Towards Layout-Aware Screenshot Generation

Yuan Gao

December 12, 2023

1 Motivation

Current UI generation models are commonly template-based or code-based, requiring external tools like browsers to visualize and cannot handle user specified layout. On the other side, Diffusion Models (DM) are good at controllable generation but struggled with structured text inpainting. ln this work, we tackled those problems by leveraging DM and Vision-Language (VL) Model for controllable UI generation with text rendering, and our method outperformed SoTA Diffusion Models like Stable Diffusion for title conditioned generation and ControlNet for title & layout conditioned generation.

2 Previous and Related Works

2.1 Website Generation

To the best of our knowledge, there aren't existing open source generative model for website generation. There are, however, online website builders like , but we differ from them as they mainly use template-based pipeline. They have fixed layout template and they infill images by searching images online. Also, they are often code-based, meaning that their website UI requires external browser to render, while our model directly output images.

2.2 Document Generation

There are also existing works on document generation. [TUHS23] directly finetune StableDiffusion [RBL+21] on document images, but the document lacks diversity in the layout and the texts are often non-readable.[HLC+23] builds a diffusion model that generates layout for documents, but they didn't proceed to infill elements in the layout.

2.3 Layout Generation

Current state-of-the-art methods mainly do layout generation with two types of models: Autogressive models with transformers [GLA+21] and diffusion-based methods[IKSS+23]. We adopt the latter as empirically it provide more consistent visual outputs. However, these models haven't extend beyonf layout generation.

3 Dataset

We wrote automatic pipeline to render screenshots and construct layouts. Specifically, we use Playwright [1], which provides a programmatic interface to headless browsers that we use to render raw HTML files into screenshots. Obtaining layout from a screenshot is realized by grouping elements under the same sub-tree in the HTML. Specifically, for each element we obtain its *cleaned xpath* by only keeping tags in `[<p>,<table>,<form>,<dl>,<button>,<ol>, <ul>,<nav>,<img>,<object>]` as they represent the semantic abstraction of the element. Then, we use the tags above as classes and we group each elements according to the value of their *cleaned xpath* to form layout of the screenshot. A visualization of the layout from a screenshot is shown in Figure 1. Besides, we also obtain the title of each website by extracting the `<title>` tag of HTML DOM Tree.

[1] https://github.com/microsoft/playwright

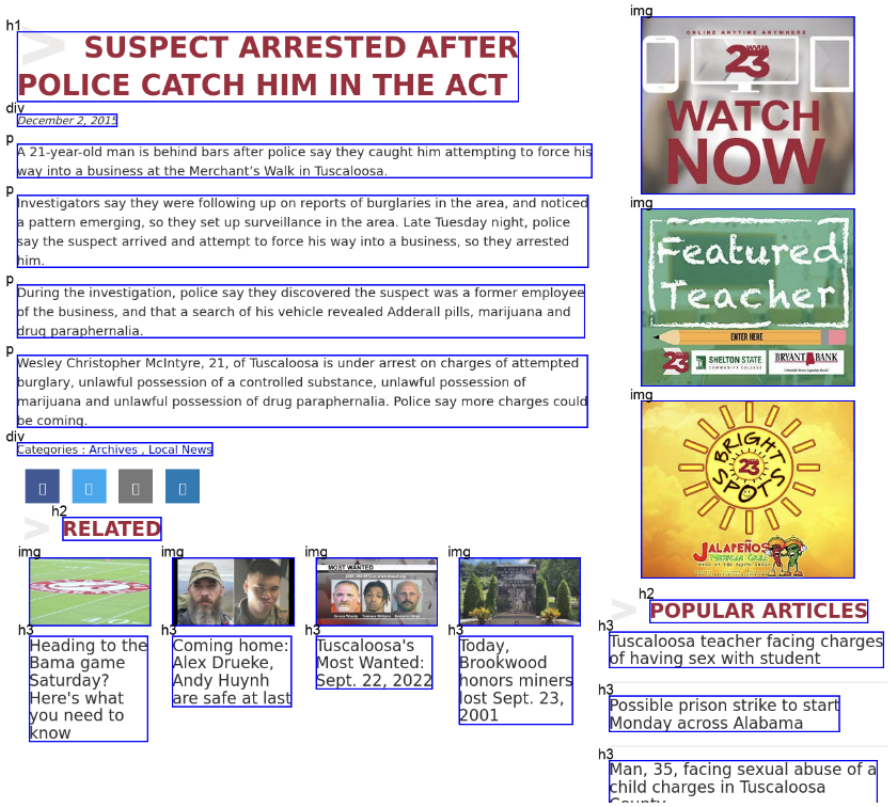

Figure 1: Visualization of layout parsed from a screenshot

4 Technical Approach and Experiments

4.1 Preliminary: Diffusion Models

Denoising Diffusion Probabilistic Models (DDPMs)[HJA20] are a class of generative models that transform data by applying a series of gradual, stochastic steps of noise addition and denoising. The process involves two key phases: a forward (noising) process and a reverse (denoising) process.

1. **Forward Process**: The forward process is a Markov chain that gradually adds noise to the data over a series of steps:
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

2. **Reverse Process**: The reverse process aims to denoise the data, essentially learning to reverse the forward process. This is where the model p_θ comes into play, which approximates the noise that was added at each step in the forward process. The denoising step can be formulated as:
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}|\mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

3. **Training**: The model is trained to minimize a specific objective function, often as the variational lower bound:
$$\begin{aligned} L_{vb} = &\mathbb{E}_{q(\mathbf{x}_0)}\left[D_{KL}\left(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T)\right)\right]_{L_T} \\ &+ \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}\left[D_{KL}\left(q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)||p(\mathbf{x}_{t-1}|\mathbf{x}_t)\right)\right]_{L_{t-1}} \\ &- \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right]_{L_0}. \end{aligned} \tag{1}$$

   Note that $q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)$ and $q(\mathbf{x}_T|\mathbf{x}_0)$ can be computed analytically as a gaussian, thus reducing the losses to MSE (mean squared error) terms.

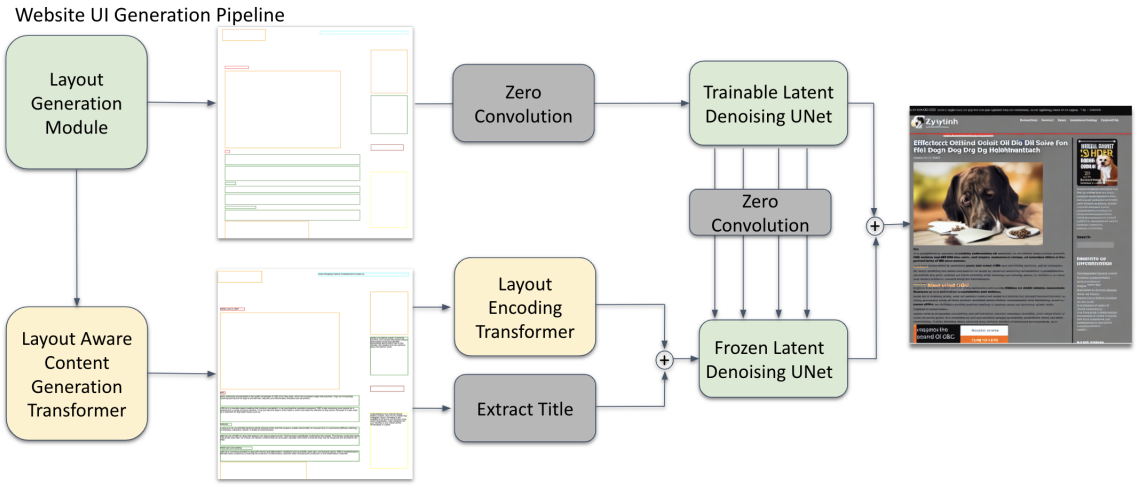

Figure 2: The overall pipeline of our UI generation. We start off by training a layout generation module to estimate $p(l)$, which is based on discrete diffusion model. Then, for content generation we use an auto-regressive vision-language transformer to estimate $p(t|l)$, which generates texts for each region of the layout. Having both $l \sim p(l)$ and $t \sim p(t|l)$, we adopt a diffusion model $log(p_\theta(x_{t-1}|x_t))$ =`NN(Encoder(Layout),Encoder(Layout,Text))` to estimate $p(w|t,l)$. This diffusion model is conditioned on layout images similarly to ControlNet, where the zero convolution is involved, and it's conditioned on the content through cross-attention with the denoising UNet.

4.2 Overall pipeline

As HTML and websites are highly structured data, we factor the generation of websites into two steps: Structure, or layout, generation, and website generation conditioned on the structure. Mathematically, let w denotes the screenshot , t denotes the text in the screenshot, and l denotes the layout of the website. We therefore have

$$p(w) = \mathbb{E}_{l\sim p(l),t\sim p(t|l)}[p(w|l,t)] \tag{2}$$

Thus, to sample a website from $p(w)$, we need divide our method into three steps:
1. Layout Generation Block to estimate $p(l)$: A diffusion model producing layout of the website.
1. Content Generation Block to estimate $p(t|l)$: A auto-regressive model producing content of the website based on the layout.
2. Infilling block: A diffusion model, conditioned on the generated layout, infills each regions with text/images, which is essentially $p(w|l,t)$.

4.3 Layout Generation

We adopt the structure from LayoutDM[IKSS+23], a discrete diffusion model, which has the following changes to DDPM:

1. **Forward Process**: The forward prosses is now perturbing with transition matrix, where $[Q_t]_{ij} = q(x_t = j|x_{t-1} = i)$. and x is an one-hot vector.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \text{Cat}(\mathbf{x}_t; p = \mathbf{x}_{t-1}Q_t),$$

2. **Reverse Process**: The reverse process is described below:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \text{Cat}\left(\mathbf{x}_t; p = \mathbf{x}_0\mathbf{Q}_t\right), \text{ with } \mathbf{Q}_t = \mathbf{Q}_1\mathbf{Q}_2\dots\mathbf{Q}_t, \tag{3}$$

$$\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} = \text{Cat}\left(\mathbf{x}_{t-1}; p = \frac{\mathbf{x}_t\mathbf{Q}_t^\top \circ \mathbf{x}_0\mathbf{Q}_{t-1}}{\mathbf{x}_0\mathbf{Q}_t\mathbf{x}_t^\top}\right). \tag{4}$$

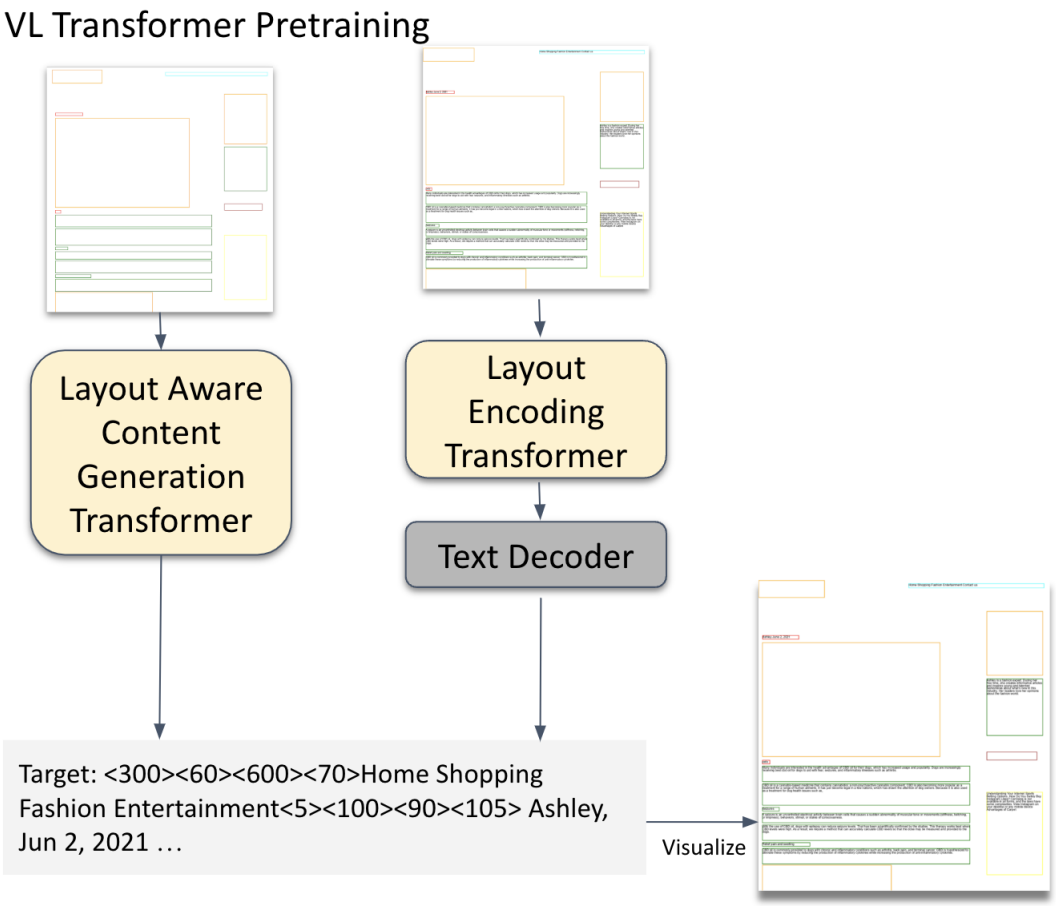

Figure 3: The training of two VL Transformer in our pipeline, which the architecture is based on Pix2struct[LJT+23]. The left one is Layout Aware Content Generator which we use to generate contents (texts) based on the input layout. It is trained by feeding the model with only rendered layout, and ask it to predict the texts contained in the layout bounding box. The right one is a feature extractor where it encodes spatial, semantic, and textual information from the given layout and content. It is trained by feeding the model with the rendered layout as well as the generated content, which is rendered inside each layout box. The target is also to predict the texts contained in the layout bounding box, which encourages the encoder to encode spatial and textual features.

3. **Modeling**: For better training quality, the noise prediction model is factorized as following:

$$p_\theta(x_{t-1}|x_t) \propto \sum_{\tilde{x}_0} q(x_{t-1}, \tilde{x}_0) p_\theta(\tilde{x}_0|x_t).$$

The authors denote the layout for each sample x_i as $\{(c_0, x_0, y_0, , w_0, , h_0), ...\}$, where c is the class of the layout and $x_0, y_0, , w_0, , h_0$ are the binned coordinates, therefore making every dimension a categorical distribution for the discrete diffusion model to train and sample. The class of the layout are common tags in html, including `<img>`,`<span>`,`<p>`,`<input>`,`<h1>`,`<table>`,`etc`

4.4 Layout Conditioned Content Generation

HTML's content is heavily based on the layout since the layout class also encodes the semantic information. While it's possible to use LLM to generate contents directly from prompts, we finetuned train a Vision-Language Transformer in order to produce contents that are more closely related to the layout. Specifically, as shown in 3, the Layout Aware Content Generator adopts Pix2struct[LJT+23]'s encoder-decoder transformer architecture. It is trained by feeding the model with only rendered layout,

and ask it to predict the texts contained in the layout bounding box. During inference, we pass in a layout image and auto-regressively sample tokens for content completion.

4.5 Layout Aware Controllable Generation

Inspired by Text Diffusers [CHL+23], we will use another latent diffusion model varied from Stable-Diffusion[RBL+21] for UI Generation guided by layout and texts. Our initial approach directly turns both conditions into an image by rendering generated texts with the layout and adopt ControlNet[ZRA] for the image conditioning of the diffusion model. However, as shown in section5, this method makes the model having a hard time generating screenshots with recognizable texts, since rendering text on image is clearly not the best representation of texts. Therefore, we instead resort again to Vision-Language Transformer and hope it provides better representation for our conditions. Specifically, given a trained VL encoder that extracts spatial and textual information of the content, we pass the feature concatenated with the title of the website to the diffusion model through cross-attention from the UNet. As for the layout conditioning, we use the structure of the ControlNet where the conditioning is added after zero-convolution. In this way, our diffusion model becomes conditioned on both image and text features through different modules. The pretraining of the VL encoder is described in Figure3, and during optimization of the diffusion model we freeze its weights.

5 Qualitative Results

5.1 Layout Generation

We have ran the layout generation block with LayouDM, and here we show some unconditional samples from the generative model. We can see that the model has learnt to generate some reasonable layouts as shown in Figure 4, while some generated layout have overlaps as shown in Figure 5

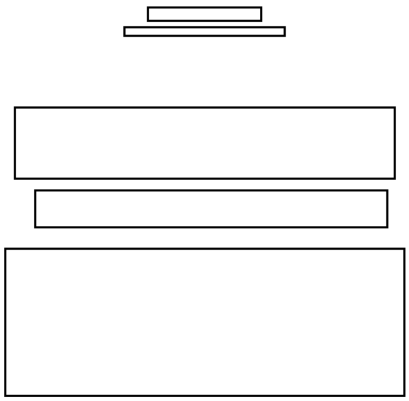

Figure 4: Visualization of reasonable layout generated by our model.

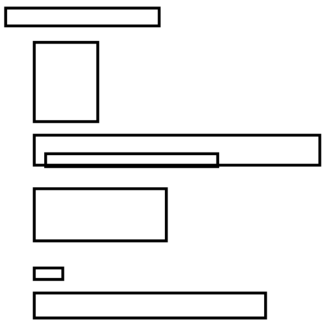

Figure 5: Visualization of unreasonable layout generated by our model, which contains overlaps.

5.2 Layout Guided UI Generation

We conduct both unconditional and conditional generation with our pipeline, and compare it so state-of-the-art diffusion models like Stable Diffusion and ControlNet. We train all of the baselines on 500k images. The Stable Diffusion baseline works as the following: We train text-to-image generation on our dataset for 2 epochs, where we use the title of the website as text prompt. The ControlNet baseline works by taking the finetuned stable-diffusion baseline at epoch 1 and continuously train with rendered layout with texts as image conditions for another 1 epoch. Our pipeline works by first training Stable-Diffusion baseline with additional cross-attention with our VL Encoder for 1 epoch, and then add rendered layout with texts as image conditions for another 1 epoch.

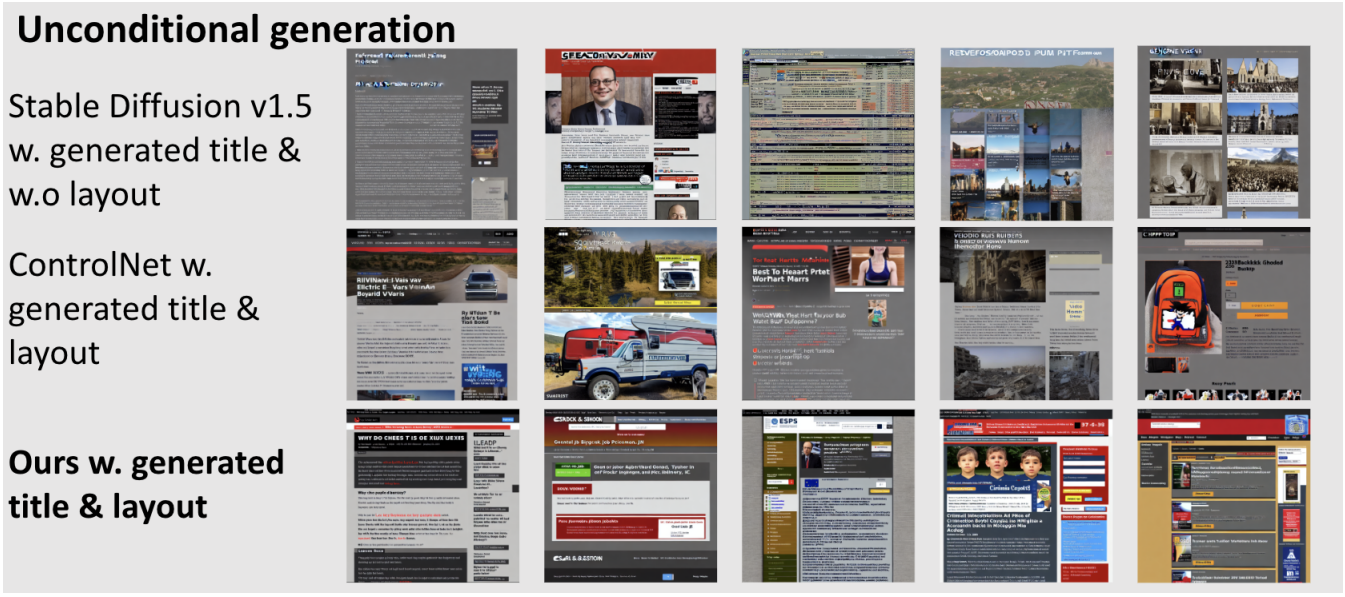

Figure 6: Qualitative comparison between Stable Diffusion, ControlNet, and Our pipeline on unconditional generation.

5.2.1 Unconditional Generation

Under the settings of unconditional generation, we obtain the title from sampling random titles from the dataset. Then we use our layout generation module to randomly sample a layout, which we then pass to the content generator to generate the content. Finally, we pass the obtained title to Stable Diffusion baseline and we pass the the obtained title + layout + content to ControlNet baseline and our pipeline. As shown in Figure6, the stabel diffusion pipeline produces websites that do not have nice arrangement, due to its lack of ability to separate layout with content. The ControlNet Baseline improves the visual arrangement of the generated website as the generate websites shows different blocks. Finally, our proposed pipeline also shows great separation of layout and content, and even produces different color for different layouts.

5.2.2 Conditional Generation

Under the settings of conditional generation, we provide title, layout, and content of the website. As shown in Figure7, both ControlNet and our model is good as producing screenshots that's aligned with the given layout. However, our model produces aesthetically better images and better recognizable texts, as we describe in the next section.

5.2.3 Recognizable Texts

The key difference between our model and ControlNet is the additional condition of spatial and txtual features extarcted by our pretrained VL encoder. This extra condition greatly improves the diffusion model's ability to generate websites with recognizable texts. As shown in Figure8, the diffusion model is asked to generate websites based on the right title. Our model clearly outperforms ControlNet by generating more recognizable texts in the generated images.

6 Quantitative Results

We evaluate our model with the following metrics:
1. Visual appearance of the generated images using FID,
2. Readability of the texts in the website by extracting texts using OCR engine and evaluate how many pages contains recognizable texts.

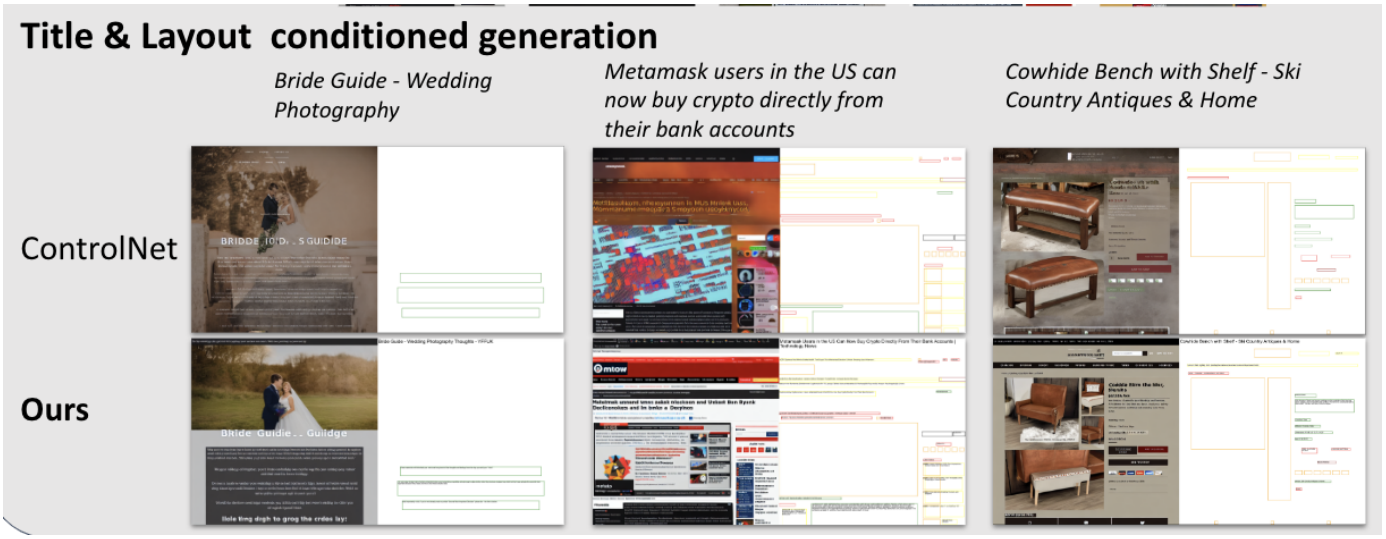

Figure 7: Qualitative comparison between ControlNet, and Our pipeline on conditional generation.

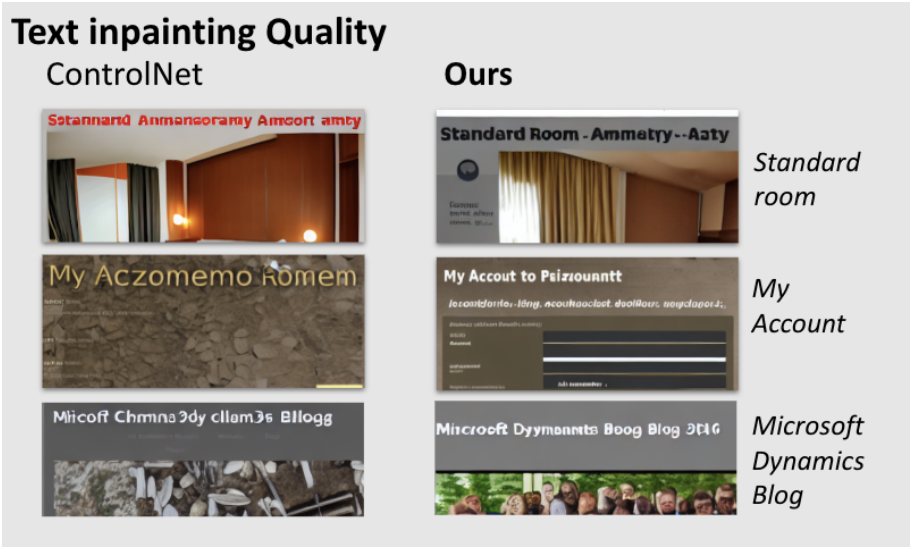

Figure 8: Qualitative comparison between ControlNet, and Our pipeline on generating recognizable texts.

	Unconditional generation FID ↓	Title & Layout conditioned generation FID ↓
Stable Diffusion v1.5	216.0	-
ControlNet	142.2	113.6
Ours	**72.1**	**60.3**

Figure 9: Quantitative FID comparison between ControlNet, and Our pipeline.

	Pages of recognizable texts from OCR (%) ↑
Stable Diffusion v1.5	1.05
ControlNet	9.20
Ours	**11.83**

Figure 10: Quantitative comparison between ControlNet, and Our pipeline on recognizable texts. We report the percentage of webpages that contains readable texts judged by external OCR engine.

6.0.1 Unconditional and Conditional Generation

Following the unconditional and conditional generation procedure described above, we report FID with 5000 images on our test set. As reflected in Table**??**, our model produces much better FID score on both controllable and uncontitional generation, demonstrating the effectiveness of our incorporated VL Encoder.

6.0.2 Recognizable Texts

Here we also report the percentage of webpages that contains readable texts judged by external OCR engine. 11.83 percent of images generated by our pipeline contains recognizable texts, while only 9.2 percent by ControlNet, demonstrating the effectiveness of our incorporated VL Encoder.

7 Limitations and Future Work

Given short training time, our proposed method successfully generates screenshots that aligns with the given layout, and also produces some reasonable text inpaintin, thanks to the encoded layout feature from the VL-Encoder. However, we observe some key obstruction that hampers the quality of the generated images which we can improve in the future:

1. The pretrained VAE for the latent diffusion in our model is not pretrained on the screenshots, so its decoder cannot generate proper text rendering if the text size is small.

2. We pooled the features from the layout encoding transformer to a shorter sequence for the UNet to do cross-attention, so therefore the spatial and text features may be less informative.

References

[CHL+23] Jingye Chen, Yupan Huang, Tengchao Lv, Lei Cui, Qifeng Chen, and Furu Wei. Textdiffuser: Diffusion models as text painters, 2023.

[GLA+21] Kamal Gupta, Justin Lazarow, Alessandro Achille, Larry Davis, Vijay Mahadevan, and Abhinav Shrivastava. Layouttransformer: Layout generation and completion with self-attention, 2021.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020.

[HLC+23] Liu He, Yijuan Lu, John Corring, Dinei Florencio, and Cha Zhang. Diffusion-based document layout generation. In *Lecture Notes in Computer Science*, pages 361–378. Springer Nature Switzerland, 2023.

[IKSS+23] Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. LayoutDM: Discrete Diffusion Model for Controllable Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10167–10176, 2023.

[LJT+23] Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding, 2023.

[RBL+21] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[TUHS23] Noman Tanveer, Adnan Ul-Hasan, and Faisal Shafait. Diffusion models for document image generation. In *International Conference on Document Analysis and Recognition*, pages 438–453. Springer, 2023.

[ZRA] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models.