LayoutDETR: Detection Transformer Is a Good Multimodal Layout Designer (Supplementary Material)

Ning Yu, Chia-Chih Chen, Zeyuan Chen, Rui Meng, Gang Wu, Paul Josel, Juan Carlos Niebles, Caiming Xiong, and Ran Xu

Salesforce Research
{ning.yu, chiachih.chen, zeyuan.chen, ruimeng, gang.wu, pjosel, jniebles, cxiong, ran.xu}@salesforce.com

1 Implementation Details

Architecture design is where we integrate object detection with layout generation. Detection transformer (DETR) architecture [6] is employed and modified for LayoutDETR generator G and conditional discriminator D^{c}. It targets to boost the understanding of background from the perspective of visual detection, and enhance the controllability of background on the layout.

As depicted in Fig. 2 bottom left in the main paper, G and D^{c} contain a visual transformer encoder for background understanding and a transformer decoder for layout generation or discriminator feature representation. The encoder part is the same as in DETR [6], and is identical in G and D^{c}. It consists of a CNN backbone that extracts a compact feature representation from a background image, as well as a multi-head ViT encoder [9, 17] that incorporates positional encoding inputs [5, 14]. It outputs tokenized visual features for cross-attention in the following layout transformer decoder.

The layout decoder is also inherited from the DETR transformer decoder with self-attention and encoder-decoder-cross-attention mechanisms [16]. In G, it transforms each of the N input embeddings (corresponding to N foreground elements) into layout bounding box parameters, whereas in D^{c}, it transforms each of the N bounding box embeddings into discriminator features. Our architecture differs from DETR where we have foreground elements as inputs to drive the transformation, while DETR does not. Therefore, we replace their freely-learnable object queries with our foreground embeddings as the input tokens to the decoder, which are detailed below.

In G, foreground elements are composed of texts $\mathcal{T} = \{\mathbf{t}^i\}_{i=1}^M = \{(\mathbf{s}^i, c^i, l^i)\}_{i=1}^M$ and image patches $\mathcal{P} = \{\mathbf{p}^i\}_{i=1}^K$. Thence, each foreground embedding is a concatenation of noise embedding and either text embedding or image embedding. To calculate the text embedding, we separately encode text string $\mathbf{s}$, text class c, and text length l, and concatenate the features together. The text string is encoded by the pretrained and fixed BERT text encoder [8]. The text class and quantized text length are encoded by learning a dictionary. To calculate the image embedding, we use the same ViT as used for background encoding. The weights are shared and initialized by the Up-DETR-pretrained model [7]. Note

that the font color is not considered in the modeling because it is trivial information. According to our empirical observation, font colors are dominated by two and only two modes: black and white. As indicated in Fig. 4 caption in the main paper: Text font colors and button pad colors are adaptively determined to be either black or white whichever contrasts more with the background.

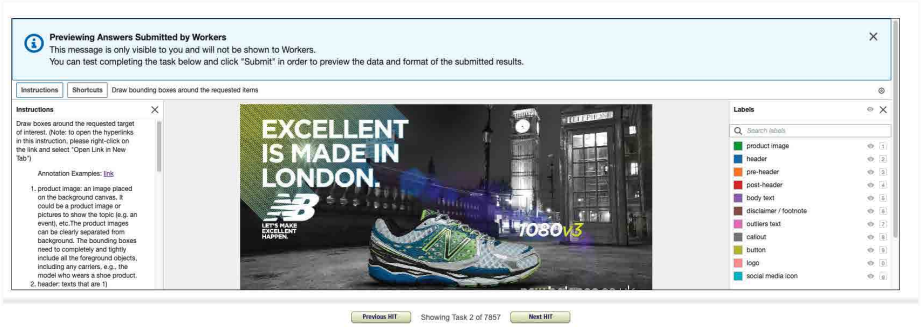
For the other networks F^{c}, D^{u}, F^{u}, E, and R that do not take background images as an input condition, we simply use the above transformer decoder architecture as their implementations. Following transformers, for each foreground image reconstruction in F^{c} and R, we employ the StyleGAN2 image generator architecture [11]. For each text string reconstruction, we employ the pretrained BERT language model decoder [8, 13]. For each text class and text length decoding, we use 3-layer MLPs.

We implement LayoutDETR in PyTorch and use Adam optimizer [12] to train the models on 8 NVIDIA A100 GPUs, each with 40GB memory. Because bounding box parameters are normalized by image resolutions, during training and inference we downsize arbitrary images to 256×256 without losing generality. For final rendering and visualization, we resize them back to their original resolutions. Small and unified image size allows us to train models with a large enough batch size, e.g., 64 in our experiments. During training, we set the learning rate constantly as 10^{-5} and train for 110k iterations in 4 days. Inference is much more efficient: we load only G into a single NVIDIA A100 GPU and it consumes only 2.82GB of memory. It takes only 0.38 sec to generate a layout given foreground and background conditions.

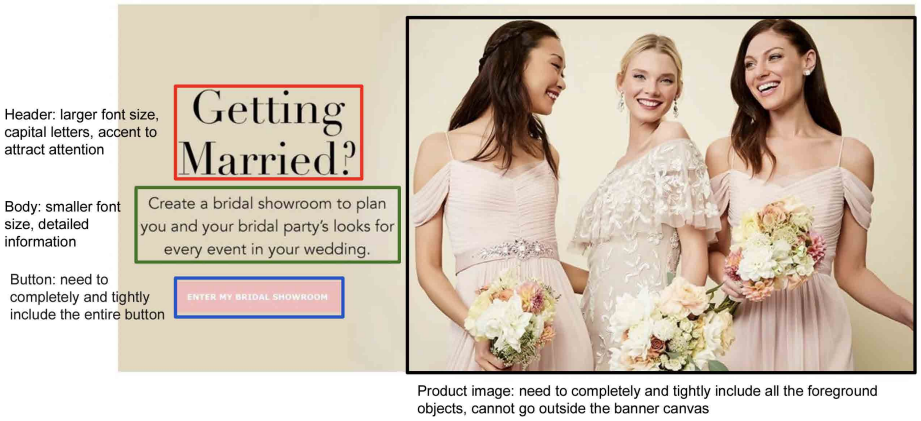
2 Details of Our Ad Banner Dataset Collection

The sources of raw ad banner images consist of two parts. First, we manually went through all the images in Pitt Image Ads Dataset [10]. We filtered out those with single modality, low quality, or old-fashioned designs. We then selected 3,536 valid ad banner images. Second, we searched on Google Image Search Engine with the keywords "XXX ad banner" where "XXX" goes through a list of 2,765 retailer brand names including the Fortune 500 brands. For each keyword search, we crawled the top 20 results and manually filtered out non-ads, single-modality, low-quality, or offensive-content images. We then selected 4,321 valid ad banner images. Combining the two sources, we in total obtained 7,857 valid ad banner images with arbitrary sizes.

Next, we crowdsourced on Amazon Mechanical Turk (AMT) [1] to obtain human annotations for the bounding box and class of each text phrase in each image. The class space spans over 11 categories as shown in the AMT interface in Fig. 1 top. Without losing representativeness, we focus on the top-4 most common categories in this work: {*header*, *body text*, *disclaimer / footnote*, *button*}. We also linked a detailed instructional document with examples for workers to fully understand the annotation task. See the instruction in Fig. 1 bottom. We assigned each annotation job to three workers. For the final annotation results, we averaged over three workers' submissions and incorporated our judgments



(a) AMT interface



(b) Instructional example

Fig. 1: Top: AMT interface with instructions on the left for users to annotate the bounding box and class of each existing copywriting text on each image. **Bottom**: one instructional example of the definitions of text bounding boxes and text classes.

for the tie cases. In total there were 67 workers involved in the task. On average each worker submitted 314 jobs in around 3 minutes per job. All of the workers have an approval rate history above 90% on AMT. We did not set any restrictions for workers' gender, race, sexuality, demographics, locations, remuneration rates, etc. Our annotation process has been reviewed and approved by our ethical board.

After annotation, it is necessary to reverse the design by separating foreground elements from background images to configure the training/testing data. We apply a modern optical character recognition (OCR) technique [2] to extract the text inside each bounding box, and adopt a modern image inpainting technique [15] to erase the texts. The separation of texts from background images is exemplified in Fig. 2. After filtering out a few samples with undesirable OCR or inpainting results, we finally obtain 7,196 valid samples for the following experiments.

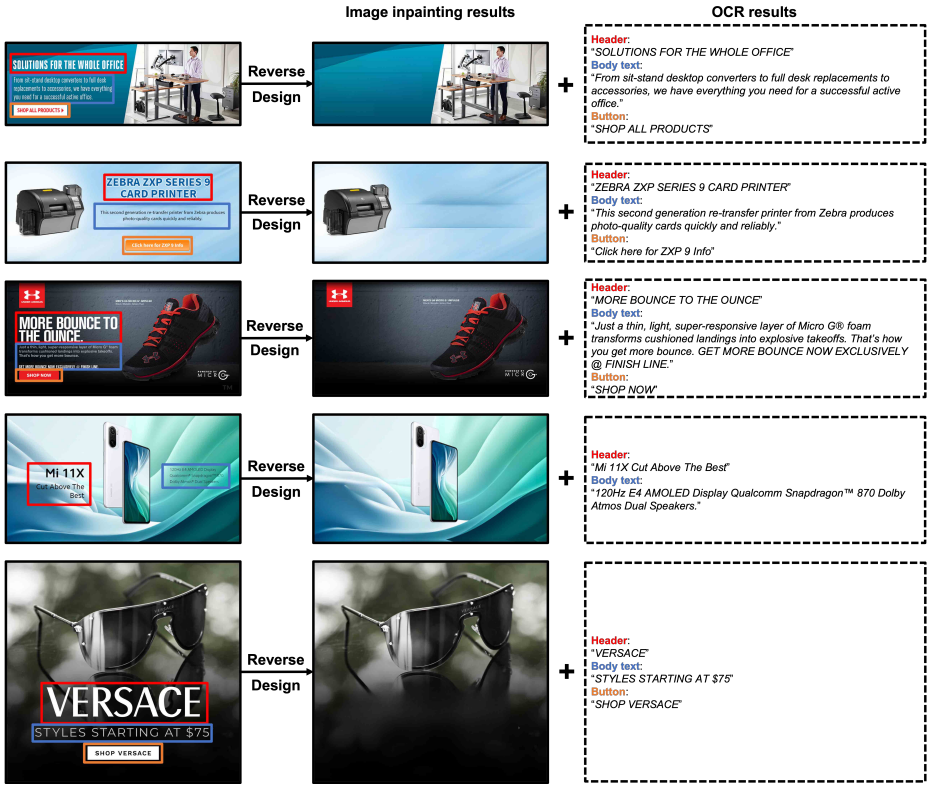
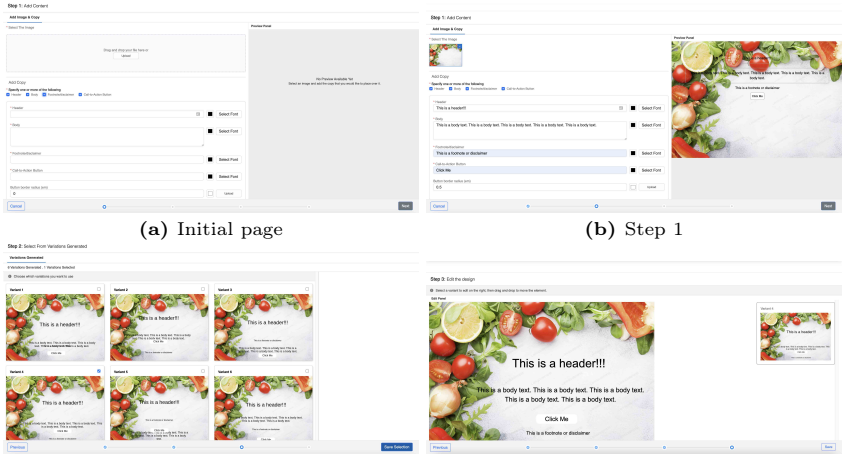


Fig. 2: Reverse engineering examples of separating foreground elements from background images using OCR and image inpainting techniques.

It is worth noting that inpainting clues may leak the layout bounding box ground truth information and shortcut training. Therefore, during training, we intentionally inpaint background images at additional random subregions that are irrelevant to their layouts.

3 Graphical System Step-by-Step Designs

Since we validate that our solution sets up a new state of the art for multimodal layout design, it is worth integrating it into a graphical system for user-friendly service in practice. Fig. 3 demonstrates our step-by-step UI designs. In specific:
(1) Fig. 3(a) shows the initial page that allows users to customize their background images and optionally foreground elements: *header text*, *body text*, *footnote/disclaimer text*, *button text*, as well as *button border radius* (zero radius means a rectangular button). Text colors, button pad colors, and text fonts can also be customized.
(2) Once users upload their background and foreground elements, they are previewed on the right part of the same page, as shown in Fig. 3(b). The locations



(a) Initial page **(b)** Step 1

(c) Step 2 **(d)** Step 3

Fig. 3: Step-by-step usage of our graphical system for customizable multimodal graphic layout design.

and sizes of foreground elements in the preview are meaningless: they just conceptually show what contents are going to be rendered on top of the background.
(3) Once users click "Next", it moves on to the next page with our design results, as shown in Fig. 3(c). Given one layout designed in the backend, we post-process it by randomly jittering the generated box parameters by 20% while keeping the original non-overlapping and alignment regularity.
(4) Afterwards, the system renders foreground elements given the layout bounding boxes. Text font sizes and line breakers are adaptively determined so as to tightly squeeze into the boxes. Considering *header texts* usually have short strings yet are assigned with large boxes, their font sizes are naturally large enough. Text font styles can also randomly vary. This optional feature is shown in our supplementary video. We showcase six of our rendered results on this page, and allow users to select one or more satisfactory designs.
(5) Once users make their selection(s) and click "Save Selection", it moves onto the last page as shown in Fig. 3(d). On this page, users are allowed to manually customize the size and location of each rendered foreground element. Once they finish, they click "Save" to exit our system.

More live demonstrations are nested in our supplementary video.

4 More qualitative results

We show in Fig. 4 more uncurated qualitative results of layout designs and text rendering on background images in the wild and on CGL Chinese ad banner inpainted background [18]. Conditioned on multiple text inputs in varying categories, our designs appear aesthetically appealing and harmonic between foreground and background.

We show in Fig. 5 the impact of varying texts on layouts given the same background image. We observe: (1) the scales of bounding boxes are adaptively proportional to the varying lengths of texts such that the font sizes remain approximately unchanged, and (2) the global and relative locations of bounding boxes are stable regardless of the changes of texts, which are reliably harmonic with the background structure.

5 Limitation on Challenging Samples

We show in Fig. 6 a few imperfect layout designs for challenging samples. When the background images are over-clutter and texts are wordy, none of our rendering variants looks very ideal. Our model struggles between (1) placing layouts in the middle regardless of background and (2) placing layouts over less busy areas at edges that breaks the spatial balance. A possible workaround could be introducing gradient blending masks into the rendering post-processing.

References

1. https://www.mturk.com/
2. https://github.com/PaddlePaddle/PaddleOCR
3. https://www.freepik.com/
4. https://www.pmi.com/
5. Bello, I., Zoph, B., Vaswani, A., Shlens, J., Le, Q.V.: Attention augmented convolutional networks. ICCV (2019)
6. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. ECCV (2020)
7. Dai, Z., Cai, B., Lin, Y., Chen, J.: Up-detr: Unsupervised pre-training for object detection with transformers. CVPR (2021)
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. NAACL (2019)
9. He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. CVPR (2022)
10. Hussain, Z., Zhang, M., Zhang, X., Ye, K., Thomas, C., Agha, Z., Ong, N., Kovashka, A.: Automatic understanding of image and video advertisements. CVPR (2017)
11. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of stylegan. CVPR (2020)
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. ICLR (2015)
13. Li, J., Li, D., Xiong, C., Hoi, S.: Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. ICML (2022)
14. Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., Tran, D.: Image transformer. ICML (2018)
15. Suvorov, R., Logacheva, E., Mashikhin, A., Remizova, A., Ashukha, A., Silvestrov, A., Kong, N., Goka, H., Park, K., Lempitsky, V.: Resolution-robust large mask inpainting with fourier convolutions. WACV (2022)
16. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. NeurIPS (2017)

17. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local neural networks. CVPR (2018)
18. Zhou, M., Xu, C., Ma, Y., Ge, T., Jiang, Y., Xu, W.: Composition-aware graphic layout gan for visual-textual presentation designs. IJCAI (2022)

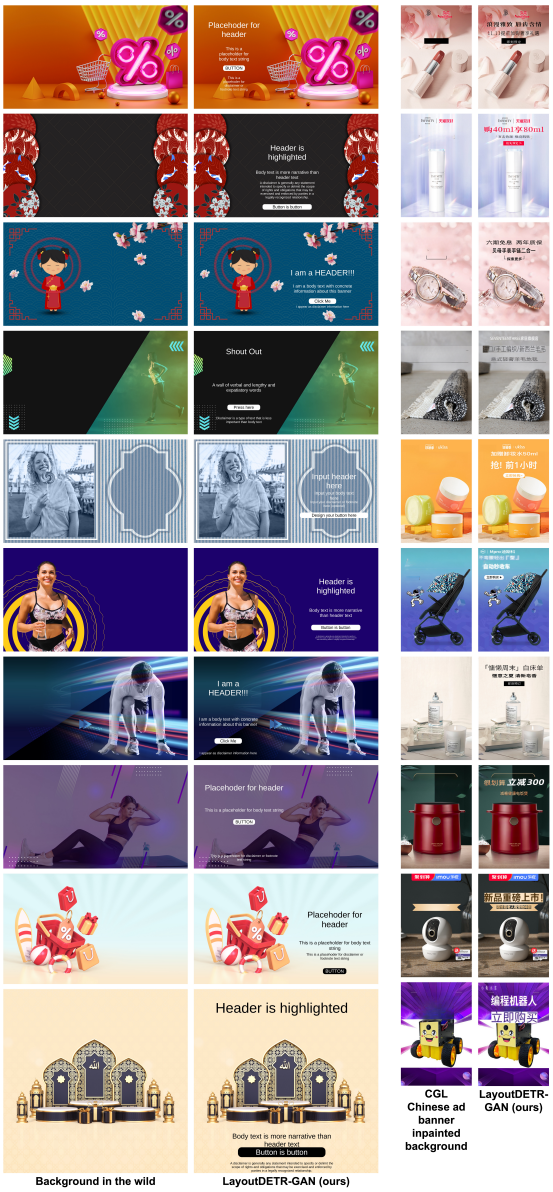


Fig. 4: Left: uncurated layout designs and text rendering on background images in the wild (extracted from PSD data downloaded from [3] with searching keywords "ad banner"). Rendering rules: (1) Text font sizes and line breakers are adaptively determined to tightly fit into their inferred boxes. (2) Text font colors and button pad colors are adaptively determined to be either black or white whichever contrasts more with the background. (3) Button text colors are then determined to contrast with the button pads. (4) Text font is set to *Arial*. **Right**: uncurated layout designs and text rendering on CGL Chinese ad banner background images inpainted by [15]. Image patches that contain foreground text elements are resized and overlaid on the background following the generated layouts.

Fig. 5: Top: ad banner design given the same background image (downloaded from [4]) and varying text combinations. **Bottom**: ad banner design given the same background image and varying only the *header* text component.

Fig. 6: Imperfect layout designs for over-clutter background images and wordy texts. Image sources are from [4].