Layout tree of (a)

```
<svg width="513" height="750" xmlns="http://www.w3.org/2000/svg">
  ⓐ <polygon id="available_area" points="47,491 492,491 492,714 47,714" />
  ⓐ <polygon id="available_area" points="215,22 470,22 470,171 215,171" />
    <rect id="canvas_0" x="0" y="0" width="513" height="750" />
    <svg x="47" y="456">
      ① <rect id="underlay_1" x="0" y="0" width="427" height="248" />
      ② <rect id="text_2" x="142" y="193" width="159" height="39" />
      ③ <rect id="text_3" x="57" y="77" width="304" height="80" />
        <svg x="74" y="22">
          ④ <rect id="underlay_4" x="0" y="0" width="270" height="43" />
          ⑤ <rect id="text_5" x="66" y="8" width="136" height="27" />
        </svg>
    </svg>
  ⑥ <rect id="logo_6" x="106" y="0" width="303" height="79" />
</svg>
```

Layout tree of (b)

```
<svg width="513" height="513" xmlns="http://www.w3.org/2000/svg">
  ⓐ <polygon id="available_area" points="60,19 465,19 465,297 60,297" />
    <rect id="canvas_0" x="0" y="0" width="513" height="513" />
  ① <rect id="embellishment_1" x="152" y="237" width="205" height="20" />
  ② <path id="text-curved_2" d="M 130 159 C 156 146 198 127 253 124
                                C 300 121 342 139 386 158" />
  ③ <path id="text-curved_3" d="M 112 228 C 148 203 183 169 254 172
                                C 313 174 361 201 404 229" />
</svg>
```

Figure A. Examples of layout trees T from (a) PKU PosterLayout and (b) PStylish7 datasets.

Prompts P. Fig. B presents examples of prompts created as depicted in Sec. 3.2. The canvases' varying sizes demonstrate PosterO's adaptability to diverse aspect ratios.

Prompt (a)

Preface The following are some scalable vector graphics (svg) allocating elements on the canvas.

Examples

Example 0: This svg uses canvas_0 of size (513, 750) with available areas (45, 504, 441, 716), (66, 45, 443, 195) to allocate{ underlay_1, text_2, logo_3, text_4, text_5 }.

```
<svg width="513" height="750" xmlns="http://www.w3.org/2000/svg">
    <polygon id="available_area" points="45,504 441,504 441,716 45,716" />
    <polygon id="available_area" points="66,45 443,45 443,195 66,195" />
    <rect id="canvas_0" x="0" y="0" width="513" height="750" />
    <svg x="179" y="638">
        <rect id="underlay_1" x="0" y="0" width="157" height="51" />
        <rect id="text_2" x="25" y="13" width="105" height="28" />
    </svg>
    <rect id="logo_3" x="30" y="18" width="146" height="38" />
    <rect id="text_4" x="75" y="110" width="365" height="38" />
    <rect id="text_5" x="80" y="502" width="379" height="68" />
</svg>
```

Example 1: This svg uses canvas_0 of size (513, 750) with available areas (71, 541, 437, 705), (55, 40, 438, 224) to allocate { text_1, text_2, text_3 }.

```
<svg width="513" height="750" xmlns="http://www.w3.org/2000/svg">
    <polygon id="available_area" points="71,541 437,541 437,705 71,705" />
    <polygon id="available_area" points="55,40 438,40 438,224 55,224" />
    <rect id="canvas_0" x="0" y="0" width="513" height="750" />
    <rect id="text_1" x="23" y="109" width="468" height="54" />
    <rect id="text_2" x="27" y="41" width="459" height="60" />
    <rect id="text_3" x="59" y="573" width="397" height="47" />
</svg>
...
```

Postcript

First, learn from the examples and understand how this template works.
Then, create a new one while following the rules:
1. The svg must be meaningful, which implies that empty, all-zero, or symbolic attributes are not allowed.
2. <rect> is the only legal svg tag, and the inner <rect> must be within the outer <svg>.
3. The id of <rect> must be unique and picked from {}.
4. The position of <rect> should be clustered neatly in avaliable areas while avoiding intersection. If intersected, <rect> should be resized or moved.

Final: This svg uses canvas_0 of size (513, 750) with available areas (69, 502, 411, 703), (59, 35, 474, 210) to allocate { underlay_1, text_2, logo_3, text_4, text_5 }.

Prompt (b)

Preface The following are some scalable vector graphics (svg) allocating elements on the canvas.

Examples

Example 0: This svg uses canvas_0 of size (513, 641) with available areas (33, 20, 455, 281) to allocate { text-curved_1 }.

```
<svg width="513" height="641" xmlns="http://www.w3.org/2000/svg">
    <polygon id="available_area" points="33,20 455,20 455,281 33,281" />
    <rect id="canvas_0" x="0" y="0" width="513" height="641" />
    <path id="text-curved_1" d="M 57 139 C 86 124 131 113 167 112 C 240 109 261 145
                                366 176 C 396 185 441 182 465 180" />
</svg>
```

Example 1: This svg uses canvas_0 of size (513, 513) with available areas (33, 0, 478, 171) to allocate { embellishment_1, embellishment_2, text-curved_3 }.

```
<svg width="513" height="513" xmlns="http://www.w3.org/2000/svg">
    <polygon id="available_area" points="33,0 478,0 478,171 33,171" />
    <rect id="canvas_0" x="0" y="0" width="513" height="513" />
    <rect id="embellishment_1" x="43" y="158" width="28" height="25" />
    <rect id="embellishment_2" x="447" y="158" width="27" height="28" />
    <path id="text-curved_3" d="M 85 164 C 112 148 152 124 189 122 C 306 117 367 191
                                436 181" />
</svg>
...
```

Postcript

First, learn from the examples and understand how this template works.
Then, create a new one while following the rules:
1. The svg must be meaningful, which implies that empty, all-zero, or symbolic attributes are not allowed.
2. <rect>, <ellipse>, <path> are the only legal svg tag, and the inner <rect> must be within the outer <svg>.
3. The id of each tag must be unique and picked from {}.
4. The position of each tag should be clustered neatly in avaliable areas while avoiding intersection. If intersected, the tags should be resized or moved.

Final: This svg uses canvas_0 of size (513, 513) with available areas (71, 19, 415, 456) to allocate { text-curved_1 }.

Figure B. Examples of prompts P on (a) PKU PosterLayout and (b) PStylish7 datasets.

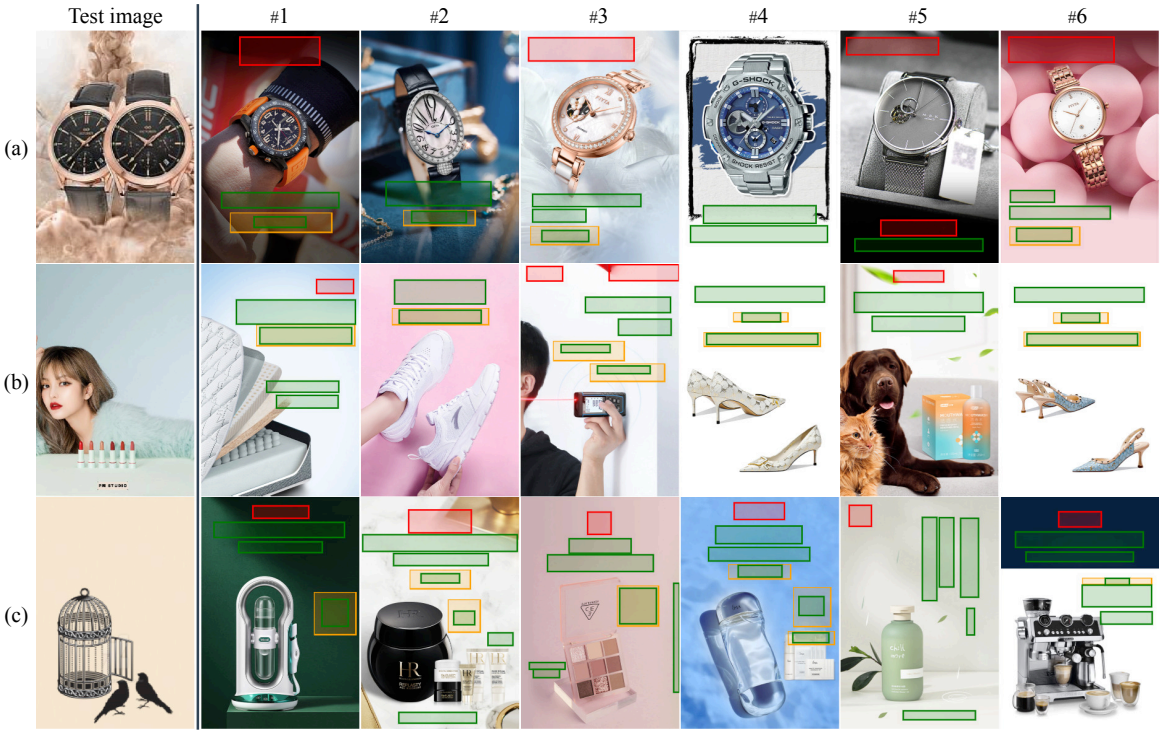


Figure C. Intent-aligned example selection results (#*) on the *unannotated test split* of PKU PosterLayout dataset.

B. Additional Experimental Results

Selected in-context examples. Fig. C shows the k layout trees $\{T_j\}_j^k$ selected as in-context examples that align with the intents of the given test images. By observing (a), we see that the design intent latent space $\mathbf{Z}_D$ effectively models both semantics and spatial distribution of objects within the images. While the composition of the test images appears similar in (b) and (c), the selection results are very different. These findings demonstrate that our selection procedure provides a comprehensive understanding of visual contents and is sensitive to subtle differences.

More visualized results. Fig. D and Fig. E visualize the layouts generated by all baselines [4, 17, 19, 21, 31, 44, 55] and the proposed PosterO. Elements violating graphic metrics and content metrics are respectively indicated by blue and yellow arrows. The comparisons with ground truth demonstrate that PosterO most accurately predicts the design intents and actively utilizes the available areas. Besides, it understands the graphic relationship in layouts well and generates non-overlapping, aligned elements.

More ablation studies on vision processing. As reported in Tab. A, we investigate more configurations as follows.

(1) Visual condition vectorization: To demonstrate the advantages of design intent over the widely utilized salient regions, the variant 'Saliency' as in LayoutPrompter [31] is implemented. It is observed that all content metrics severely drop, surprisingly, even worse than not performing any visual perception (*-d-f*). This underscores that design intent is an encouraging substitute for the current saliency paradigm.

	$Und_l\uparrow$	$Und_s\uparrow$	$Int\downarrow$	$Sal\downarrow$	$Rea\downarrow$	$Avg\downarrow$
Ours	**0.9856**	**0.9241**	**0.1427**	0.2131	0.0248	**0.0677**
Visual condition vectorization (Ours: Design intent)						
Saliency	0.9694	0.9114	1.2470	1.1867	0.0374	0.3702
Visual information perception (Ours: *d+f*)						
-d-f	0.6873	0.5776	1.1941	0.4980	0.0275	0.3512
+v	0.8784	0.6092	0.4868	0.4145	0.0270	0.2063
+v-d	0.8874	0.7429	0.3142	**0.2078**	**0.0239**	0.1311
+v-d-f	0.8335	0.6399	0.6432	0.6116	0.0302	0.2591

Table A. More ablation studies on visual condition vectorization and visual information perception. (*d*: Design intent vectorization, *f*: Intent-aligned example selection, *v*: LLaMA-3-LLaVA-NeXT.)

(2) Visual information perception: To demonstrate our (*d*, Sec. 3.1) vectorized conditions and (*f*, Sec. 3.2) learning examples have already considered input images very well, three variants are implemented with (*v*) LLaMA-3-LLaVA-NeXT[§]. It is a large multimodal model (LMM) with powerful visual reasoning capabilities based on Llama 3-8B. However, *v* only marginally improves $Sal\downarrow$ and $Rea\downarrow$ when it replaces *d*, and the rest variants obtain poor results. As our detection model (16M) is much smaller than LLaVA-NeXT's vision head (encoder: 304M, connector: 20M), *d* is a proper alternative. In addition, using *d* avoids cross-modal differences between visual tokens and geometric layout elements, reflecting on its better graphic performance.

[§]https://huggingface.co/llava-hf/llama3-llava-next-8b-hf

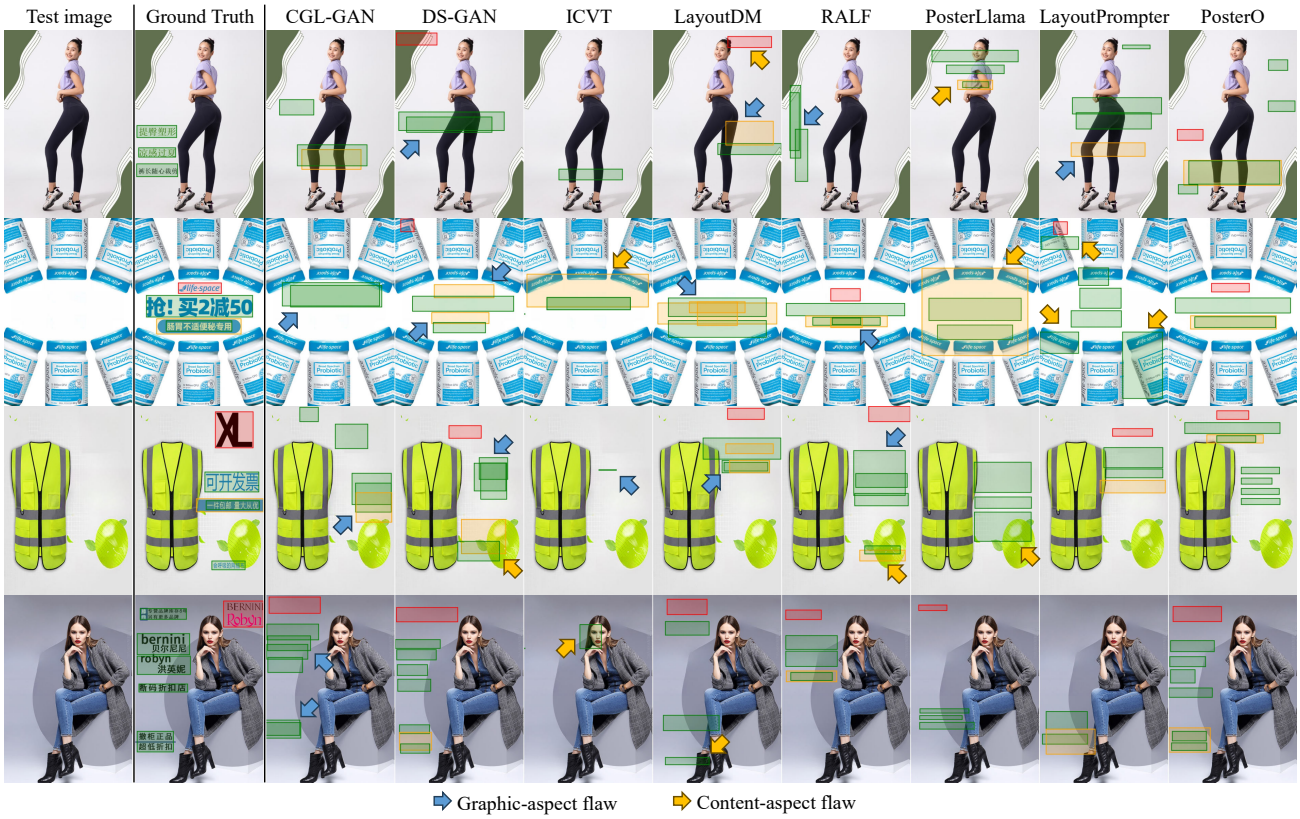


Figure D. Visualized results on the *annotated test split* of PKU PosterLayout dataset.

Figure E. Visualized results on the *annotated test split* of CGL dataset.

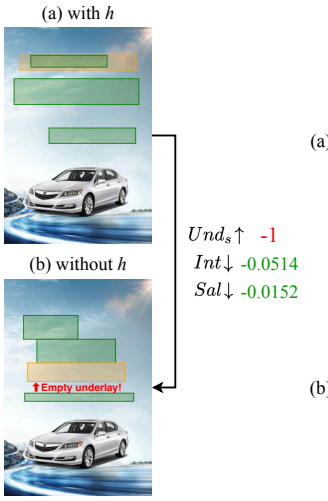


Figure F. Impact of removing h: hierarchical nodes in Tab. 5.

Figure G. Results of poster design realization.

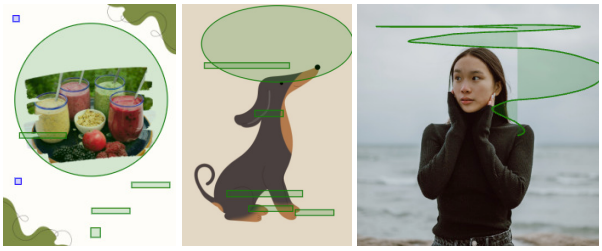
Ours$_{L2}$	*Paint*	*Poem*	*Metro*	*Movie*	*Menu*	*Animal*	*Insta.*
$Ove\downarrow$	0.0094	0.0068	0.2622	0.0094	0.0063	0.0085	0.0004
$Int\downarrow$	0.4159	0.2402	0.6382	0.7213	0.1260	0.3516	0.1408
$Sal\downarrow$	3.1897	0.6106	0.5532	0.5806	0.4312	1.6667	0.0590
$Avg\downarrow$	1.2050	0.2859	0.4845	0.4371	0.1879	0.6756	0.0668

Table B. More quantitative results on PStylish7 dataset.

'Negative' impact of hierarchical nodes. As the enclosing structure is equivalent to *nested layouts*, the current design of (h, Sec. 3.1.3) hierarchical node representation has effectively constrained the solution space. However, by observing the first two rows of Tab. 5, removing h appears to improve the standardized content metrics, $Int\downarrow$ and $Sal\downarrow$. To determine the primary cause, we analyzed the visualization results and found a significant occurrence of *empty underlays*, as depicted in Fig. F. These invalids improve in Cov and Uti, thereby deceptively enhancing the standardized metrics. This finding again highlights the importance of h in generating layouts with satisfactory integrity.

Results of poster design realization. Sec. 3.3 introduces the zero-shot transformation from generated layouts to actual posters. Fig. G shows the results of each step and the final designs. During *mockup creation*, not only the font size and text color are properly predicted, but also the link to the logo image, *i.e.*, `href` attribute, is correctly created with an initial unknown value. For *material synthesis*, all given contents are perfectly placed into the elements of corresponding `id`. We also request LLMs with *style* in simple words, such as *elegant*, and the resultant posters are astonishingly appealing.

PosterO$_{L2}$ on PStylish7. In the main text, Tab. 8 reported the quantitative results of PosterO$_{CL}$ and PosterO$_{L3}$ on PStylish7. Here, Tab. B reports those of PosterO$_{L2}$. Surprisingly, it achieves the best overall performance on *Metro*, *Menu*, *Animal*, and especially *Instagram*. By accumulating



(a) *Menu* (b) *Animal* (c) *Instagram*

Figure H. Visualized results of PosterO$_{L2}$ on PStylish7 dataset.

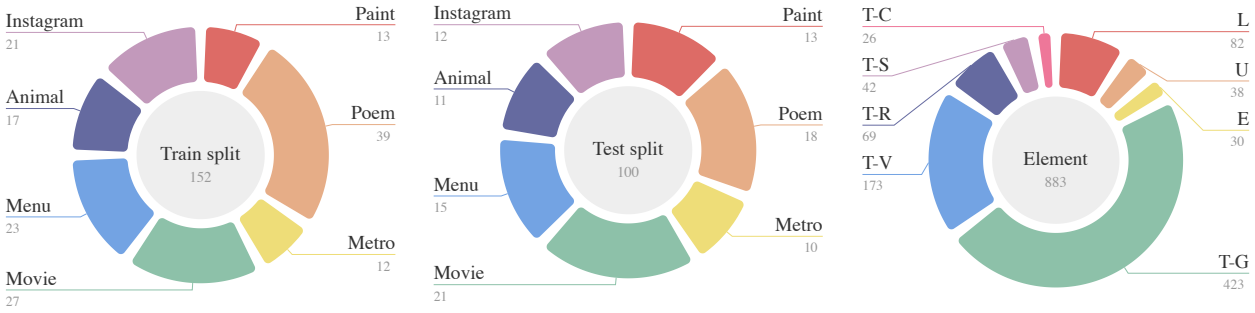
$Avg\downarrow$ across seven categories, PosterO$_{L2}$, $_{CL}$, and $_{L3}$ obtain **3.3427**, 4.3799, and 4.2877, respectively. We visualize some amazing layouts generated by PosterO$_{L2}$ in Fig. H.

C. PStylish7: More Details and Statistics

PStylish7 is the first dataset for generalized content-aware layout generation. We built it with 152 image-layout pairs for few-shot learning and 100 image canvases for benchmark testing. The data comes from a variety of sources, including Canva, Pixabay, Ucshe, and the New York Heritage Digital Collections. In the remaining section, statistics on the distribution of sample categories, element types, and image aspect ratios within the PStylish7 dataset are provided to offer a clear understanding of its diversity and complexity. Additionally, a detailed explanation of the metrics calculation is presented.

C.1. Statistics

Sample categories. The seven categories in PStylish7 are artwork exhibition (*Paint*), cultural education (*Poem*), public safety (*Metro*), entertainment marketing (*Movie*), merchandising display (*Menu*), public advocacy (*Animal*), and social-media interaction (*Instagram*). The distributions of



(a) Distribution of poster categories (b) Distribution of canvas categories (c) Distribution of layout element types

Figure I. Statistics on sample categories and layout element types in PStylish7 dataset.

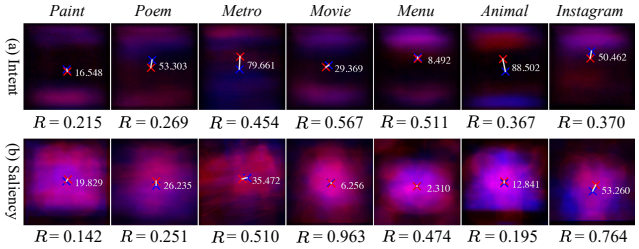


Figure J. Varied difficulty degrees of PStylish7 categories.

samples across different categories in the train and test splits are illustrated in Fig. I(a) and (b).

Inter-category gaps. With the diverse purposes and entities in different categories, their difficulty degrees H are also noticeably varying. To gain insight into H, we analyzed each category by visualizing the distribution variation ΔD between their train/test splits and indicating the trade-off factors $R = \frac{Unmatch_L}{Match_L}$ for content metrics, as shown in Fig. J. As observed, $H \propto \frac{\Delta D}{R}$, aligning with PosterO's performance reported in Tab. 8 and Tab. B.

Layout element types. The eight layout element types within PStylish7 are logo (L), underlay (U), embellishment (E), and text (T-G, general), along with four text variants (T-V, vertical; T-R, rotated; T-S, ellipse; T-C, complex curve), which are exclusive to this new dataset. The distribution of elements across different types is illustrated in Fig. I(c), showing a total of 883 elements. Notably, over one-third of them are specialized variants, highlighting the dataset's emphasis on capturing the complexity inherent in real-world design work.

Image aspect ratios. The distribution of image aspect ratios within PStylish7 is illustrated in Fig. K. Unlike existing domain-specific datasets [19, 55], where samples are predominantly of a fixed aspect ratio (*i.e.*, 0.68), PStylish7 includes a variety of aspect ratios, with the most common being 5:7 (*i.e.*, 0.71), 9:16 (*i.e.*, 0.56), and 1:1. This highlights the dataset's emphasis on providing a more realistic conditions for layout generation tasks.

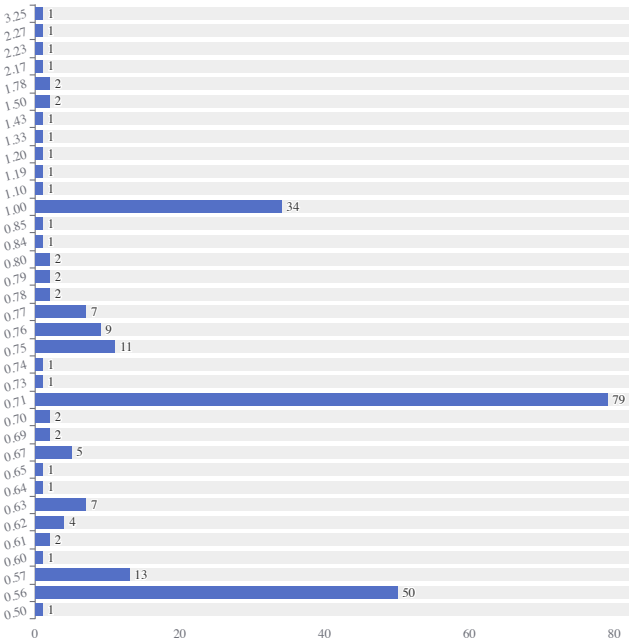
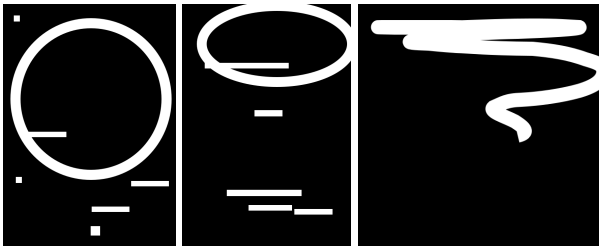


Figure K. Distribution of image aspect ratios in PStylish7 dataset.

C.2. Metrics

Given the high flexibility of the specialized elements, current numerical metrics (*e.g.*, $Ali \downarrow$) often fail to evaluate their placement and organization. To this end, we resort to pixel-level metrics and regularize the rendering process of element maps for computation. While the content metrics (*i.e.*, ($Cov \uparrow$, $Con \downarrow$), standardized as $Int \downarrow$, and ($Uti \uparrow$, $Occ \downarrow$), standardized as $Sal \downarrow$) are inherently based on element maps, the graphic ones are not. Therefore, we introduce a pixel-level overlay $Ove \downarrow$ to fill in this vacancy.

Element map rendering process. An element map $m_{e_i} \in \{0,1\}^{h \times w}$ serves as the visual indicator revealing the spatial coverage of layout elements $\{e_i\}_i^n$. Fig. L illustrates the corresponding element maps of layouts in Fig. H. While rectangular elements are straightforwardly filled with white, those approximated by ellipses and curves require a different way. Concretely, we utilize strokes with a width of 30 pixels to outline these elements. Every element map is uni-



(a) (b) (c)

Figure L. Rendered element maps of layouts in Fig. H.

formly scaled to a width h of 513 pixels to maintain consistency in scale for appropriate comparisons. With our SVG language-based representations, the rendering can be easily implemented by introducing CSS-style declarations as:

```
rect { fill: white; }
ellipse { stroke: white; stroke-width: 30; }
path { stroke: white; stroke-width: 30; }
```

Pixel-level overlay $Ove\downarrow$**.** To calculate overlay between non-underlay elements $\{e_j\}_j^m$, each element e_j is rendered individually as a layer ℓ_j of the element map. The overlay is then determined by calculating the sum of layers $\{\ell_j\}_j^m$ and subtracting the map m_{e_j} that results from their combination. Specifically, as the maximum value of each pixel in the map is 1, this pixel-level operation accumulates the number of pixels where more than one element is present. By dividing the size of the map m_{e_j}, we obtain a normalized measure of overlay.