Design2Code: How Far Are We From Automating Front-End Engineering?

Chenglei Si[*]
Stanford University
clsi@stanford.edu

Yanzhe Zhang[*]
Georgia Tech
z_yanzhe@gatech.edu

Zhengyuan Yang
Microsoft

Ruibo Liu
Google DeepMind

Diyi Yang
Stanford University

[*] Equal Contribution
Project Page: https://salt-nlp.github.io/Design2Code/

Abstract

Generative AI has made rapid advancements in recent years, achieving unprecedented capabilities in multimodal understanding and code generation. This can enable a new paradigm of front-end development, in which multimodal LLMs might directly convert visual designs into code implementations. In this work, we formalize this as a Design2Code task and conduct comprehensive benchmarking. Specifically, we manually curate a benchmark of 484 diverse real-world webpages as test cases and develop a set of automatic evaluation metrics to assess how well current multimodal LLMs can generate the code implementations that directly render into the given reference webpages, given the screenshots as input. We also complement automatic metrics with comprehensive human evaluations. We develop a suite of multimodal prompting methods and show their effectiveness on GPT-4V and Gemini Pro Vision. We further finetune an open-source Design2Code-18B model that successfully **matches the performance of Gemini Pro Vision**. Both human evaluation and automatic metrics show that GPT-4V performs the best on this task compared to other models. Moreover, annotators think GPT-4V generated webpages can replace the original reference webpages in **49%** of cases in terms of visual appearance and content; and perhaps surprisingly, in **64%** of cases GPT-4V generated webpages are considered better than the original reference webpages. Our fine-grained break-down metrics indicate that open-source models mostly lag in recalling visual elements from the input webpages and in generating correct layout designs, while aspects like text content and coloring can be drastically improved with proper finetuning.

1 Introduction

Implementing visual designs of websites into functional code is a challenging task as it requires understanding visual elements and their layouts and then translating them into structured code. Such dependencies on sophisticated skills have prevented many laypeople from building their own web applications, even when they have concrete ideas for what to build or design. Furthermore, the requirement for domain expertise complicates the whole webpage production pipeline, requiring collaboration among people with different skill sets and potentially causing discrepancies between the intended design and actual implementation. Effective automatic generation of functional code from

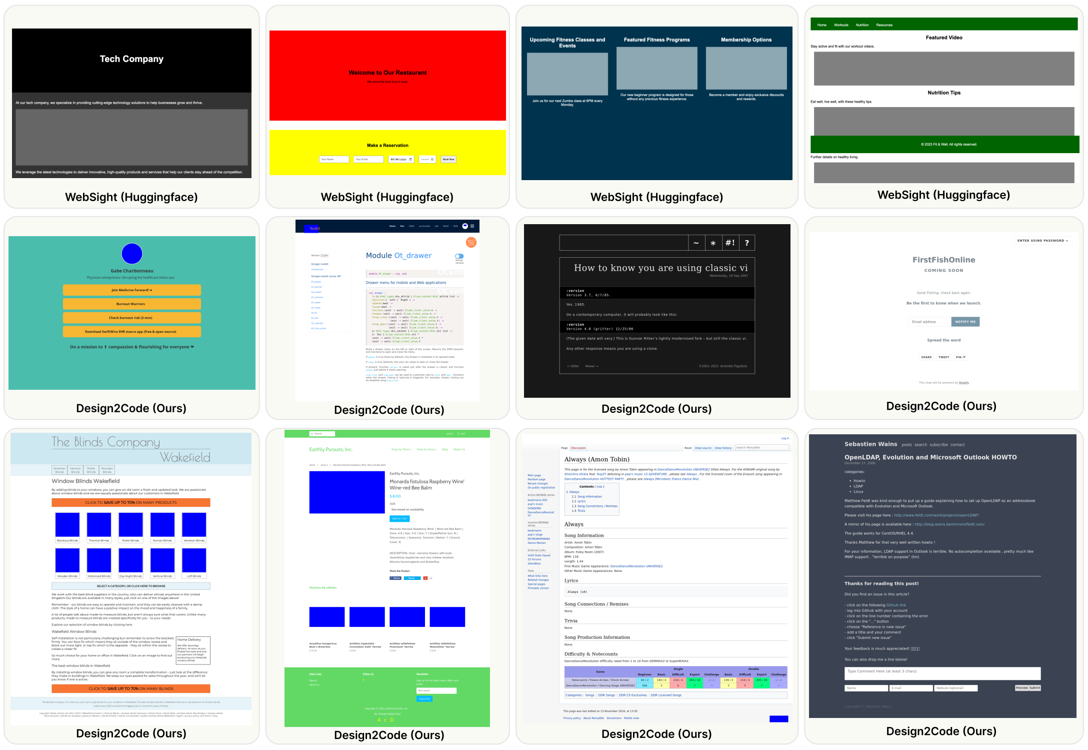


Figure 1: Examples from the prior WebSight dataset (first row) and our new Design2Code benchmark (last two rows). We use real-world webpages for benchmarking to ensure they are realistic and diverse, while WebSight uses synthetically generated webpages for scalability.

visual designs has the potential to democratize the development of front-end web applications [Nguyen and Csallner, 2015], allowing non-experts to build applications easily and quickly.

While code generation from natural language has advanced rapidly in recent years [Yin and Neubig, 2017, Le et al., 2020, Li et al., 2023b], generating code implementation from user interface (UI) design has not received much attention due to a wide range of challenges, such as diversity in visual and text signals on the user interface and the vast search space in the resulting code. Beltramelli [2018] made a notable attempt back in 2017 with CNN and RNN models on a narrow set of simplistic user interface designs. Over the years, despite many follow-up attempts along this quest [Robinson, 2019, Soselia et al., 2023], they are all constrained to simplistic or synthetic examples with a narrow set of layout designs, hardly useful for real-world front-end development applications. Until recently, the development of multimodal LLMs has entered a new era where large-scale pretrained models can process both visual and text input and generate text output for various visually grounded tasks, with representative examples being Flamingo [Alayrac et al., 2022], GPT-4V [OpenAI, 2023], and Gemini [Google, 2023]. Such advancement has unlocked a brand new paradigm for this long-standing unsolved task [1]: Take a screenshot of the user's website design and give this image to the system to obtain the full code implementation that can render into the desired webpage in a fully end-to-end manner. We term this task as **Design2Code** and tackle it with current multimodal models we have in our toolbox to benchmark and understand how far we are from automating front-end engineering.

Toward systematic and rigorous benchmarking, we construct the first-ever real-world benchmark for Design2Code (examples in Figure 1). To best reflect realistic use cases, we use real-world webpages in the wild as our test examples rather than synthetically generated ones as in prior works [Soselia et al., 2023, Huggingface, 2024]. We scrape webpages in the C4 [Raffel et al., 2019] validation set and perform careful manual curation over all examples to obtain a set of 484 high-quality, challenging, and diverse webpages representing a variety of real-world use cases with different level of complexities. We show both quantitatively and qualitatively that our benchmark covers a wide spectrum of HTML

[1]Greg Brockman's presentation during the GPT-4 release: `https://www.youtube.com/live/outcGtbnMuQ?si=5Yge32m5mnB85r4E&t=980`

tag uses, domains, and complexity levels. To facilitate efficient evaluation and model development, we also develop automatic metrics for this task that compare the generated webpage's screenshot with the given screenshot input. Our metrics consider a comprehensive set of dimensions including bounding box matches, text content, position, and color of all matched visual elements on the webpages, which we later show highly correlate with human judgment.

We then investigate how current multimodal LLMs like GPT-4V and Gemini perform on this task. To elicit their best capabilities, we introduce a variety of prompting methods, including our text-augmented prompting that complements visual input with extracted text elements from the webpage to reduce the load on OCR, as well as a self-revision prompting method that asks the model to compare its previous generation and the input webpage for self-improvement. We see consistent improvement from the text-augmented prompting method as compared to direct prompting on both GPT-4V and Gemini Pro, while only observing positive effect of self-revision on GPT-4V.

Despite demonstrating state-of-the-art performance, these commercial models are black boxes with limited transparency. To this end, we contribute an open-source 18B finetuned model for this task: `Design2Code-18B`. Concretely, we build upon the state-of-the-art open-source model CogAgent [Hong et al., 2023] and finetune it with synthetically generated Design2Code data [Huggingface, 2024]. Surprisingly, this "small" open-source model performs competitively on our benchmark, matching the performance of Gemini Pro Vision despite the discrepancy between synthetic training data and realistic testing data (an overview of automatic evaluation results is in Figure 3), indicating the potential of specialized "small" open models and skill acquisition from synthetic data.

To summarize, our contributions in this work include:

1. Formalize the Design2Code task and construct the manually curated `Design2Code` benchmark with 484 diverse real-world test examples.
2. Develop a comprehensive suite of automatic evaluation metrics that capture both high-level visual similarity and low-level element matching, which complement the human evaluation.
3. Propose new multimodal prompting methods that improve over direct prompting baselines.
4. Finetune our open-source `Design2Code-18B` model that matches the performance of Gemini Pro Vision as judged by both human and automatic evaluation.

2 The `Design2Code` Benchmark

In this section, we describe the curation and processing of our benchmark data. We first scrape all website links in the C4 [Raffel et al., 2019] validation set. We then embed all CSS code into the HTML file to obtain one single code implementation file for each webpage. This results in a total of 127.9k webpages, which we perform further filtering and processing as described below.

2.1 Test Set Curation

Our overall goal is to obtain a set of well-formed webpages that represent diverse real-world use cases. We follow the following steps for automatic processing and manual filtering.

Automatic Length and Layout Filtering We first apply a round of automatic filtering. We strip all comments from the code files and then apply a length filter to exclude examples where the source code file has over 100k tokens (based on the `GPT-2` tokenizer), as a way to avoid excessively long webpages that current multimodal LLMs cannot process as input or cannot decode such long outputs. Next, we filter all webpages whose layout consists of only images or only texts, in which cases the layout designs tend to be too simplistic to be interesting for benchmarking. This results in 14k webpages after filtering and deduplication.

Making Webpages Stand-alone We assume a setting where we will only provide the screenshot of the webpage for the model, without providing all the external dependencies such as multimedia files (images, audio, videos, etc.). To make this possible, we strip all such external file dependencies to make all the webpages stand-alone, this includes: removing all `<script><audio><iframe><map><svg>` tags, removing all `<link>` tags that link to external sites, removing all href links in `<a>` tags, and removing all external files in `<object>` elements. For all the

	WebSight (Huggingface)	**Design2Code (Ours)**
Purpose	Training	Testing
Source	Synthetic (Deepseek-Coder)	Real-World (C4)
Size	823K	484
Avg Length (tokens)	647±216	31216±23902
Avg Tag Count	19±8	158±100
Avg DOM Depth	5±1	13±5
Avg Unique Tags	10±3	22±6

Table 1: Comparison of datasets statistics between the WebSight dataset and our new Design2Code benchmark. WebSight only provides the training set while Design2Code only provides the test set. Examples in our Design2Code benchmark are much more complex on all measures and have a wider variety of difficulty levels as indicated by the bigger standard deviations.

image and video files, we replace them with a placeholder file, and during benchmarking we will instruct the models to insert this placeholder file wherever applicable to preserve the original layout.

Manual Curation After the above processing, we perform a final round of manual curation to filter examples based on the following criteria: (1) The webpage has no external file dependency and can render in a stand-alone manner from the processed code file and provided placeholder image file. (2) The webpage does not contain any private, sensitive, or potentially harmful information (e.g., we removed profile pages from dating websites). (3) The rendered webpage is well-formatted (e.g., there should not be overlaps between different layout elements and the automatic processing above should not disrupt any part of the webpage design). The first two authors of this paper performed this curation step by checking every single example from the sampled 7k examples. They first annotated 200 examples together to reach an 75% agreement, then split the annotation work on 7k randomly sampled examples from the filtered set of 14k examples above. This entire manual curation process took approximately one week. We tend to be more aggressive in the manual filtering process to only keep high-quality webpages with a variety of HTML and CSS elements involved. In the end, we obtained 484 test examples that we use as our benchmark.

2.2 Data Statistics and Diversity

Quantitative Metrics To provide an estimate of the difficulty levels of the test examples, we provide some quantitative measures. We compare with the most recent and most similar existing dataset – WebSight [Huggingface, 2024] in Table 1. **(1) Length**: We tokenize the scraped code files with the `GPT-2` tokenizer. The average number of tokens per file is 31215.6 (min=784, max=98637, std=23902.9). This is much longer than WebSight the typical max output length of modern language models, posing a unique challenge (although we note that the code files here are only for the reference implementation, there can be much more succinct ways to reproduce the given webpages). **(2) Total number of tags**: We count the total number of HTML tags involved, which is 158.3 on average (min=12, max=528, std=100.4). The examples in our benchmark cover 84 types of standard HTML5 tags. We present a chart of the most frequently used tags in Table 5 (Appendix A). **(3) DOM tree depth**: We measure the depth of the Document Object Model (DOM) tree as another measure of complexity. The average depth is 12.5 (min=4, max=32, std=4.7). **(4) Number of unique tags**: Lastly, we compute the number of unique HTML tags in each example, and the mean is 22.2 (min=8, max=45, std=6.0), suggesting that our benchmark covers a wide range of HTML tags. Overall, the examples in our benchmark are much more challenging and cover a wider spectrum of complexities than prior efforts like WebSight.

Domain Distribution To get a sense of the range of domains covered in our benchmark, we randomly sample 25% examples (N=120) from the benchmark and manually annotate what type of webpages they are based on their functions. We present the pie chart of the most frequent domains in Figure 2. The most prominent genres are websites of companies or organizations, personal blogs (including technical blogs), and personal homepages. Other genres include information-sharing sites (e.g., Wikipedia pages, FAQ pages, tax policy pages, online dictionaries), online forums, news article pages, and product description pages. Sampled examples are shown in Figure 1.

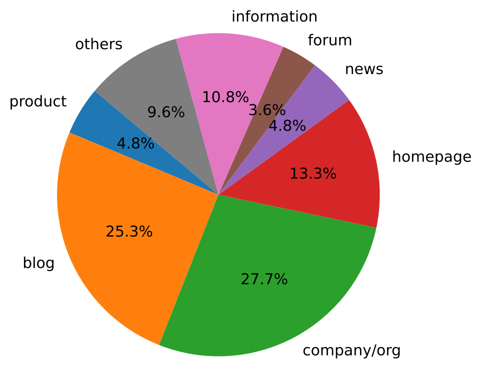


Figure 2: Main topics of the webpages in the Design2Code benchmark.

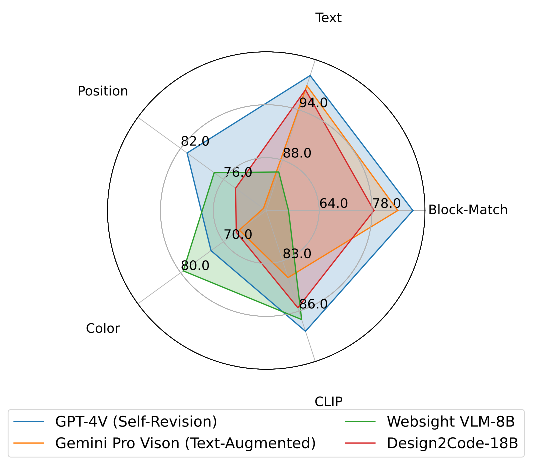


Figure 3: Performance of the benchmarked models broken down by our fine-grained metrics.

2.3 Automatic Metrics

Previously, generated HTML codes are usually evaluated by text-based similarity metrics, such as Normalized Edit Distance [Lv et al., 2023] and htmlBLEU [Soselia et al., 2023]. However, such metrics cannot directly assess whether the visual design of the original screenshot is correctly generated as there can be many different ways of implementing the same webpage, and minor differences in generated code could result in major visual differences in the rendered output. To this end, we propose to automatically evaluate generated webpages by calculating the similarity between the screenshots of reference webpages I_R and the rendered screenshots of generated webpages I_G. We break down the evaluation into both high-level visual similarity and low-level element matching.

High-level Visual Similarity To evaluate the visual similarity of I_R and I_G, we use the similarity of their CLIP [Radford et al., 2021] embedding, denoted as $\textbf{CLIP}(I_R, I_G)$. Specifically, we extract features by `CLIP-ViT-B/32` after resizing screenshots to squares. To rule out the texts in the screenshots, we use the inpainting algorithm from Telea [2004] to mask all detected text boxes using their bounding box coordinates. [2]

Low-level Element Matching Metrics like CLIP similarity only capture the similarity of the overall images rather than the matching of all the details like text. Moreover, the metric itself does not offer any fine-grained breakdown to help diagnose model weaknesses. To complement that, we introduce a suite of element-matching metrics. Specifically, we consider whether the generated webpages manage to recall all visual elements, and whether the corresponding visual elements in the reference and generated webpages have aligned text content, position, and color.

Given a reference webpage screenshot I_R and a generated webpage screenshot I_G, we use a text detection module to output a set of detected visual element blocks for each: $R = \{r_1, r_2, ..., r_m\}$ and $G = \{g_1, g_2, ..., g_n\}$, where each block contains its textual content and bounding box coordinates. See Appendix B for the details of implementing the block detection module. Based on the two sets of detected blocks, we use the Jonker-Volgenant algorithm [Crouse, 2016] to get the optimal matching M between R and G based on text similarity, where $(p, q) \in M$ indicates r_p is matched with g_q. Given R, G, and matched pairs in M, we evaluate similarity along the following aspects:

- **Block-Match**: The first desideratum of the task is that all visual elements from the reference webpage should be reproduced in the generated webpage, and the generated webpage should not hallucinate non-existent new elements. We measure this by computing the total sizes of all matched blocks divided by the total sizes of all blocks, including unmatched ones (either because the generated webpages missed them or because the generated webpages contain hallucinated blocks):

[2] `https://docs.opencv.org/4.3.0/df/d3d/tutorial_py_inpainting.html`

$$\mathbf{match_{block}}(r_p, g_q) = \frac{S(r_p) + S(g_q)}{\sum_{(i,j)\in M}(S(r_i) + S(g_j)) + (\sum_{i\in U_R} S(r_i) + \sum_{j\in U_G} S(g_j))},$$

$$\mathbf{match_{block}}(R, G) = \sum_{(p,q)\in M} \mathbf{match_{block}}(r_p, g_q),$$

where $S(\cdot)$ returns the size of the blocks, U_R and U_G denotes the unmatched blocks in R and G. The intuition here is that unmatched blocks will lower the score as they indicate missing original blocks or generating hallucinated blocks, and the larger the unmatched blocks are, the lower this score is.

- **Text**: Given two strings from two matched blocks r_p and g_q, the text similarity $\mathbf{sim_{text}}(r_p, g_q)$ is calculated as twice the number of overlapping characters divided by the total number of characters in the two strings (character-level Sørensen-Dice similarity). The overall score is averaged across all matched pairs.
- **Position**: The positioning of the blocks largely impacts the overall layout. For each matched pair (p, q), we calculate the position similarity $\mathbf{sim_{pos}}(r_p, g_q) = 1 - max(abs(x_q - x_p), abs(y_q - y_p))$, where (x_p, y_p) and (x_q, y_q) are normalized coordinates (in $[0, 1]$) of r_p and g_q's centors. The overall score is averaged across all matched pairs.
- **Color**: We use the CIEDE2000 color difference formula [Luo et al., 2001] to assess the perceptual difference between the colors of the generated text in block g_q and the reference text in block r_p, denoted as $\mathbf{sim_{color}}(r_p, g_q))$, where the formula considers the complexities of human color vision. The overall score is averaged across all matched pairs.

Note that we intentionally do not compute an aggregate score over these different dimensions because they are designed as fine-grained diagnostic scores. Ideally, models and methods should score well along all these dimensions.

3 Benchmarking: Prompting and Finetuning

We benchmark a variety of models and methods to compare their performance on our benchmark, including both prompting commercial API models and finetuning open-source models.

3.1 Multimodal Prompting Methods

As per modern deep learning practice, we resort to prompting commercial LLMs as the first set of baselines. We develop a suite of multimodal prompting methods for our benchmark. We assume access to a model that can take both image input and text prompts and then produce code as output. We will experiment with GPT-4V [OpenAI, 2023] and Gemini Pro Vision [Google, 2023] as the two best-performing publicly available APIs.

Direct Prompting We start with the most simple direct prompting baseline, We provide the reference webpage screenshot, along with the following instruction:

```
You are an expert web developer who specializes in HTML and CSS. A user will provide
you with a screenshot of a webpage.  You need to return a single html file that uses
HTML and CSS to reproduce the given website.  Include all CSS code in the HTML file
itself.  If it involves any images, use "rick.jpg" as the placeholder.  Some images
on the webpage are replaced with a blue rectangle as the placeholder, use "rick.jpg"
for those as well.  Do not hallucinate any dependencies to external files.  You
do not need to include JavaScript scripts for dynamic interactions.  Pay attention
to things like size, text, position, and color of all the elements, as well as the
overall layout.  Respond with the content of the HTML+CSS file.
```

Text-Augmented Prompting The above prompt asks the model to do everything at once: recognize all the text and layout elements and generate the corresponding code. In reality, users often have an idea of what content they want to put on their webpage. Instead, they are only looking for expertise in converting the design into code implementation. To reflect such a setting, we also explore a text-augmented prompting method, where we extract all text elements from the original webpage first [3] and append these texts after the instruction prompt along with the screenshot input. In this setting, we mitigate the difficulty of performing OCR and instead allow the model to focus more on layout design, where the model could copy text content from the prompt and insert it into the correct positions.

Self-Revision Prompting Inspired by recent works on using LLMs to self-improve their own generations [Madaan et al., 2023, Shinn et al., 2023], we also develop a self-revision prompt where we provide the following as input: (1) the screenshot of the input webpage, (2) the screenshot of the generated webpage from text-augmented prompting, (3) the generated code from text-augmented prompting as the initial solution; then we ask the model to improve the generated implementation code so that the result can look closer to the reference webpage (full prompt is in Appendix C).

We use the same prompts for both GPT-4V and Gemini Pro Vision, and we use the high-resolution mode with max output tokens 4096 and temperature 0.0 for all generations.

3.2 Finetuning `Design2Code`-18B

While commercial API models are performant and easy to use, they are opaque and limited in transparency. To enable open-source alternatives, we finetune an open-source model for this task and compare it with commercial API models.

Base Model We use CogAgent-18B Hong et al. [2023] as our base model, which supports high-resolution input (1120×1120) and is pretrained on intensive text-image pairs [Byeon et al., 2022, Schuhmann et al., 2022], synthetic documents [Kim et al., 2022], LaTeX papers [Blecher et al., 2023], and a small amount of website data.

Training Data We finetune the base model with the recently released Huggingface WebSight dataset [4], which consists of website screenshot and code implementation pairs. The dataset is generated in two steps: (i) Generate random website ideas from `Mistral-7B-v0.1` [Jiang et al., 2023]. (ii) Prompt `Deepseek-Coder-33b-Instruct` [Guo et al., 2024a] with the generated ideas to generate a simple and short website. While the original WebSight dataset has 823K examples, we only randomly sample 20% for training due to the limited computation resources. We also reverse the order of HTML style and body as we find that it lead to a lower loss in our preliminary experiment. Note that we have also experimented with training on real-world webpage data scraped from the C4 training set. Such training is extremely unstable and difficult because real-world code implementation data tend to be extremely long and noisy, resulting in even lower performance than training on synthetic data. We thus leave such exploration to future work.

Settings We use LoRA [Hu et al., 2021] to fine-tune the base model, where the LoRA modules are added to the language decoder with LoRA rank 8. Using a batch size of 32 and a learning rate of 1e-5, we fine-tune the model for 5000 steps with 100 steps warmup. Using $4\times$ NVIDIA A6000, this takes about 2 days of training. We use a temperature of 0.5 and a repetition penalty of 1.1 during inference and select the best checkpoint based on the average of all automatic metrics on a small dev set (20 examples).

Additional Baselines Apart from our own finetuned `Design2Code`-18B, we also compare with two other open-source baselines. For both baselines, we follow their default sampling settings. First, we compare it with the original CogAgent-18B model so that we can analyze the gains from finetuning. We use the screenshot and a simple prompt `Write the HTML code.` as the input. Second, we compare with the Huggingface WebSight VLM-8B, which is presumably (at least) finetuned on the full WebSight dataset, where the base models (SigLIP [Zhai et al., 2023] and Mistral-7B [Jiang et al., 2023]) are fully finetuned. Note that this is a beta version without any paper release or detailed documentation of the training details. We include this baseline purely for comprehensiveness despite it is not an apple-to-apple comparison with `Design2Code`-18B given the different base models and amount of training data.

[3]We use the Beautiful Soup library.

[4]`https://huggingface.co/datasets/HuggingFaceM4/WebSight`

	Block-Match	Text	Position	Color	CLIP
GPT-4V					
Direct Prompting	85.8	97.4	80.5	73.3	86.9
Text-Augmented Prompting	87.6	**98.2**	80.2	73.0	**87.2**
Self-Revision Prompting	**88.8**	98.1	**81.1**	72.9	**87.2**
Gemini Pro Vision					
Direct Prompting	80.2	94.6	72.3	66.2	83.9
Text-Augmented Prompting	84.8	96.9	70.4	66.3	84.0
Self-Revision Prompting	84.1	96.6	70.1	66.2	83.7
Open-Source Models					
WebSight VLM-8B	55.9	86.6	77.3	**79.4**	86.5
CogAgent-Chat-18B	7.1	18.1	13.3	13.0	75.5
`Design2Code-18B`	78.5	96.4	74.3	67.0	85.8

Table 2: Automatic evaluation results of the four fine-grained similarity measures as well as the high-level visual similarity with CLIP. The best result per dimension is highlighted in bold. GPT-4V is the best on all dimensions apart from color, on which WebSight VLM-8B is leading. Note that WebSight VLM-8B is finetuned on 5 times more data than our `Design2Code-18B`.

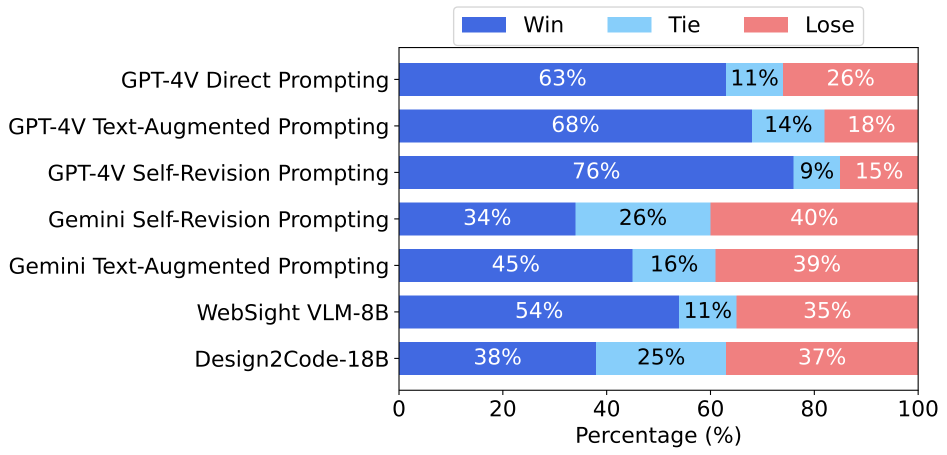


Figure 4: Human pairwise preference evaluation results with Gemini Pro Vision Direct Prompting as the baseline (this method itself is not shown in the table since it serves as the baseline for pairwise comparison). We sample 100 examples and ask 5 annotators for each pair of comparisons, and we take the majority vote on each example. Higher win rate and lower lose rate suggest best quality as judged by human annotators.

4 Results: Automatic and Human Evaluation

4.1 Automatic Evaluation

We present all automatic evaluation results in Table 2 and Figure 3. Note that the comparisons here are by no means fair comparisons, given the differences in model sizes and training data. We compare them as they are the most relevant and accessible baselines for our benchmark. We observe that: (1) GPT-4V is the best on all dimensions apart from color, on which WebSight VLM-8B is leading. (2) Text-augmented prompting successfully increases the block-match score and text similarity score on both GPT-4V and Gemini Pro Vision, indicating the usefulness of providing extracted text elements. (3) Self-revision has some minor improvement on block-match and position similarity for GPT-4V, but brings no improvement on Gemini Pro Vision, potentially due to the limited capabilities of LLMs to perform intrinsic self-correction without external feedback [Huang et al., 2023]. (4)

	coef	std err	p
Block-Match	2.0442	0.648	0.002
Text	1.3758	2.217	0.535
Position	7.8037	1.312	0.000
Color	2.0731	0.757	0.006
CLIP	10.2353	2.855	0.000

Table 3: Coefficients of predicting human annotations (Win/Lose) using logistic regression on automatic metric's differences of the same pair. Human evaluation mostly correlates with position and CLIP similarity.

Finetuning achieves huge improvement on all dimensions as indicated by the comparison between `Design2Code-18B` and the base version CogAgent-18B. (5) Our finetuned `Design2Code-18B` is better at block-match and text similarity, but worse at position similarity and color similarity as compared to WebSight VLM-8B. We could potentially attribute the first two to the stronger and larger base model and the latter two to the larger amount of finetuning data. We provide an in-depth analysis of the learning process of our finetuned model in Section 5.2.

4.2 Human Evaluation

While the above automatic metrics provide a fine-grained breakdown of model performance, it is also crucial to ask what humans, the ultimate audience of these webpages, think of the generated webpages. By recruiting human annotators (paid at the rate of $16/hour) from Prolific [5], we conducted a series of human evaluations to compare across models and methods, as well as to directly assess the quality of the best-performing model. We sample 100 examples from our benchmark for the human evaluations. In all human evaluations, each question is annotated by 5 human annotators, and we derive the results by majority voting. We provide all instructions that we provided to annotators in Appendix D and we outline the main protocols and results below.

Pairwise Model Comparison Following the conventional practice of evaluating instruction-following LLMs (e.g., [Zhou et al., 2023, Dubois et al., 2023]), we ask human annotators to rank a pair of generated webpages (one from the baseline, the other from the tested methods) to decide which one is more similar to the reference. We use Gemini Pro Vision Direct Prompting as the baseline and collect the other seven methods' Win/Tie/Lose rates against this baseline (we randomly shuffle the ordering to avoid position biases). Each pair will count as Win (Lose) only when Win (Lose) receives the majority vote (≥ 3). All other cases are considered Tie.

Based on the human evaluation in Figure 4, we find that: (1) GPT-4V is substantially better than other baselines, while both text-augmented prompting and self-revision prompting can further improve over direct prompting. (2) Text-augmented prompting can slightly improve the Gemini direct prompting baseline, while further adding self-revision is not helpful. Intuitively, self-revision needs the model to understand the differences between the two given images (the reference screenshot and the screenshot of the initial model generation) and reflect them correspondingly in the modified HTML code, which is harder than leveraging text augmentation and thus might require more advanced model capabilities. (3) WebSight VLM-8B performs better than Gemini direct prompting (54% win rate and 35% lose rate), suggesting that finetuning on a large amount of data can match commercial models in specific domains. (4) Our model `Design2Code-18B` matches the performance of Gemini Pro Vision direct prompting (38% win rate and 37% lose rate).

Direct Assessment While the automatic and human evaluation offer a comparison among different models and methods, readers might still wonder: "*How far are we from automating front-end engineering?*" To offer a more intuitive answer to this question, we further ask human annotators to compare each reference webpage with the best AI-generated webpage (using GPT-4V self-revision prompting). All examples are annotated by 5 annotators, and we take the majority vote. Full

[5]We restrict the annotators to people in the U.S. who have completed 2,500 surveys with a pass rate of 98% or higher.

instructions given to the annotators can be found in Appendix D. Concretely, we perform direct assessment from two perspectives:

1. **Can the AI-generated webpage replace the original webpage?** We shuffle the ordering of all examples and ask annotators to judge whether the two webpages are similar enough in terms of appearance and content so that they can be deployed interchangeably. We find that **49% of the AI-generated webpages are considered exchangeable with the reference webpages**.
2. **Is the reference webpage or AI generation better?** We then ask a different question, where we shuffle the example ordering and ask annotators which webpage is better designed (annotators do not know which one is the reference and which one is AI-generated). Perhaps surprisingly, **webpages generated by GPT-4V are preferred in 64% cases**, i.e., they are considered better designed than even the original reference webpages. We hypothesize it is possible that the model has more access to modern and popular webpage design principles [Ivory and Megraw, 2005, Beaird et al., 2020], such that it can automatically improve the original design based on these best practices. This also opens up many new opportunities for future work on website design improvement tools.

4.3 Automatic Evaluation vs Human Evaluation

It is worth noting that there are some interesting discrepancies between the automatic evaluation results and human evaluation results. For example, human evaluation ranks GPT-4V self-revision prompting better than text-augmented prompting, while the automatic metrics show mixed results. Moreover, even though humans rank WebSight VLM-8B as better than `Design2Code-18B`, it has much worse block-match and text similarity as measured by the automatic metrics. In this part, we take a closer look at such discrepancy and discuss why such discrepancy is a feature rather than a bug.

We study the correlation between the automatic metrics and human pairwise preferences. Specifically, we randomly split 588 pairwise human annotations (Win/Lose only) into a 50% training set and a 50% test set. Given one reference R and two candidates G_1, G_2, we use the difference of each dimension (e.g., $\mathbf{match_{block}}(R, G_1) - \mathbf{match_{block}}(R, G_2)$) as features and predict Win (1) or Lose (0) by logistic regression. The derived logistic regression model achieves 76.9% accuracy, and the features' coefficients and significance are in Table 3. We find that text similarity has almost no correlation with humans. In contrast, the position similarity of matched blocks and the CLIP similarity are the two most correlated automatic metrics. This suggests that **humans usually pay more attention to high-level visual effects and the layout rather than the detailed content, reflecting the top-down processing [Gilbert and Li, 2013] of humans**. In summary, we argue that human evaluation should not be blindly trusted as the oracle here due to their cognitive bias to only consider "principle components" of the webpages. Instead, both high-level similarity (human pairwise preference and CLIP similarity) and low-level elements (fine-trained block-wise similarity) should be taken into consideration when evaluating new models and methods, and ideally, they should score well on both fronts.

5 Analysis

5.1 What Makes A Webpage Difficult?

To understand what makes a webpage difficult to generate, we compute the correlation between automatic metrics and various difficulty indicators, including: (1) total number of tags in the reference implementation; (2) number of unique tags in the reference implementation; and (3) DOM tree depth of the reference implementation. Table 4 shows that the total number of tags is a strong indicator of the difficulty, where webpages with more tags tend to have lower scores along all fine-grained dimensions.

5.2 What is the Learning Process of Different Dimensions?

We further plot the learning process of different automatic evaluation dimensions in Figure 5 to help us better understand the performance differences in Table 2. Specifically, we show the normalized performance of each aspect (so that 0 before training and 1 after training) for the base model

Total Num of Tags		Num of Unique Tags		DOM Tree Depth	
Metric	Corr	Metric	Corr	Metric	Corr
Block-Match	-0.28*	Block-Match	-0.16*	Block-Match	-0.04
Text	-0.13*	Text	-0.08	Text	0.01
Position	-0.19*	Position	-0.15*	Position	-0.10*
Color	-0.13*	Color	-0.09	Color	-0.04
CLIP	-0.12	CLIP	-0.02	CLIP	0.03

Table 4: Correlation between automatic metrics and three proxy difficulty indicator variables on GPT-4V self-revision prompting. The total number of tags is the strongest indicator, where webpages with more tags tend to be more challenging for the model. * indicates p-value < 0.05.

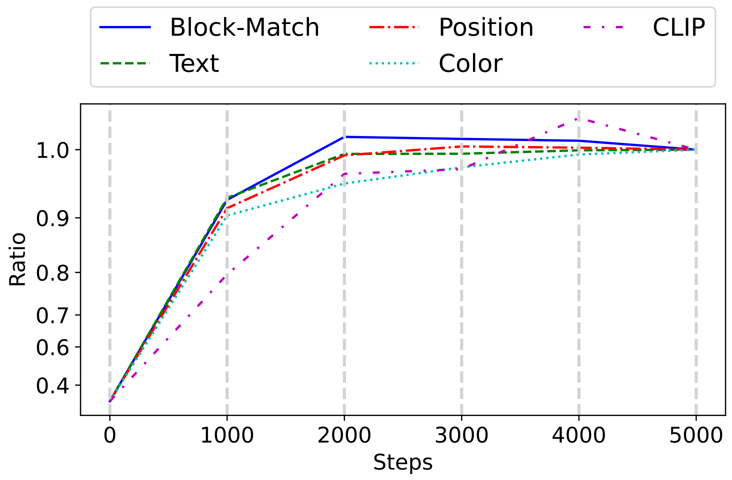


Figure 5: Learning process for different automatic evaluation dimensions, where we plot the performance for the base model checkpoint and all training checkpoints. For each dimension, the score is re-scaled so that it is 0 before training (0 steps) and 1 after training (5000 steps). The y-axis is rescaled to highlight the differences for bigger values.

checkpoint and all training checkpoints. On the one hand, performance on block-match, text, and position quickly saturate after training for 2000 steps and remain stable afterward, possibly because these are the most transferable capabilities from the base model. On the other hand, the color similarity and the CLIP similarity steadily increase until $4000 - 5000$ steps. We assume that generating the correct color codes for texts and backgrounds benefits more from the HTML training data than other aspects and might be further improved by using the full Websight dataset and fully finetuning.

5.3 Qualitative Analysis

We manually check through examples where our proposed text-augment prompting and self-revision prompting achieve improvement on GPT-4V.

Examples of text-augmented prompting improving over direct prompting We find that text-augmented prompting mostly improves over direct prompting by having higher recall in the generated content, especially texts, as exemplified in Figure 6, where the output from direct prompting misses most of the text contents but text-augmented prompting recovers them, improving the block-match score from 0.25 to 0.84.

Examples of self-revision prompting improving over text-augmented prompting We next analyze examples where self-revision improves upon the initial generations from text-augmented prompting. We find two main sources of improvement. The first example in Figure 7 shows that case where self-revision brings back missing elements from the webpage, increasing the block-match

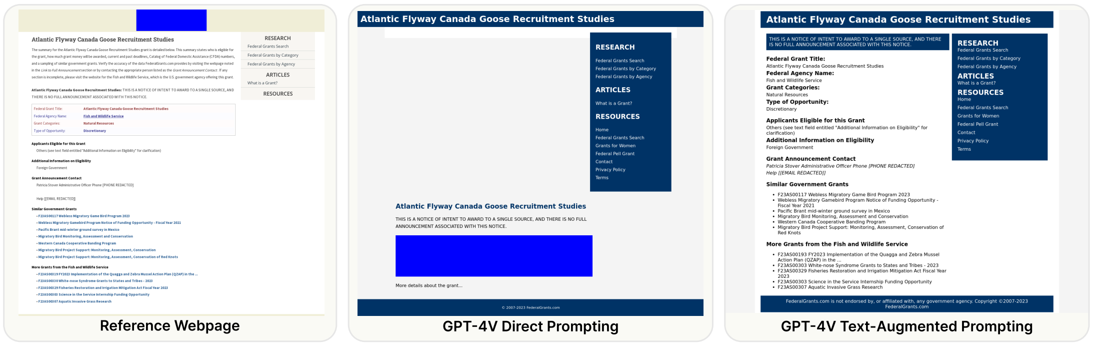


Figure 6: Example of text-augmented prompting improving over the direct prompting baseline, where missing texts are successfully generated.

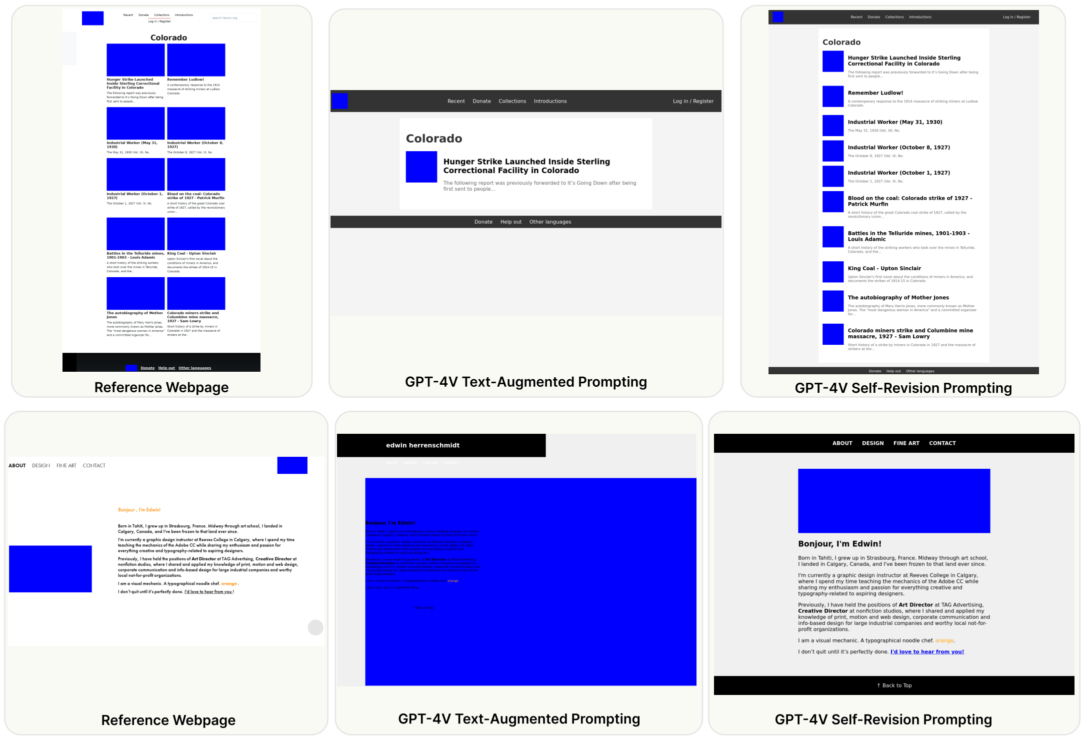

Reference Webpage
GPT-4V Text-Augmented Prompting
GPT-4V Self-Revision Prompting
Reference Webpage
GPT-4V Text-Augmented Prompting
GPT-4V Self-Revision Prompting

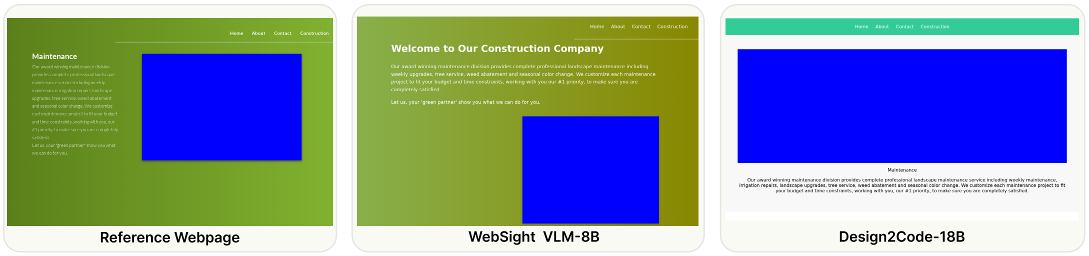
Figure 7: Examples of self-revision prompting improving over text-augmented prompting. The self-revision can either add the missing texts or fix layout errors.

score from 0.48 to 1.00, and consequently CLIP similarity from 0.87 to 0.91. The second example in Figure 7 shows the case where layout errors are fixed through self-revision, improving the overall CLIP similarity from 0.85 to 0.91.

WebSight VLM-8B vs Design2Code-18B We show a representative example in Figure 8, where WebSight VLM-8B is much better in coloring than `Design2Code-18B` (color score 0.99 vs 0.66) and overall layout (position score 0.91 vs 0.63 and CLIP similarity 0.90 vs 0.83). However, WebSight VLM-8B tends to hallucinate texts and results in lower block-match (0.85 vs 0.99) and text similarity scores (0.98 vs 1.0). In general, we find that WebSight VLM-8B tends to have lower precision and recall than our model in terms of text matching.

6 Related Work

Multimodal LLMs To enable multimodal understanding and grounded generation, multimodal LLMs are usually augmented with an extra encoder to accept multimodal input, prominently visual



Maintenance

Our award winning maintenance division provides complete professional landscape maintenance service including weekly maintenance, irrigation repairs, landscape upgrades, tree service, weed abatement and seasonal color change. We customize each maintenance project to fit your budget and time constraints, working with you, our #1 priority, to make sure you are completely satisfied.

Let us, your "green partner" show you what we can do for you.

Welcome to Our Construction Company

Our award winning maintenance division provides complete professional landscape maintenance including weekly upgrades, tree service, weed abatement and seasonal color change. We customize each maintenance project to fit your budget and time constraints, working with you our #1 priority, to make sure you are completely satisfied.

Let us, your 'green partner' show you what we can do for you.

Maintenance

Our award winning maintenance division provides complete professional landscape maintenance service including weekly maintenance, irrigation repairs, landscape upgrades, tree service, weed abatement and seasonal color change. We customize each maintenance project to fit your budget and time constraints, working with you, our #1 priority, to make sure you are completely satisfied.

Figure 8: Comparison of WebSight VLM-8B and Design2Code-18B. WebSight VLM-8B excels at color recognition but hallucinates text contents.

input. As a representative example, BLIP-2 [Li et al., 2023a] connects ViT [Dosovitskiy et al., 2020] with large language models with Q-Former. To further improve generalization capability on unseen tasks, instruction tuning is introduced to multimodal LLMs, where LLaVA [Liu et al., 2023] generates complex image-based QA based on prompting GPT-4 [OpenAI, 2023] with COCO captions and InstructBLIP [Dai et al., 2023] transform 13 datasets into the same format of instruction-following. Ye et al. [2023] further scales up the pretraining data while Bai et al. [2023] includes grounding and OCR data into the multitask finetuning. While commercial models like GPT-4V have demonstrated promising performance on a wide range of vision-language tasks, Yan et al. [2023], Zhang et al. [2023], Zheng et al. [2024] adapt them to operate smartphone UIs and websites. Our works offers a new challenging benchmark to assess the capabilities in the realistic front-end engineering task.

UI Code Generation Nguyen and Csallner [2015] reverse engineer mobile UI by identifying elements through classic text recognition and computer vision techniques (OCR, edge detection, etc) and generating code on top of them. Pix2Code [Beltramelli, 2018] builds an end-to-end system for UI-to-code transformation based on CNN and RNN, which faces the challenge of complex visual encoding and long text decoding while dealing with real-world UIs. Robinson [2019], Aşıroğlu et al. [2019] further incorporate neural network-based object detection and semantic segmentation into the pipeline. Recently, Soselia et al. [2023] utilize advanced visual encoders (e.g., ViT, Dosovitskiy et al., 2020) and language decoders (e.g., LLaMA, Touvron et al., 2023a,b) and finetune the pipeline using visual similarity as the signal. However, their training and testing examples mainly contain a small number of simple elements (e.g., a square, a circle, a button).

Code LLMs and Programming Support Tools Our work also connects to code language models and programming support tools. LLMs trained on code, such as Codex [Chen et al., 2021], StarCoder Li et al. [2023b], InCoder [Fried et al., 2022], CodeLlama [Rozière et al., 2023], and DeepSeek-Coder [Guo et al., 2024b], enable a wave of programming support applications such as automatic code completion and infilling, and allowing users to chat with a codebase [6]. This also leads to a new wave of HCI studies on how to design better programming tools to facilitate human-AI collaboration [Kalliamvakou, 2022, Vasconcelos et al., 2023, Liang et al., 2023]. Our benchmark offers realistic evaluation for code LLMs and aims to enable more powerful programming support to front-end designers who do not have to code by themselves and can just collaborate with LLMs.

7 Conclusion and Future Work

In this work, we introduced the Design2Code benchmark consisting of diverse real-world webpages as test examples. We develop comprehensive automatic metrics and conduct a series of human evaluations to compare various multimodal code LLMs, showing that finetuned open-source models can match prompting Gemini Pro Vision, but still lag behind GPT-4V. Moreover, human annotators find 49% of the GPT-4V generations to be good enough to replace the original references, while 64% are judged as even better designed than the original references.

We believe Design2Code can serve as a useful benchmark to power many future research directions. We highlight a few of them:

[6]`https://github.com/features/copilot`

1. Better prompting techniques for multimodal LLMs, especially in handling complex webpages, for example by incrementally generating different parts of the webpage.
2. Training open multimodal LLMs with real-world webpages. Our preliminary experiments showed the difficulty of directly training on real webpages since they are too long and noisy, future work could explore data cleaning pipelines to make such training stable.
3. Extending beyond screenshot inputs, for example, to collect Figma frames or sketch designs from front-end designers as the test input. Such extension also requires careful re-design of the evaluation paradigm.
4. Extending from static webpages to also include dynamic webpages. This also requires the evaluation to consider interactive functions, beyond just visual similarity.

Ethical Considerations

Privacy We used the dataset C4 which is released under ODC-By license, allowing free share, modification, and use subject to the attribution requirements. We release our dataset under the same license. Moreover, when performing manual filtering, we explicitly filtered out webpages containing private or sensitive information (e.g., dating website profiles).

Dual Use Despite our intention of democratizing webpage building, we recognize the potential dual use danger of Design2Code technologies, such as automated generation of malicious websites, or even generating code for licensed websites. We emphasize is intended for research purposes and for the community to better understand multimodal LLM capabilities. We will provide clear ethical use guidelines for all data, code, and model releases to define acceptable and unacceptable use cases.

Acknowledgement

We thank Aryaman Arora, Jihyeon Je, Irena Gao, Will Held, Ryan Louie, Weiyan Shi, Dora Zhao, Rose Wang, Caleb Ziems, Michael Ryan, Camille Harris, Harshit Joshi, Yijia Shao, Jiaao Chen, Omar Shaikh, Julie Kallini, Lucia Zheng, Julia Kruk, Tianyu Gao and Tristan Thrush for their helpful comments and discussion.

References

J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan. Flamingo: a visual language model for few-shot learning. *ArXiv*, abs/2204.14198, 2022. URL `https://api.semanticscholar.org/CorpusID:248476411`.

B. Aşıroğlu, B. R. Mete, E. Yıldız, Y. Nalçakan, A. Sezen, M. Dağtekin, and T. Ensari. Automatic html code generation from mock-up images using machine learning techniques. In *2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT)*, pages 1–4, 2019. doi: 10.1109/EBBT.2019.8741736.

J. Bai, S. Bai, S. Yang, S. Wang, S. Tan, P. Wang, J. Lin, C. Zhou, and J. Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond, 2023.

J. Beaird, A. Walker, and J. George. *The principles of beautiful web design*. SitePoint Pty Ltd, 2020.

T. Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 1–6, 2018.

L. Blecher, G. Cucurull, T. Scialom, and R. Stojnic. Nougat: Neural optical understanding for academic documents, 2023.

M. Byeon, B. Park, H. Kim, S. Lee, W. Baek, and S. Kim. Coyo-700m: Image-text pair dataset. `https://github.com/kakaobrain/coyo-dataset`, 2022.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Ponde, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. W. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, I. Babuschkin, S. Balaji, S. Jain, A. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021. URL `https://api.semanticscholar.org/CorpusID:235755472`.

D. F. Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016. doi: 10.1109/TAES.2016.140952.

W. Dai, J. Li, D. Li, A. M. H. Tiong, J. Zhao, W. Wang, B. Li, P. Fung, and S. Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

Y. Dubois, X. Li, R. Taori, T. Zhang, I. Gulrajani, J. Ba, C. Guestrin, P. Liang, and T. Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *ArXiv*, abs/2305.14387, 2023. URL `https://api.semanticscholar.org/CorpusID:258865545`.

D. Fried, A. Aghajanyan, J. Lin, S. I. Wang, E. Wallace, F. Shi, R. Zhong, W. tau Yih, L. Zettlemoyer, and M. Lewis. Incoder: A generative model for code infilling and synthesis. *ArXiv*, abs/2204.05999, 2022. URL `https://api.semanticscholar.org/CorpusID:248157108`.

C. D. Gilbert and W. Li. Top-down influences on visual processing. *Nature Reviews Neuroscience*, 14(5):350–363, 2013.

Google. Gemini: A family of highly capable multimodal models. *ArXiv*, abs/2312.11805, 2023. URL `https://api.semanticscholar.org/CorpusID:266361876`.

D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024a.

D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024b.

W. Hong, W. Wang, Q. Lv, J. Xu, W. Yu, J. Ji, Y. Wang, Z. Wang, Y. Zhang, J. Li, B. Xu, Y. Dong, M. Ding, and J. Tang. Cogagent: A visual language model for gui agents, 2023.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021.

J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou. Large language models cannot self-correct reasoning yet. *ArXiv*, abs/2310.01798, 2023. URL `https://api.semanticscholar.org/CorpusID:263609132`.

Huggingface. Huggingface websight, 2024. URL `https://huggingface.co/datasets/HuggingFaceM4/WebSight`.

M. Y. Ivory and R. Megraw. Evolution of web site design patterns. *ACM Transactions on Information Systems (TOIS)*, 23(4):463–497, 2005.

A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023.

E. Kalliamvakou. Quantifying github copilot's impact on developer productivity and happiness, 2022.

G. Kim, T. Hong, M. Yim, J. Nam, J. Park, J. Yim, W. Hwang, S. Yun, D. Han, and S. Park. Ocr-free document understanding transformer. In *European Conference on Computer Vision*, pages 498–517. Springer, 2022.

T. H. Le, H. Chen, and M. A. Babar. Deep learning for source code modeling and generation: Models, applications, and challenges. *ACM Computing Surveys (CSUR)*, 53(3):1–38, 2020.

J. Li, D. Li, S. Savarese, and S. Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023a.

R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M.-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. M. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries. Starcoder: may the source be with you! *ArXiv*, abs/2305.06161, 2023b. URL `https://api.semanticscholar.org/CorpusID:258588247`.

J. T. Liang, C. Yang, and B. A. Myers. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *International Conference on Software Engineering*, 2023. URL `https://api.semanticscholar.org/CorpusID:257833548`.

H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning, 2023.

M. R. Luo, G. Cui, and B. Rigg. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur*, 26(5):340–350, 2001.

T. Lv, Y. Huang, J. Chen, L. Cui, S. Ma, Y. Chang, S. Huang, W. Wang, L. Dong, W. Luo, S. Wu, G. Wang, C. Zhang, and F. Wei. Kosmos-2.5: A multimodal literate model, 2023.

A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Welleck, B. P. Majumder, S. Gupta, A. Yazdanbakhsh, and P. Clark. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651, 2023. URL `https://api.semanticscholar.org/CorpusID:257900871`.

S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of ocr research and development. *Proceedings of the IEEE*, 80(7):1029–1058, 1992.

T. A. Nguyen and C. Csallner. Reverse engineering mobile application user interfaces with remaui (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 248–259. IEEE Computer Society, 2015.

OpenAI. Gpt-4v(ision) system card. 2023. URL `https://api.semanticscholar.org/CorpusID:263218031`.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019.

A. Robinson. Sketch2code: Generating a website from a paper mockup. *ArXiv*, abs/1905.13750, 2019. URL `https://api.semanticscholar.org/CorpusID:173188440`.

B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. P. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. D'efossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code llama: Open foundation models for code. *ArXiv*, abs/2308.12950, 2023. URL `https://api.semanticscholar.org/CorpusID:261100919`.

C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, P. Schramowski, S. Kundurthy, K. Crowson, L. Schmidt, R. Kaczmarczyk, and J. Jitsev. Laion-5b: An open large-scale dataset for training next generation image-text models, 2022.

N. Shinn, F. Cassano, B. Labash, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. 2023. URL `https://api.semanticscholar.org/CorpusID:258833055`.

D. Soselia, K. Saifullah, and T. Zhou. Learning ui-to-code reverse generator using visual critic without rendering. 2023. URL `https://api.semanticscholar.org/CorpusID:265302631`.

A. Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004.

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023a.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

H. Vasconcelos, G. Bansal, A. Fourney, Q. V. Liao, and J. W. Vaughan. Generation probabilities are not enough: Exploring the effectiveness of uncertainty highlighting in ai-powered code completions. *ArXiv*, abs/2302.07248, 2023. URL `https://api.semanticscholar.org/CorpusID:256846746`.

A. Yan, Z. Yang, W. Zhu, K. Lin, L. Li, J. Wang, J. Yang, Y. Zhong, J. McAuley, J. Gao, et al. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023.

Q. Ye, H. Xu, G. Xu, J. Ye, M. Yan, Y. Zhou, J. Wang, A. Hu, P. Shi, Y. Shi, C. Li, Y. Xu, H. Chen, J. Tian, Q. Qi, J. Zhang, and F. Huang. mplug-owl: Modularization empowers large language models with multimodality, 2023.

P. Yin and G. Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017.

X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. Sigmoid loss for language image pre-training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2023. doi: 10.1109/iccv51070.2023.01100. URL `http://dx.doi.org/10.1109/ICCV51070.2023.01100`.

C. Zhang, Z. Yang, J. Liu, Y. Han, X. Chen, Z. Huang, B. Fu, and G. Yu. Appagent: Multimodal agents as smartphone users, 2023.

B. Zheng, B. Gou, J. Kil, H. Sun, and Y. Su. Gpt-4v(ision) is a generalist web agent, if grounded, 2024.

C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, S. Zhang, G. Ghosh, M. Lewis, L. Zettlemoyer, and O. Levy. Lima: Less is more for alignment, 2023.

Tag	Frequency	Tag	Frequency	Tag	Frequency
<div>	17790	**<style>**	1181	**<head>**	486
<a>	13309	**<td>**	997	**<body>**	486
<li>	6883	**<input>**	995	**<tr>**	436
<span>	6813	**<h3>**	759	**<b>**	429
<meta>	4629	**<h2>**	709	**<nav>**	416
<p>	3413	**<strong>**	595	**<i>**	400
** **	2453	**<h1>**	536	**<section>**	381
<ul>	2078	**<button>**	525	**<label>**	339
<img>	1870	**<title>**	492	**<form>**	292
<option>	1194	**<html>**	486	**<h4>**	289

Table 5: The most frequent HTML tags in the reference implementations of our benchmark examples.

A Additional Dataset Statistics

We present the table of most frequent HTML tags in Table 5.

B Text Detection and Merging Details

The common approach to detect the texts in a given screenshot is to use OCR tools [Mori et al., 1992], which returns a list of text segments with their bounding boxes. However, in our case, we find that open-source OCR tools usually output noisy outputs, which may affect the stability of downstream evaluation. Since we already have the source HTML codes for reference webpage screenshots, we apply an alternative approach: we alter the color differently for different text segments in the source HTML code and detect text segments in the webpage by taking two extra screenshots and tracking pixels with different colors. This helps us locate text segments from the HTML source code in the screenshots without text recognition errors.

Based on the two sets of detected blocks, we use the Jonker-Volgenant algorithm [Crouse, 2016] (implemented in Scipy [7]) to get the optimal matching M between R and G, where $(p, q) \in M$ indicates r_p is matched with g_q. Specifically, we use the negative sequence similarity between textual contents ($-\mathbf{sim}_{\mathbf{text}}(,)$) to initialize the cost matrix and ignore the matched pairs with a sequence similarity lower than 0.5. Since detected text blocks might be in different granularity, we also enumerate merging neighbor text blocks to search for matching with the highest similarity. However, the matching may still not be perfect, especially when there are large granularity differences (our search does not consider merging non-contiguous blocks).

C Prompts

We use the following prompt for self-revision prompting:

```
You are an expert web developer who specializes in HTML and CSS. I have
an HTML file for implementing a webpage but it has some missing or wrong
elements that are different from the original webpage.  The current
implementation I have is:  [generated code from text-augmented prompting].
I will provide the reference webpage that I want to build as well as the
rendered webpage of the current implementation.  I also provide you all
the texts that I want to include in the webpage here:  [extracted texts
from the original webpage].  Please compare the two webpages and refer to
the provided text elements to be included, and revise the original HTML
implementation to make it look exactly like the reference webpage.  Make
sure the code is syntactically correct and can render into a well-formed
webpage.  You can use "rick.jpg" as the placeholder image file.  Pay
```

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html

```
attention to things like size, text, position, and color of all the
elements, as well as the overall layout.  Respond directly with the content
of the new revised and improved HTML file without any extra explanations.
```

D Human Annotation Details

In the instructions, the annotators are asked to check the pair following the order of priority (content > layout > style). This priority list is based on two intuitions: (i) Layout comparison is only meaningful when the content is (almost) complete. (ii) The style of independent elements is easier to fix than the layout of multiple elements. The detailed instructions are below:

Task Overview
In this survey, you will be given a reference webpage's screenshot, as well as two candidate webpages (Example 1 and Example 2) that try to replicate the reference webpage. Your task is to judge which of the two candidates is closer to the reference. Each (Reference, Example 1, Example 2) is presented in a row, where the original boundary of screenshot is marked by black.
Comparison Guide
Initial Step: Content Check

- ***Text Content:*** *Examine if the text on the candidate webpages matches the reference. Pay special attention to missing or extra content, especially key elements like titles.*

- ***Image Content:*** *Assess the placement of the blue placeholder blocks (for images).*

- ***Primary Judgment Criterion:*** *If one example has significant missing or additional content compared to the other, it should be considered less similar to the reference.*

Second Step: Layout Check

- ***Element Arrangement:*** *If the content (text and images) of both examples is similarly good or bad, proceed to evaluate the arrangement of these elements. Check if their organization, order, and hierarchy match the reference.*

- ***Secondary Judgment Criterion:*** *If differences in layout are observed, the example with the layout most similar to the reference should be rated higher.*

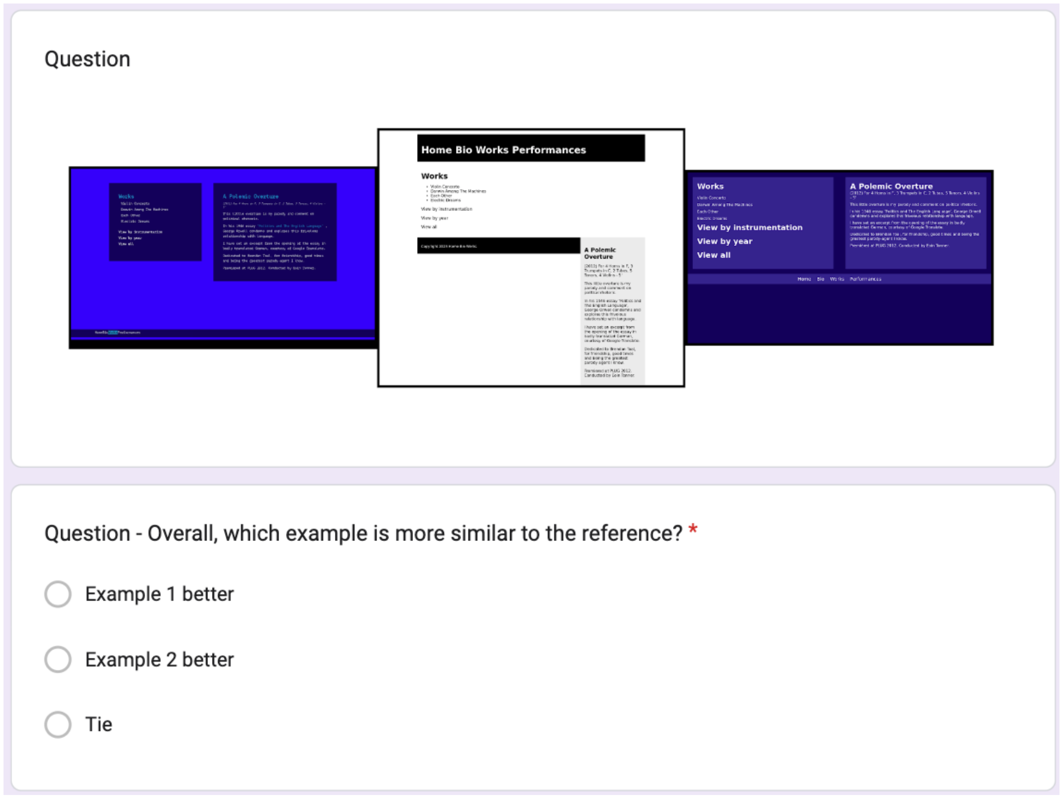
Final Step: Style Check

- ***Style Attributes:*** *Only if Example 1 and Example 2 are comparable in content and layout, examine the style elements like font style, color, and size.*

- ***Tertiary Judgment Criterion:*** *In cases where content and layout are equally matched, preference should be given to the example with style attributes closer to the reference.*

Overall Judgment
Based on the criteria in the order of priority (Content > Layout > Style), make an overall judgment on which example (Example 1 or Example 2) is more similar to the reference webpage.
Judgment Options
1. Select "Example 1 better" if Example 1 is closer to the reference.
2. Select "Example 2 better" if Example 2 is closer to the reference.
3. Opt for "Tie" only if both examples are similarly accurate or equally distant from the reference.
Additional Tips
1. Use zoom-in for detailed inspection.
2. Focus on major discrepancies in each step before moving to the next.
3. Your judgment should be based on a cumulative assessment of content, layout, and style.

We also provide 8 examples after the instruction. The UI of the annotation question is Figure 9. Fleiss' kappa for pairwise model comparison is 0.36 (5 annotators).


Question

Question - Overall, which example is more similar to the reference? *

- ◯ Example 1 better
- ◯ Example 2 better
- ◯ Tie

Figure 9: User Interface for pairwise model comparison.

Furthermore, we provide the instructions for direct assessment (comparing the reference and webpages generated by GPT-4V self-revision prompting). The Fleiss' kappa is 0.32 (5 annotators) for the first question and 0.26 (5 annotators) for the second question.

Can the AI-generated webpage replace the original webpage?

> ***Task Overview***
> *In each question, you will be given two webpage screenshots.*
> ***By comparing the two webpages, you need to decide whether they are exchangeable.***
> *Please zoom in to take a closer look at the screenshots if necessary.*
> *You should answer "Yes", if:*
> *1. They look roughly similar.*
> *2. They have similar content.*
> *3. They can serve the same functions.*
> *(Minor details don't matter that much)*
> *Otherwise, you should answer "No".*

Is the reference webpage or AI generation better?

> ***Task Overview***
> *In each question, you will be given two webpage screenshots.*
> ***By comparing the two webpages, you need to decide which one is better.***
> *Please zoom in to take a closer look at the screenshots if necessary.*
> *To decide which one is better, you might consider the following aspects:*
> *1. More readable*

2. Better layout
3. Better style