

# WebCode2M: A Real-World Dataset for Code Generation from Webpage Designs

## Abstract

Automatically generating webpage code from webpage designs can significantly reduce the workload of front-end developers, and recent *Multimodal Large Language Models* (MLLMs) have shown promising potential in this area. However, our investigation reveals that most existing MLLMs are constrained by the absence of high-quality, large-scale, real-world datasets, resulting in inadequate performance in automated webpage code generation. To fill this gap, this paper introduces WebCode2M, a new dataset comprising 2.56 million instances, each containing a design image along with the corresponding webpage code and layout details. Sourced from real-world web resources, WebCode2M offers a rich and valuable dataset for webpage code generation across a variety of user scenarios. The dataset quality is ensured by a highly accurate scoring model that filters out instances with aesthetic deficiencies or other incomplete elements. To validate the effectiveness of our proposed dataset, we introduce a baseline model based on the *Vision Transformer* (ViT), named WebCoder, and establish a benchmark for fair comparison. Additionally, we introduce a new metric, TreeBLEU, to measure the structural hierarchy recall. The benchmarking results demonstrate that our dataset significantly improves the ability of MLLMs to generate code from webpage designs, confirming its effectiveness and usability for future applications in front-end design tools. Finally, we highlight several practical challenges introduced by our dataset, calling for further research. We have hosted the WebCode2M on an anonymous webpage: <https://webcode2m-anonymous.github.io>.

## 1 Introduction

Front-end software developers typically create webpages based on *Graphical User Interface* (GUI) mockups designed by UI designers. However, this process is often time-consuming and costly. To this end, several neural models have been proposed to automate the process of generating code from GUI design images, thereby alleviating the burden on front-end developers. Among these, *pix2code* [14] and *sketch2code* [48] are two exemplary works that translate images, whether simple-styled UI designs or hand-drawn sketches, into front-end code. Recently, *Multimodal Large Language Models* (MLLMs), such as GPT-4V [42], have also demonstrated impressive potential in this area.

Despite its potential, we are still far from fully automating front-end engineering to achieve true “screenshot in, code out” functionality. In particular, as highlighted in a recent work [53], the

complexity of code generation increases with the increase in the total number of *HyperText Markup Language* (HTML) tags, the diversity of unique tags, and the depth of the Document Object Model (DOM) tree. Proprietary MLLMs, such as GPT-4V, also exhibit a notable decline in performance when confronted with real-world webpage designs that feature complex structures and a larger variety of unique HTML tags [53].

One possible solution lies in fine-tuning pre-trained LLMs, with the potential for improved performance as the amount of data increases. However, this approach faces a significant limitation because existing datasets are either too small to provide meaningful generalization [21, 53] or consist of synthetic data that does not fully capture the complexity and variability of real-world webpage designs [28, 62]. For instance, Design2Code [53] contains only 484 real-world samples, intended solely for testing and insufficient for effective fine-tuning. WebSight [28] is another dataset comprising approximately 0.8 million synthesized samples generated by LLMs. However, a significant disparity exists between these samples and real-world data [53]. Specifically, WebSight samples average 647 tokens, 19 tags, and a DOM depth of 5, whereas our study reveals that real-world samples can involve up to 50 times more tokens, six times as many tags, and double the DOM depth (See Fig. 2). This substantial gap between synthetic and real-world data can limit the practical effectiveness of fine-tuned MLLMs when applied to more complex, real-world scenarios.

**Our Work.** To fill this gap, this paper introduces a large-scale real-world dataset for webpage generation, named WebCode2M, which includes 2.56 million instances. Each instance features a high-quality webpage design image paired with its corresponding HTML and *Cascading Style Sheets* (CSS) code. This dataset overcomes the limitations of existing datasets by offering a diverse and comprehensive collection of real-world webpage designs and their associated code. On average, the samples contain 31,216 tokens, 158 tags, and a DOM depth of 13. WebCode2M is poised to be an invaluable resource for advancing the development of webpage code generation models.

To construct our dataset, we first collect approximately 0.5 billion real-world webpages from the Common Crawl dataset [2], which includes a diverse array of web domains and styles. For each webpage, we extract the associated CSS code and image elements, remove noise and irrelevant code, and generate screenshots. To ensure data quality, we develop a scoring model to filter out instances with incomplete elements or suboptimal aesthetic quality, such as disorganized layouts or excessive blank spaces, as illustrated in Figure 9 (See Appendix). This scoring model is trained on a manually annotated subset of 10,000 entries, curated by six annotators using consensus-based annotation, achieving a validation accuracy of 90% in distinguishing high- from low-quality instances.

To demonstrate the potential of our dataset for improving automatic webpage generation, we fine-tune a ViT model [17] as a new baseline for translating webpage design images into HTML/CSS

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

Conference'17, July 2017, Washington, DC, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

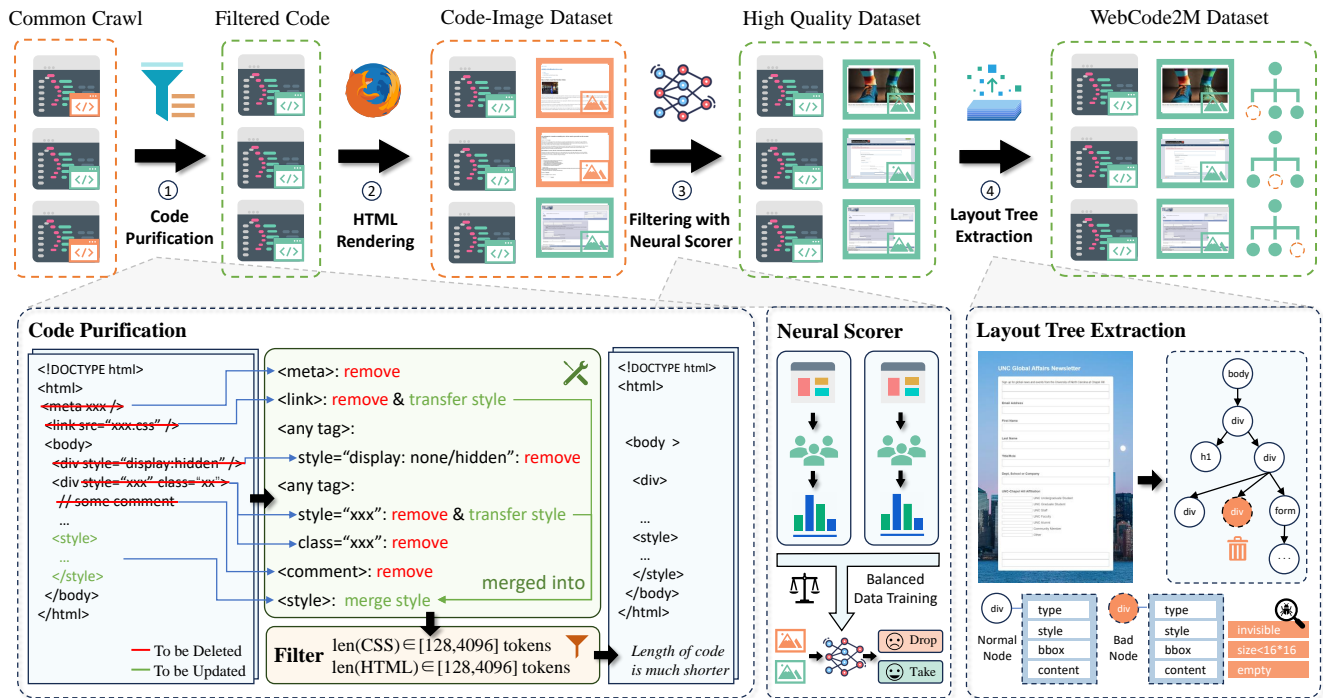


Figure 1: The pipeline of constructing the WebCode2M dataset.

code and establish a benchmark for fair comparison. Compared to the two fine-tuned baselines, Design2Code-18B [53] and WebSight VLM-8B [28], our model, fine-tuned from the smaller Pix2Struct-1.3B, outperforms both across all evaluation metrics, including CLIP-based visual similarity [44], low-level appearance accuracy [53], and our proposed TreeBLEU to measure the structural hierarchy recall. We also benchmark a broad array of general-purpose MLLMs, including the LLaVa family [34], CogAgent-Chat-18B [25], GPT-4V, GPT-4o [41], Gemini [13], and Claude [1]. Experimental results show that WebCoder outperforms these models across most evaluation metrics. The only exception is GPT-4o, which achieves higher similarity in CLIP and visual appearance but has a lower substructure recall rate.

**Contributions.** The primary contributions of our work are summarized as follows:

- **New Dataset.** To the best of our knowledge, WebCode2M is the first real-world and large-scale dataset tailored to empower MLLMs in the domain of generating webpage code from high-fidelity images.
- **Comprehensive Benchmark.** We fine-tune an MLLM, named WebCoder, on our WebCode2M dataset and evaluate it through a comprehensive set of experiments alongside other fine-tuned baselines. Experimental results demonstrate the effectiveness of the dataset in enabling MLLMs to automatically generate code from webpage designs. Additionally, we introduce a novel metric, TreeBLEU, to measure the structural hierarchy recall.
- **Open-Source Resources.** We open-source the code base, the dataset, and the new benchmark model, making them freely available to the research and developer communities, for further

innovation in automating front-end engineering. The resources are available at <https://webcode2m-anonymous.github.io>.

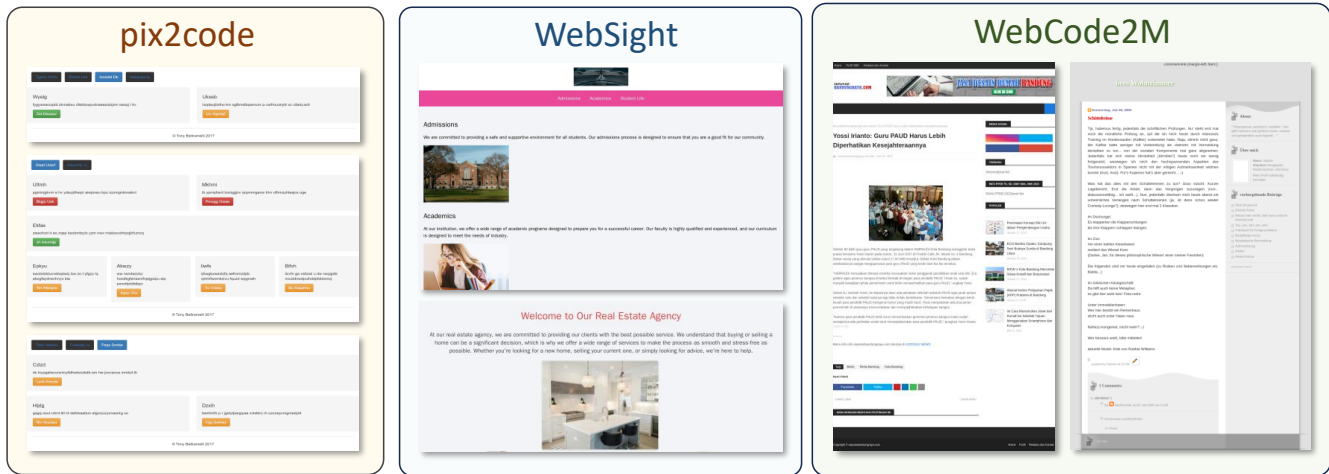
## 2 WebCode2M: The Dataset

This section details the construction process of the WebCode2M dataset, outlines its ethical compliance, describes the dataset partitioning, and highlights its key characteristics.

### 2.1 Dataset Construction

The aim of this study is to curate a dataset that facilitates training neural models to generate code from webpage designs. As large-scale human-designed screenshots are hard to collect manually, we opt to reversely generate screenshot image from a curated open-source web dataset via rendering the webpage code. Figure 1 illustrates the pipeline for constructing WebCode2M, encompassing steps such as code purification, HTML rendering, filtering with a neural scorer, and layout tree extraction.

**Raw Data Collection.** We build our dataset on top of the Common Crawl dataset [2], a comprehensive collection of global webpage data spanning from 2013 to the present, updated monthly through web crawling. Previously, the Common Crawl dataset is primarily used for pre-training models on text-based tasks. Due to our computational resource and download speed constraints, we randomly sample approximately 0.5 billion webpages from the first segment of the CC-MAIN-2023-50 version, which contains about 3.35 billion webpages, as our initial data. We then download the external CSS code for each HTML file and integrate it into the HTML text.



**Figure 2: Representative screenshots of webpages in WebCode2M and other datasets. From left to right are Pix2code, WebSight, and our WebCode2M dataset. Compared to the first two artificially synthesized datasets, ours is derived from real-world online websites, showcasing significantly greater diversity in elements, content, colors, and structural layouts.**

**Code Purification.** Our investigation reveals that webpage code from online websites is often lengthy and includes redundant elements, which significantly impairs the model’s ability to accurately learn the correlation between webpage code and screenshots [28]. To ensure data quality, we meticulously clean the combined HTML and CSS text adhering to the following steps.

- **Quick length filtering.** Furthermore, we filter out samples that are either excessively long or too short. This is because parsing errors or other issues often lead to excessively short HTML or CSS code, while excessively long input contexts significantly slow down our training and inference procedures. Specifically, we employ a rapid filtering method based on code length, measured by the number of characters. Assuming that one word is approximately five characters long, we establish length ranges for HTML and CSS code between  $[128 \times 5, 2048 \times 5]$  characters and  $[128 \times 5, 4096 \times 5]$  characters, respectively. Webpages that fall outside these ranges are filtered out.
- **Redundant code elements cleansing.** The code samples in the raw dataset may include redundant elements, such as comments and hidden elements, as well as components that do not directly affect the rendering of static HTML pages. To address this, we propose removing the following contents from both HTML and CSS code: comments, `<meta>` and `<script>` tags, hidden elements (hidden, zero-sized, or outside the display range), attributes not in (class, id, width, height, style, src) of all HTML elements, and CSS styles that are not effective in the HTML code.

**HTML Rendering for Screenshot Generation.** After cleansing the data, we generate webpage screenshots from the combined HTML and CSS code. This process is implemented using Playwright [4], a headless browser automation tool that allows us to render webpages and capture high-fidelity screenshots. By simulating a real browser environment, Playwright ensures that the rendered webpage accurately reflects the appearance of the HTML and CSS code. This process is highly time-consuming, accounting

for roughly 80% of the total processing time, which spans approximately one month.

**Filtering with a Neural Scorer.** In our empirical data analysis, we observe that a considerable proportion of the generated screenshots exhibit deficiencies in aesthetics, as shown in Figure 9 (See Appendix). These low-quality screenshots are generally attributed to incompletely loaded pages resulted from various factors, for instance, invalid image links, and cases where the content is mainly composed of textual content. The presence of flawed screenshots can compromise the overall quality of the dataset, necessitating a rigorous filtering of the acquired data. Given the large volume of our dataset, manually screening all the data is impractical. Therefore, we train a classification model to serve as a neural scorer, assessing the screenshots and subsequently eliminating samples that fall below a specified score threshold.

In practice, we devise an annotating tool (See Figure 10 in Appendix) and manually annotate a subset of the generated screenshots. The scoring criteria are thoughtfully crafted, and each criterion satisfied will be awarded one point: (1) Normal webpage layout (human-designed layout, not simple auto single-column arrangement); (2) Normal webpage styling (elements like lists and blocks are styled, not using default styles); (3) No excessive blank areas; (4) Rich color combinations; and (5) Good aesthetic appearance. During the manual annotation process, we invite six annotators who hold a Bachelor’s degree in Computer Science and have at least three years of web development experience. We then divide them into two groups to perform consensus annotation, where annotators within each group evaluate the same data. This annotation strategy minimizes the influence of subjective factors on the scoring results. The annotation process takes approximately two weeks for all the participants, ultimately yielding 10,000 manually scored data entries. The detailed annotation procedure is presented in Appendix C.

The score distribution of the manually labeled subset is depicted in Figure 4 (inner circle). The statistics reveal that 80% of the data

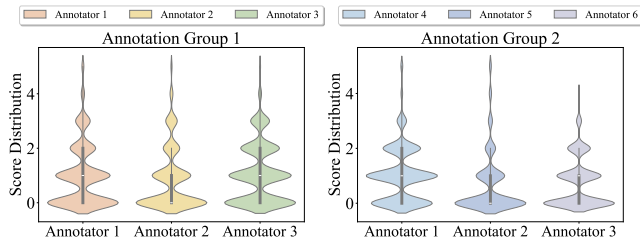


Figure 3: Score distributions of annotators in two groups.

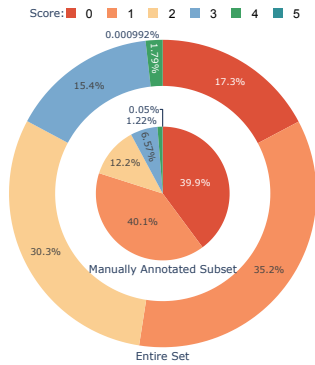


Figure 4: Score distribution of the manually annotated subset (inner ring) and the entire dataset (outer ring) before score-based filtering.

fell within the low-quality category, scoring between 0 and 1. Conversely, only 20% of the entries, exhibiting scores of 2 or higher, demonstrate a commendable level of structural integrity and aesthetic appeal. We also conduct a consistency analysis of all the data from the annotators (See Figure 3). Although certain differences exist among annotators within the same group, the overall trend remains similar. By averaging scores within the same group, the impact of subjectivity is significantly reduced. Utilizing the rated data, we train a *ResNet-50* [23] model to serve as a scorer, predicting the score of input screenshots. This scorer achieves 75% accuracy on the test portion of the manually scored subset and nearly 90% accuracy in binary classification, determining whether the score is greater than or equal to two. Using this scorer, we remove samples with scores less than two, which accounted for 52.5% of the entire raw dataset (as shown in the outer circle of Figure 4).

**Layout Tree Extraction.** Considering that the webpage’s layout defines the spatial arrangement and relationships between UI components, it can serve as a critical source of information. If available, the layout can act as a training target for the model, facilitating code generation by guiding the model to understand not only the structure of the webpage but also the precise positioning of elements. Thus, each data instance in our dataset is upgraded to a triplet: (webpage code, design image, layout). The layout, represented by the bounding boxes (BBox) of HTML elements, includes key information such as the size, location, and hierarchy of page components. This additional layout data will aid the model in learning to generate the webpage DOM tree structure more accurately.

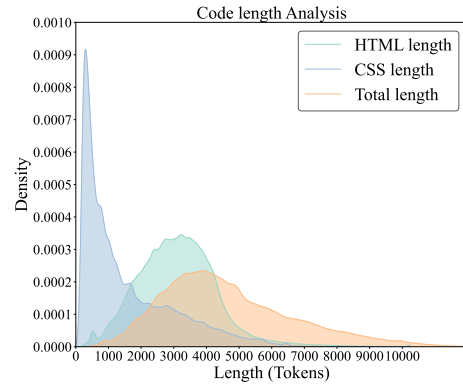


Figure 5: Length density of the WebCode2M dataset.

## 2.2 Ethical Compliance

Since our dataset is sourced from online webpages, it may contain content that is inappropriate for public release, such as explicit material or violent content. To mitigate ethical concerns regarding potential negative impacts, such as the misuse of models trained on this dataset, we perform additional filtering steps. Specifically, we apply an image filter to the screenshots and a profanity filter to the web text. Only samples that passed both filters are retained. Detailed filtering procedures are provided in Appendix F.

Table 1: Dataset Partition.

Subset	Purpose	Size	Length (Tokens)
WebCode2M	Training	2,563,905	[368,16668]
WebCode2M-Short	Testing	256	[551, 2045]
WebCode2M-Mid	Testing	256	[2052, 4085]
WebCode2M-Long	Testing	256	[4098, 10990]

## 2.3 Dataset De-Duplication and Partition

To support the use of our dataset in both fine-tuning models for webpage code generation and evaluating their performance, we organize the data into well-structured partitions. After using the hash codes of the screenshots to quickly de-duplicate the refined dataset, which comprises millions of entries, we sample approximately two thousand entries with a score above 4 as our candidate test dataset. The remaining 2.56 million entries serve as the training dataset. For the candidate test dataset, we further remove duplicates using CLIP [44] similarity and conduct a manual inspection on each entry.

Furthermore, we partition the test subsets based on code length to assess the model’s code generation capability across varying levels of difficulty. As illustrated in Figure 5, the dataset shows a wide range of data length variations. Specifically, we use two length thresholds (i.e., 2048 and 4096), to select 256 entries from within three length ranges, thus creating three test subsets. We refer to them as WebCode2M-Short, WebCode2M-Mid, and WebCode2M-Long. Table 1 summarizes the overall statistics and provides detailed length statistics of WebCode2M for both training and testing.

## 2.4 Dataset Characteristics

Upon acquiring the final dataset WebCode2M, we conduct an analysis to identify several key characteristics. To quantitatively assess

**Table 2: A statistical comparison between our dataset and all the publicly available datasets. The statistical data of WebSight and Design2Code is referred to [53].**

Dataset	Pix2code	WebSight	Design2Code	WebCode2M	WebCode2M_Short	WebCode2M_Mid	WebCode2M_Long
Purpose	Training&Testing	Training	Testing	Training	Testing	Testing	Testing
Source	Synthetic	Synthetic	Real-World	Real-World	Real-World	Real-World	Real-World
Size (#samples)	1742	0.8M	484	2.5M	256	256	256
Avg. Len (#tokens)	1316±177	647±216	31216±23902	5366±2393	2025±514	3750±765	7940±1853
Avg. Tags	52±8	19±8	158±100	184±77	81±34	144±61	222±81
Avg. DOM Depth	8±0	5±1	13±5	15±5	10±4	13±7	16±4
Avg. Unique Tags	17±0	10±3	22±6	24±6	18±4	21±5	26±5

the diversity and quality of our dataset, we employ the same statistical metrics used in Design2Code, facilitating a comparison with other datasets. The results are presented in Table 2. Specifically, Avg. Len represents the token length as determined by the GPT-2 tokenizer [3]; Avg. Tags indicates the total number of tags in the HTML code; Avg. Unique Tags denotes the count of distinct tags in the HTML code; and Avg. DOM Depth signifies the maximum depth of the HTML’s DOM Tree.

**Superior Diversity.** From Table 2, it is apparent to see that our dataset contains a significantly greater number and diversity of HTML tags and exhibits a more intricate DOM tree structure compared to the pix2code and WebSight datasets. This suggests that our dataset, sourced from real-world webpages, offers a remarkable diversity advantage over synthetic datasets generated by LLMs like WebSight. Design2Code, which also utilizes real-world data through the C4 dataset [45] from the Common Crawl corpus, exhibits a comparable distribution across these metrics, underscoring the benefits of real-world data in capturing the complexity of actual webpages. Moreover, this comparison highlights significant deviations in data attribute distributions between LLM-generated datasets and real webpages.

Figure 2 presents several representative screenshots from the datasets (excluding Design2Code). The pix2code dataset comprises basic block elements and text-based UI elements, suitable for both Android and iOS UIs. In contrast, WebSight consists of structurally simple webpages. Our dataset, on the other hand, closely mirrors typical real webpages, featuring a variety of layout structures and rich elements such as images. Additionally, our dataset captures webpages in a diverse range of languages (See Figure 12 in the Appendix).

**Large Scale and High Quality.** Compared to pix2code and Design2Code, which contain only a few thousand or fewer data samples, and WebSight, which includes 0.8 million samples, our dataset is significantly larger, comprising 2.56 million samples. This dataset includes both a comprehensive training dataset and a high-quality test dataset, making it much larger in scale. Notably, compared to Design2Code, our WebCode2M significantly reduces the average code length to about one-tenth of its original size, while maintaining the diversity and quantity of HTML tags, thereby preserving the high quality of the dataset.

### 3 Benchmarking

We introduce a baseline model based on the ViT, named WebCoder, and establish a benchmark for fair comparison.

#### 3.1 WebCoder: A Reference Baseline

To demonstrate the potential of our dataset in enhancing automatic webpage code generation, we fine-tune a ViT model to establish a new baseline for translating design images into HTML/CSS code. Specifically, we select Google’s Pix2Struct-1.3B [29] as our base model. This model, based on the ViT architecture, has been pre-trained on webpage code derived from URLs in the C4 dataset [45]. Pix2Struct-1.3B is notable for its robustness to extreme aspect ratios and ability to adapt dynamically to changes in sequence length and resolution. Furthermore, it rescales images by distorting the aspect ratio to preserve original image information, facilitating the processing of variable resolutions. We conduct a full fine-tuning of the pre-trained Pix2Struct on our training dataset, resulting in our model, WebCoder.

#### 3.2 Setup and Baselines

Our evaluation experiments focus on two primary Research Questions (RQs):

**RQ1: The effectiveness of the training dataset.** To investigate the ability of our training dataset to empower MLLMs in webpage generation, we compare WebCoder with several state-of-art models which are also fine-tuned specifically for the webpage generation task:

- **WebSight VLM-8B** [28]. Huggingface’s WebSight utilized its training dataset and the DoRA [35] mechanism to fine-tune a base VLM, which has been pre-trained on image/text pairs.
- **Design2Code-18B** [53]. Stanford’s Design2Code is also fine-tuned on the WebSight dataset. It adopts CogAgent [25] as its base model and utilizes LoRA [26] as the finetuning method to accelerate the training process.
- **WebCoder\***. Another Pix2Struct model in the same setting but trained on the WebSight dataset for comparative experiments.

**RQ2: Benchmarking on the test datasets.** We also introduce a broad array of the latest and most powerful general-purpose pre-trained MLLMs for benchmarking:

- **LLaVA Family** [34]. The LLaVA family consists of various MLLMs that connect a vision encoder and an LLM for general-purpose visual and language understanding. In our work, we introduce **LLaVA-v1.5-7B**, **LLaVA-onevision-0.5B**, and **LLaVA-onevision-7B** as the baselines. The prompt used for these models follows [8] and is detailed in Appendix A.
- **CogAgent-Chat-18B** [25]. CogAgent-Chat-18B is a general MLLM that supports both low- and high-resolution images and performs quite well on webpage navigation. We also input the

screenshot and a simple prompt *Write an HTML code* to generate the webpage as Design2Code does.

- **Commercial Models.** Some general commercial models have demonstrated impressive performance across various fields, proficient in both code generation and web understanding. Therefore, we introduce OpenAI’s **GPT-4V** and **GPT-4o** [41], Google DeepMind’s **Gemini** [13], and Anthropic’s **Claude** [1] as baselines. The prompt for these models also follows [8], detailed in Appendix A.

Although previous work [53] suggests that multi-round generation methods (e.g., self-correction) may outperform one-pass generation, baseline models such as Design2Code-18B and WebSight VLM-8B are fine-tuned with one-pass dataset and only support one-pass generation. Therefore, to ensure a fair comparison, all baselines will employ the one-pass generation strategy.

### 3.3 Evaluation Metrics

This section presents the evaluation metrics used in our work, including measurements for visual similarity and structural similarity, as well as several classic metrics.

**Visual Similarity Measurement.** We adopt CLIP [44] similarity and *Visual Score* [53] as two major metrics to assess visual similarities between the generated webpage page and ground truth. CLIP similarity is derived from calculating the cosine value of two images’ latent vectors encoded by CLIP, which measures the overall visual similarity of two images. *Visual Score* is utilized to measure the matching degree of low-level elements in terms of appearance, calculating the average scores of the matching ratio between the reference and candidate blocks, as well as the similarity at four block levels in terms of color, text, CLIP, and position.

**Structure Similarity Measurement.** Skeletons of webpage code that determines the layout and appearance of the page, also known as the HTML DOM Tree, can also serve as a metric to evaluate structural similarity that compares ground truth (for instance, during the training phase or when the target code is provided in the inference stage) and the DOM Tree of the generated code. Inspired by [47], we propose a new metric **TreeBLEU** to evaluate the matching degree of the generated HTMLs’ DOM tree (without terminal nodes that contain tags’ attributes, e.g., content and style) compared to the ground truth.

TreeBLEU is defined as the proportion of all 1-height subtrees (See Algorithm 1) in a given tree that can be matched with that of a reference tree. Let  $S(\cdot)$  be the set of 1-height subtrees, then it can be formulated as:

$$\text{TreeBLEU} = \frac{|S(t) \cap S(t_r)|}{|S(t_r)|},$$

where  $t$  and  $t_r$  denote the given tree and the reference tree, respectively. Different from htmlBLEU [54], a hybrid metric composed of four scores (the detailed definition of which is not available), our TreeBLEU focuses on the similarity of HTML DOM Tree in an integrated manner, as detailed in Appendix B.

**Classic Metrics.** We also assess the experimental results with several traditional metrics. Although these indicators primarily originate from Natural Language Processing (NLP) and some Computer Vision (CV) tasks, making them potentially less applicable to

---

#### Algorithm 1 Get All 1-height Subtrees of DOM tree.

---

**Require:** A multiway tree node  $root(chlds, name)$

**Ensure:** A set  $S$  of all 1-height subtrees

```

1: Initialize an empty set  $S$ 
2: function TRAVERSE( $node, S$ )
3:   if number of children of  $node \neq 0$  then
4:     Initialize an empty string  $subtree$ 
5:     Append  $node.name$  to  $subtree$ 
6:     for each child  $c$  in  $node.chlds$  do
7:       Append  $c.name$  to  $subtree$ 
8:     end for
9:     Add  $subtree$  to  $S$ 
10:  end if
11:  for each child  $c$  in  $node.chlds$  do
12:    TRAVERSE( $c, S$ )
13:  end for
14: end function
15: TRAVERSE( $root, S$ )

```

---

our context, they still provide valuable insights into specific performance aspects. The conventional metrics we utilize include widely accepted evaluation indicators such as BLEU, ROUGE-I, MSE, and SSIM [59]. Detailed results can be found in Table 8 (See Appendix).

### 3.4 Implementation Details

We configure the model to process a maximum of 1,024 patches, representing the upper limit of image segments it can handle and set the maximum sequence length to 2,048 tokens. These settings are selected to balance training and inference speed with task accuracy. Due to GPU memory limitations, we set the batch size as 1 during training. In the initial phase of training, we fine-tune the model on a subset of our WebCode2M dataset, with a sequence length capped at 2,048 tokens. This phase consisted of three training epochs, totaling 90,000 iterations, with a maximum learning rate of  $5e-5$  and a cosine learning rate scheduler. The primary objective was to equip the model with the ability to generate code from visual inputs. Subsequently, we refine our approach by decreasing the maximum learning rate to  $1e-5$  and performing an additional three epochs of fine-tuning on a subset of the dataset, featuring a reduced sequence length of 1,024 tokens and consisting of 10,000 iterations. All the experiments are run on a Linux server equipped with 4 NVIDIA A100 80G GPUs.

### 3.5 Effectiveness of the Training Dataset (RQ1)

Table 3 presents the performance of WebCoder both on the WebSight and WebCode2M datasets, compared to other benchmark models on the WebSight dataset. From this figure, we can observe that our method consistently outperforms all specialized baselines across all three metrics on the real-world test dataset, noting that these specialized models were fine-tuned on the WebSight dataset. Comparative experiments also demonstrate that the base model, Pix2Struct, achieves a significant performance boost when fine-tuned on our training dataset compared to WebSight. For TreeBLEU—a metric measuring the recall of 1-height subtrees in the target DOM tree—our approach surpasses both specialized and

**Table 3: The performance comparison among the specialized models (the best is marked in bold).**

Model	Training Dataset	WebCode2M-Short			WebCode2M-Mid			WebCode2M-Long		
		Visual	CLIP	TreeBLEU	Visual	CLIP	TreeBLEU	Visual	CLIP	TreeBLEU
WebSight VLM-7B	WebSight	.57 $\pm$ .24	.69 $\pm$ .12	.03 $\pm$ .04	.52 $\pm$ .23	.67 $\pm$ .11	.03 $\pm$ .04	.48 $\pm$ .27	.64 $\pm$ .11	.03 $\pm$ .03
Design2Code-18B	WebSight	.75 $\pm$ .14	.68 $\pm$ .10	.04 $\pm$ .05	.69 $\pm$ .23	.70 $\pm$ .10	.05 $\pm$ .05	.61 $\pm$ .28	.68 $\pm$ .10	.06 $\pm$ .03
WebCoder *-1.3B	WebSight	.42 $\pm$ .32	.68 $\pm$ .11	.06 $\pm$ .06	.36 $\pm$ .30	.67 $\pm$ .11	.04 $\pm$ .04	.38 $\pm$ .29	.65 $\pm$ .11	.04 $\pm$ .04
WebCoder-1.3B	WebCode2M	<b>.78<math>\pm</math>.25</b>	<b>.73<math>\pm</math>.13</b>	<b>.35<math>\pm</math>.17</b>	<b>.69<math>\pm</math>.19</b>	<b>.71<math>\pm</math>.10</b>	<b>.22<math>\pm</math>.11</b>	<b>.65<math>\pm</math>.21</b>	<b>.69<math>\pm</math>.12</b>	<b>.15<math>\pm</math>.07</b>

**Table 4: Benchmarking performance of several general-purpose MLLMs using the WebCode2M. (the best is marked in bold).**

Model	WebCode2M-Short			WebCode2M-Mid			WebCode2M-Long		
	Visual	CLIP	TreeBLEU	Visual	CLIP	TreeBLEU	Visual	CLIP	TreeBLEU
LLaVA-v1.5-7B	.43 $\pm$ .27	.60 $\pm$ .33	.07 $\pm$ .05	.21 $\pm$ .28	.29 $\pm$ .38	.05 $\pm$ .04	.19 $\pm$ .27	.28 $\pm$ .37	.04 $\pm$ .03
LLaVA-onevision-0.5B	.24 $\pm$ .31	.62 $\pm$ .11	.06 $\pm$ .03	.28 $\pm$ .31	.61 $\pm$ .10	.05 $\pm$ .03	.22 $\pm$ .29	.59 $\pm$ .11	.03 $\pm$ .02
LLaVA-onevision-7B	.34 $\pm$ .32	.63 $\pm$ .10	.08 $\pm$ .07	.30 $\pm$ .30	.64 $\pm$ .09	.06 $\pm$ .06	.30 $\pm$ .30	.61 $\pm$ .10	.04 $\pm$ .04
CogAgent-Chat-18B	.46 $\pm$ .31	.68 $\pm$ .11	.01 $\pm$ .03	.40 $\pm$ .31	.66 $\pm$ .10	.01 $\pm$ .02	.39 $\pm$ .30	.65 $\pm$ .10	.01 $\pm$ .01
Gemini	.35 $\pm$ .41	.75 $\pm$ .10	<b>.16<math>\pm</math>.10</b>	.38 $\pm$ .40	.74 $\pm$ .11	<b>.15<math>\pm</math>.08</b>	.34 $\pm$ .41	.73 $\pm$ .10	<b>.14<math>\pm</math>.06</b>
Claude	.52 $\pm$ .43	.77 $\pm$ .10	.13 $\pm$ .08	.35 $\pm$ .42	.76 $\pm$ .09	.14 $\pm$ .08	.37 $\pm$ .43	.74 $\pm$ .09	.13 $\pm$ .06
GPT-4V	.68 $\pm$ .32	.74 $\pm$ .10	.12 $\pm$ .07	.65 $\pm$ .33	.71 $\pm$ .10	.11 $\pm$ .06	.62 $\pm$ .35	.67 $\pm$ .10	.10 $\pm$ .05
GPT-4o	<b>.85<math>\pm</math>.16</b>	<b>.77<math>\pm</math>.10</b>	.15 $\pm$ .09	<b>.81<math>\pm</math>.20</b>	<b>.77<math>\pm</math>.09</b>	.13 $\pm$ .08	<b>.82<math>\pm</math>.18</b>	<b>.74<math>\pm</math>.09</b>	.11 $\pm$ .05

general-purpose models, indicating that our model better reflects real-world node types and substructures. Additionally, on the two visual similarity metrics—visual score and CLIP similarity—our model exceeds most general-purpose models and either matches or outperforms GPT-4V. Collectively, these results demonstrate that our dataset offers greater practical potential than synthetically generated datasets and suggest that our proposed training dataset can effectively unleash the potential of MLLMs in webpage generation.

### 3.6 Benchmarking on the Test Datasets (RQ2)

Table 4 benchmarks the performance of several general-purpose MLLMs using the WebCode2M test dataset. From this figure, we can observe several interesting findings: **(1) Generating lengthy code is challenging.** Almost all metrics for nearly all models drop significantly as the target code length increases. For example, as the dataset transitions from WebCode2M-short to WebCode2M-mid and finally to WebCode2M-long, the highest TreeBLEU score for specialized models drops from 0.35 to 0.15, the highest CLIP similarity decreases from 0.73 to 0.69, and the highest Visual Score declines from 0.78 to 0.65. **(2) Model size matters.** In LLaVA family, several models show a significant improvement across all metrics as model parameters increase, with LLaVA-v1.5-7B and LLaVA-onevision-7B achieving the best performance, while LLaVA-onevision-0.5B performs poorly across all metrics, indicating that MLLMs require more parameters to achieve better results in webpage generation tasks. **(3) Most general-purpose MLLMs struggle with webpage code generation.** Among these models, only GPT-4V matches the performance of our model trained on WebCode2M, while GPT-4o significantly outperforms all other models. All remaining general-purpose models generally underperform compared to specialized models, with consistently low scores across all metrics.

Notably, GPT-4o significantly outperforms all specialized and other general-purpose MLLMs across all metrics. Moreover, its performance remains highly stable as the complexity increases, without showing significant degradation. For instance, as the dataset

transitions from WebCode2M-short to WebCode2M-mid and finally to WebCode2M-long, its visual score, from 0.85 to 0.81 and then to 0.82. However, our goal is not to propose a dataset that allows small specialized models to surpass super MLLMs with hundreds of billions of parameters, as that would be unrealistic. Instead, **our aim is to assist MLLMs in the webpage generation task and enable smaller models to achieve competitive performance.**

## 4 Related Work

Generating code from webpage designs is essentially an image-to-code task, which primarily consists of two key components: image representation and code generation. We review the related works from the perspectives of image representation learning, code generation, and image-to-code.

**Image Representation Learning.** To obtain more suitable representations of images, early works proposed using Variational Autoencoders (VAEs) to generate latent vectors for images [27, 56]. Other researchers have explored employing contrastive learning to derive image encoders from large-scale training datasets [16]. The ViT was introduced to break down an image into sequences of fixed-size patches, applying a transformer to process them and accommodating variable resolutions [9]. In recent years, Diffusion Models (DM) [24] have achieved significant success in image representation learning, understanding, and generation tasks. To alleviate the computational burden of operating in pixel space, researchers proposed training DMs within the latent space of advanced pre-trained autoencoders [49]. Recently, SDXL[43] was developed, leveraging a three-times larger UNet [50] backbone and introducing a refinement model.

**Code Generation.** The development of code models has advanced significantly with the increasing availability of computational resources, evolving from small models with only a few million parameters to medium-sized models with billions of parameters, and ultimately to ultra-large models exceeding hundreds of billions. Early

works typically employed simple network architectures trained on small datasets for code generation tasks. For instance, some work [12, 20, 22] use RNNs [60] to treat code as token sequences, while others explored tree-structured neural networks [33, 37, 39] or graph neural networks [10, 11, 15] to capture the structural information of code. With the rise of transformer models, researchers began to explore supervised and unsupervised methods for training transformer-based models on large-scale codebases, such as GitHub. Notable works in this area include CodeBERT [18], CodeT5 [57], StarCoder [30], and AlphaCode [32], with AlphaCode even achieving an average ranking in the top 54.3% in simulated programming competitions on the Codeforces platform. More recently, the landscape of code generation has been significantly influenced by large language models (LLMs) such as CodeGen [38], CodeT5+ [58], InCoder [19], GPT-3.5 [40], StarCoder [31], Code Llama [51], and WizardCoder [36].

**Image to Code.** Several early works have made pioneering contributions by focusing on generating code from simple images. For example, to reverse engineer program code from Graphical User Interfaces, [14] introduced *pix2code*, which is trained on a synthetic dataset of GUI screenshots and corresponding source code, including iOS, Android, or web-based GUI, to generate Domain-Specific Language code. *Sketch2code* [48] generates website code from wireframe sketches exploring two approaches: a computer vision-based method that detects elements and structures, and a deep learning-based method. With advances in computational power, some studies have explored using larger models to advance the image-to-code task. [61] addressed the challenge of *screen parsing* by predicting UI hierarchy graphs from screenshots, by using Faster-RCNN [46] to encode the screenshot images and employing an LSTM-based attention mechanism to construct graph nodes and edges. *Pix2Struct* [29], pre-trained by learning to generate simplified HTML from masked website screenshots, demonstrated substantial improvements in visual language understanding across nine tasks in four different domains. To address the challenges of rendering inefficiencies and non-differentiability in website generation, [55] employed reinforcement learning to fine-tune a vision-code transformer to minimize the visual differences between the original and generated HTML. While recent efforts have aimed at improving code generation from high-definition images, the results are still far from being ready for practical application.

Nowadays, several powerful commercial models have emerged, such as OpenAI's GPT-4V and GPT-4o [41], Google DeepMind's Gemini [13], and Anthropic's Claude [1]. These models have shown impressive performance across various tasks, including image understanding and code generation with the advantage of allowing continuous adjustment and optimization via chat. However, their performance in generating webpages from high-resolution images remains suboptimal.

## 5 Discussion

In this section, we discuss several practical challenges associated with our dataset when applied to webpage code generation and highlight several limitations that warrant further study.

## 5.1 Practical Challenges to Study

In the course of our research, we identify three practical challenges that need to be addressed to achieve the ideal generation of webpage code from design images. These challenges are presented here to guide future research: **(1) Lengthy code generation.** As shown in Table 2, despite our efforts to clean up noise in the webpage code, such as invisible elements, the HTML text remains lengthy, reflecting its complexity to some extent. This presents significant challenges to both the training effectiveness and efficiency of MLLMs. **(2) Capturing structural information of UI visions.** Given the potential overlap of sub-elements in images and the lack of distinct borders for some elements, extracting structured or hierarchical information from images presents a significant challenge. Our empirical study of GPT-4V reveals that, while it excels in capturing text and color from images when generating webpage code, it struggles with capturing the hierarchy of UI elements. Therefore, designing a model that is more proficient in generating the hierarchical structure for translating design diagrams into webpage code is a promising avenue. **(3) Generation of image elements.** All existing webpage code generation models fail to accurately reproduce image elements in the design visions, severely hindering their practical application. There is an urgent need for a framework capable of generating or extracting image elements from the original design and assembling them into the final webpage code.

## 5.2 Limitations and Future Directions

Firstly, although we employ a meticulously designed neural scorer to enhance data quality, this scoring method inherently contains a degree of subjectivity and achieves an accuracy of approximately 90%. Consequently, some low-quality data remain in the final dataset. However, we consider this acceptable given the trade-off between efficiency and quality, as manual screening of millions of data is impractical. Secondly, the results presented in Table 3 and Table 4 indicate that all models exhibit significant variance on certain metrics. This suggests that the model's generation capability is insufficiently stable and performs poorly on some test data, underscoring the need for a more robust framework to accomplish this task. Finally, because our dataset is sourced from crawled online data, it inevitably contains a minimal amount of inappropriate content, such as violent material, despite our extensive filtering efforts.

## 6 Conclusion

In this paper, we have proposed WebCode2M, the first real-world and large-scale dataset with layout information for generating webpage code from designs. This dataset consists of over 2.56 million samples for both training and testing. We have presented the detailed pipeline of dataset construction and conducted analysis on the curated dataset. The analysis results demonstrate the diversity of our dataset. We fine-tune an MLLM, named WebCoder, on our training dataset. Along with two visual measures and a structure metric, we evaluate our WebCoder with other baselines on the proposed dataset. The experiment results demonstrate that our dataset can better empower MLLMs to generate code from webpage designs. We believe that the dataset and benchmark proposed in this work can further advance research in this field.



## References

- [1] The claude 3 model family: Opus, sonnet, haiku. URL <https://api.semanticscholar.org/CorpusID:268232499>.
- [2] The common crawl dataset. <https://data.commoncrawl.org/>.
- [3] Gpt2 model on the huggingface. <https://huggingface.co/openai-community/gpt2>.
- [4] Playwright. <https://playwright.dev/>.
- [5] Sensitive stop words. <https://github.com/fwzwdn/sensitive-stop-words/>.
- [6] The bs4 python package. <https://pypi.org/project/beautifulsoup4/>.
- [7] Fine-tuned vision transformer (vit) for nsfw image classification. [https://huggingface.co/Falconsai/nsfw\\_image\\_detection/](https://huggingface.co/Falconsai/nsfw_image_detection/).
- [8] The screenshot project on the github crawl dataset. <https://github.com/abi/screenshot-to-code/>.
- [9] Dosovitskiy Alexey. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv: 2010.11929*, 2020.
- [10] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL <https://openreview.net/forum?id=BJOFETxR->.
- [11] Miltiadis Allamanis, Earl T. Barr, Soline Ducouso, and Zheng Gao. Typilus: neural type hints. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 91–105. ACM, 2020. doi: 10.1145/3385412.3385997. URL <https://doi.org/10.1145/3385412.3385997>.
- [12] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL <https://openreview.net/forum?id=H1gKY09tX>.
- [13] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakob Sygnowski, and et al. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023. doi: 10.48550/ARXIV.2312.11805. URL <https://doi.org/10.48550/arXiv.2312.11805>.
- [14] Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2018, Paris, France, June 19-22, 2018*, pages 3:1–3:6. ACM, 2018. doi: 10.1145/3220134.3220135. URL <https://doi.org/10.1145/3220134.3220135>.
- [15] Marc Brockschmidt, Miltiadis Allamanis, Alexander L. Gaunt, and Aleksandr Polozov. Generative code modeling with graphs. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL <https://openreview.net/forum?id=Bke4KsA5FX>.
- [16] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.
- [17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiuhua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020. URL <https://api.semanticscholar.org/CorpusID:225039882>.
- [18] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiao Cheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. *ArXiv*, abs/2002.08155, 2020. URL <https://api.semanticscholar.org/CorpusID:211171605>.
- [19] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. InCoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*, 2022.
- [20] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 933–944. ACM, 2018. doi: 10.1145/3180155.3180167. URL <https://doi.org/10.1145/3180155.3180167>.
- [21] Hongcheng Guo, Wei Zhang, Junhao Chen, Yaonan Gu, Jian Yang, Junjia Du, Binyuan Hui, Tianyu Liu, Jianxin Ma, Chang Zhou, and Zhoujun Li. Iw-bench: Evaluating large multimodal models for converting image-to-web. *CoRR*, abs/2409.18980, 2024.
- [22] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish K. Shevade. Deepfix: Fixing common C language errors by deep learning. In Satinder Singh and Shaull Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1345–1351. AAAI Press, 2017. doi: 10.1609/AAAI.V31I1.10742. URL <https://doi.org/10.1609/aaai.v31i1.10742>.
- [23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL <https://doi.org/10.1109/CVPR.2016.90>.
- [24] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Hugo Larochelle, Marc Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/4c5bfc8584f0d967f1ab10179ca4b-Abstract.html>.
- [25] Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for GUI agents. *CoRR*, abs/2312.08914, 2023. doi: 10.48550/ARXIV.2312.08914. URL <https://doi.org/10.48550/arXiv.2312.08914>.
- [26] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL <https://openreview.net/forum?id=nZeVKeeFFy9>.
- [27] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL <http://arxiv.org/abs/1312.6114>.
- [28] Hugo Laurenceon, L'eo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset. 2024. URL <https://api.semanticscholar.org/CorpusID:268385510>.
- [29] Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiong Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 18893–18912. PMLR, 2023. URL <https://proceedings.mlr.press/v202/lee23g.html>.
- [30] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel ZeBaze, Ming-Ho Yee, Loshaj Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nourhan Fahmy, Urvashi Bhattacharyya, W. Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kuznetsov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, JanaBERT, Tri Dao, Mayank Mishra, Alexander Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean M. Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder: may the source be with you! *ArXiv*, abs/2305.06161, 2023. URL <https://api.semanticscholar.org/CorpusID:258588247>.
- [31] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.
- [32] Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom, Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de, Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel Jaymin Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378:1092–1097, 2022. URL <https://api.semanticscholar.org/CorpusID:246527904>.
- [33] Fang Liu, Lu Zhang, and Zhi Jin. Modeling programs hierarchically with stack-augmented LSTM. *J. Syst. Softw.*, 164:110547, 2020. doi: 10.1016/J.JSS.2020.110547. URL <https://doi.org/10.1016/j.jss.2020.110547>.
- [34] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023.
- [35] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *CoRR*, abs/2402.09353, 2024. doi: 10.48550/ARXIV.2402.09353. URL <https://doi.org/10.48550/arXiv.2402.09353>.

- [36] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.
- [37] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 1287–1293. AAAI Press, 2016. doi: 10.1609/AAAI.V30I1.10139. URL <https://doi.org/10.1609/aaai.v30i1.10139>.
- [38] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.
- [39] Changan Niu, Chuanyi Li, Vincent Ng, Jidong Ge, Liguo Huang, and Bin Luo. Spt-code: Sequence-to-sequence pre-training for learning source code representations. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 1–13. ACM, 2022. doi: 10.1145/3510003.3510096. URL <https://doi.org/10.1145/3510003.3510096>.
- [40] OpenAI. ChatGPT. <https://openai.com/blog/chatgpt/>, 2022.
- [41] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL <https://doi.org/10.48550/arXiv.2303.08774>.
- [42] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155, 2022. URL <https://api.semanticscholar.org/CorpusID:246426909>.
- [43] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: improving latent diffusion models for high-resolution image synthesis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=di52zR8xgf>.
- [44] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021. URL <http://proceedings.mlr.press/v139/radford21a.html>.
- [45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL <http://jmlr.org/papers/v21/20-074.html>.
- [46] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015. URL <https://api.semanticscholar.org/CorpusID:10328909>.
- [47] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, M. Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *ArXiv*, abs/2009.10297, 2020. URL <https://api.semanticscholar.org/CorpusID:221836101>.
- [48] Alex Robinson. Sketch2code: Generating a website from a paper mockup. *CoRR*, abs/1905.13750, 2019. URL <http://arxiv.org/abs/1905.13750>.
- [49] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10674–10685. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01042. URL <https://doi.org/10.1109/CVPR52688.2022.01042>.
- [50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells III, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015. doi: 10.1007/978-3-319-24574-4\_28. URL [https://doi.org/10.1007/978-3-319-24574-4\\_28](https://doi.org/10.1007/978-3-319-24574-4_28).
- [51] Baptiste Szolere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.
- [52] Inc. Shutterstock. List of dirty naughty obscene and otherwise bad words. <https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/>.
- [53] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating front-end engineering? 2024. URL <https://api.semanticscholar.org/CorpusID:268248801>.
- [54] Davit Soselia, Khalid Saifullah, and Tianyi Zhou. Learning ui-to-code reverse generator using visual critic without rendering.
- [55] Davit Soselia, Khalid Saifullah, and Tianyi Zhou. Learning ui-to-code reverse generator using visual critic without rendering. 2023. URL <https://api.semanticscholar.org/CorpusID:265302631>.
- [56] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6306–6315, 2017. URL <https://proceedings.neurips.cc/paper/2017/hash/7a98af17e63a0ac09ce2e96d03992fbc-Abstract.html>.
- [57] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *ArXiv*, abs/2109.00859, 2021. URL <https://api.semanticscholar.org/CorpusID:237386541>.
- [58] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.
- [59] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.
- [60] Ronald J. Williams and David Zipser. *Gradient-based learning algorithms for recurrent networks and their computational complexity*, page 433–486. L. Erlbaum Associates Inc., USA, 1995. ISBN 0805812598.
- [61] Jason Wu, Xiaoyi Zhang, Jeffrey Nichols, and Jeffrey P. Bigham. Screen parsing: Towards reverse engineering of ui models from screenshots. *The 34th Annual ACM Symposium on User Interface Software and Technology*, 2021. URL <https://api.semanticscholar.org/CorpusID:237571719>.
- [62] Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *CoRR*, abs/2406.20098, 2024. doi: 10.48550/ARXIV.2406.20098. URL <https://doi.org/10.48550/arXiv.2406.20098>.

## A An Empirical Study

We conducted an empirical study by employing the state-of-the-art MLLM, GPT-4V, to generate webpage code from images in a one-pass manner. We refer to the prompt of a famous open-source project *screenshot-to-code* [8] on the GitHub, making only minor adjustments as shown in Figure 6.

We also investigated the capabilities of the *pix2code* and *Pix2Struct* models. The *pix2code* dataset was first divided into training and testing datasets with an 8:2 split. We then trained the *pix2code* model from scratch on the training dataset while simultaneously fine-tuning the *Pix2Struct* model. To quantitatively assess the performance of *pix2code*, *Pix2Struct*, and GPT-4V on the *pix2code* test dataset, we utilized two automated metrics.

Table 5 shows the comparison results of the performance of several baselines on the *pix2code* test dataset. From this table, it can be seen that ChatGPT-4V’s performance in one-pass generation mode lags behind that of the *pix2code* model, and the model fine-tuned from *Struct2Code*, even when applied to the simplest *pix2code* dataset. This performance discrepancy is particularly evident in the generation of the HTML DOM tree structure, which is represented as TreeBLEU in Table 5.

To more vividly illustrate the shortcoming of GPT-4V in capturing the structural information in original screenshots, we choose a representative example (similar situations are often found in actual test results), as shown in Figure 7. It can be clearly observed from the figure that the webpage generated by GPT-4V does not reflect the hierarchical structure of the referenced image, but rather appears as a simple auto-sorted list. This situation is also frequently encountered during our experience with the *screenshot-to-code*. The primary explanation for this disparity may lay in the fact that both models have been fine-tuned on specific task datasets, while ChatGPT-4V has not.

**Table 5: The performance comparison on the *pix2code* test dataset. The *pix2code* model and *Pix2Struct* model are both fine-tuned on the *pix2code* training dataset. The GPT-4V is prompted to generate webpage code in one-pass mode.**

Model	TreeBLEU
<i>pix2code</i> -Beam	0.98
<i>pix2code</i> -Gready	0.99
<i>Pix2Struct</i> -282M	0.79
<i>Pix2Struct</i> -1.3B	0.92
GPT-4V	0.09

## B The Implementation of TreeBLEU

The core implementation of TreeBLEU includes two parts: the generation of a minimalist HTML DOM Tree and the Subtree Collection Algorithm.

- Minimalist HTML DOM Tree. The original HTML DOM Tree can be easily obtained by some tools such as the *bs4* [6] package in Python. After acquiring the HTML DOM Tree, the next step is to remove the terminal nodes, the text nodes and image nodes, etc. The resulting DOM Tree consists only of the type names of HTML tags.
- Subtree Collection Algorithm, as shown in Algorithm 1.

Given that the subtree collections of the generated HTML DOM Tree and the reference are represented as sets of strings, subtree matching can be straightforwardly performed through string comparison. The TreeBLEU value can then be calculated using Equation 3.3. An example of subtree matching is illustrated in Figure 8.

## C Details of Manual Annotation

### C.1 Motivating Examples

After performing extensive data cleansing and formatting on our large-scale web-crawled dataset—including length filtering, removal of unnecessary tags and attributes, and style consolidation—we conducted a manual sampling review. This review revealed that many instances still exhibited structural and aesthetic flaws (Figure 9), as evidenced by the rendered screenshots, often due to missing styles or broken image links. To address these issues, we implemented a web screenshot quality scoring system, trained on manually labeled data, to further refine the dataset through an additional, meticulous screening process.

### C.2 Scoring Tool

To enhance the efficiency of manual data labeling, we developed an image scoring tool using Gradio. As illustrated in Figure 10, the tool features a screenshot display area on the left and a scoring interface on the right. The right side includes selectable options for evaluating image quality, along with buttons for navigating data entries, labeling the current entry, and saving all annotations.

### C.3 Scoring Procedure

To ensure consistency in our final labeled dataset and reduce subjectivity among annotators, we developed a transparent grading system with five hierarchical criteria, ranging from basic to advanced, integrated into our labeling interface. Annotators are instructed to select the relevant options for each data entry, with the total number of selected options determining the score. The grading scale spans from 0 to 5 points, making it straightforward for annotators to apply and ensuring uniform data evaluation. The scoring process involves six annotators divided into two groups. Each group assesses a designated subset, and individual scores are averaged and rounded to the nearest whole number to produce the final score for each entry.

### C.4 Consistency Validation

We conduct a statistical analysis of the score distribution within two annotated cohorts, as shown in Figure 3. The graphical representation reveals noticeable disparities among the annotations, despite the consistency of the dataset and a generally aligned distribution across the three annotators within each group. This observation highlights the subjective nature of screenshot scoring. Therefore, adopting an averaging approach within each group proves to be an effective strategy for reducing the impact of subjectivity on scoring outcomes.

## D More Statistics of Dataset

We perform a comprehensive statistical analysis on the textual content of webpages, and the findings are illustrated in Figure 12.

### Prompt for using GPT-4V in webpage generation task

You are an expert Tailwind developer. You take screenshots of a reference webpage from the user, and then build single-page apps using Tailwind, HTML and JS.

- Make sure the app looks exactly like the screenshot.
- Make sure the app has the same page layout like the screenshot, i.e., the generated html elements should be at the same place with the corresponding part in the screenshot and the generated html containers should have the same hierarchy structure as the screenshot.
- Pay close attention to background color, text color, font size, font family, padding, margin, border, etc. Match the colors and sizes exactly.
- Use the exact text from the screenshot.
- Do not add comments in the code such as "`<!-- Add other navigation links as needed -->`" and "`<!-- ... other news items ... -->`" in place of writing the full code. WRITE THE FULL CODE.
- Repeat elements as needed to match the screenshot. For example, if there are 15 items, the code should have 15 items. DO NOT LEAVE comments like "`<!-- Repeat for each news item -->`" or bad things will happen.
- For images, use placeholder images from <https://placeholder.co> and include a detailed description of the image in the alt text so that an image generation AI can generate the image later. In terms of libraries,
- Use this script to include Tailwind: `<script src="https://cdn.tailwindcss.com"></script>`
- You can use Google Fonts
  - Font Awesome for icons: `<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css"></link>` Return only the full code in `<html></html>` tags.
- Do not include markdown `""` or `""html` at the start or end.

Figure 6: Prompt for using GPT-4V in webpage generation task.

Table 6: Licenses of the datasets in this work.

Dataset	License
pix2code	Apache 2.0
Common Crawl	LIMITED license
WebSight	CC-BY-4.0
Design2Code	MIT
WebCode2M	CC-BY-4.0

English constitutes approximately 50% of the corpus, with other languages such as Russian, German, Spanish, and French each representing nearly 5% of the total corpus. We also present the licenses of the datasets in our work in Table 6.

## E More Experimental Results

The Visual Score used in Table 3 and Table 4 is actually a composite metric, composed of five indicators: block, text, position, text color, and CLIP match. We list these detailed indicators in Table 7 for reference. Table 8 presents a detailed performance comparison based on traditional metrics.

## F Implementation of Filtering Inappropriate Content

The main steps and details of the cleansing process are as follows: **Step 1: Filtering harmful images in screenshots.** We employ a widely used NSFW detector [7] from Hugging Face, which classifies images as either "normal" or "NSFW" (Not Safe for Work) with high accuracy. This detector is based on a Vision Transformer (ViT) model fine-tuned on a dataset containing both "safe" and "explicit" images. It predicts two scores: "normal" and "NSFW," with lower

NSFW scores indicating a lower probability of harmful content. A conservative threshold of 0.04 was adopted, meaning only samples with NSFW scores below this value were retained.

**Step 2: Multi-language harmful keyword filtering.** We apply harmful keyword filtering to the web text using two popular GitHub repositories: bad words 1 [52] and bad words 2 [5]. The first list contains dirty, naughty, obscene, and otherwise inappropriate words in multiple languages (e.g., 'fuck', 'raping'), while the second serves as a supplementary list with additional sensitive or stop words. Samples with more than 20 occurrences of these bad words are removed. While normal web pages may occasionally contain a small number of inappropriate words, those with excessive amounts of inappropriate content, such as adult websites, tend to have significantly higher word counts.

The thresholds for both the NSFW score and the frequency of bad words were determined through experiments on the first data chunk 9. We selected thresholds that balanced maximizing sample retention with minimizing the presence of harmful content. To evaluate the effectiveness of the filters, we manually reviewed and documented the filtering results for the first chunk under different thresholds. The table below clearly indicates that an NSFW threshold of 0.04 produced the best results.

## G Page Samples Generated By WebCoder

In Figure 13, we present several sample pages generated by our WebCoder model to demonstrate its capabilities. The left side shows the original webpage screenshots from our WebCode2M dataset, while the right side displays the outputs generated by our model via end-to-end inference after being trained on this dataset. The

**Table 7: Experimental Breakdown on the Visual Scores.**

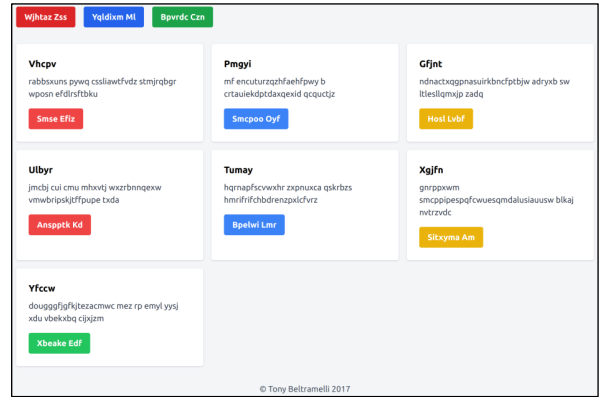
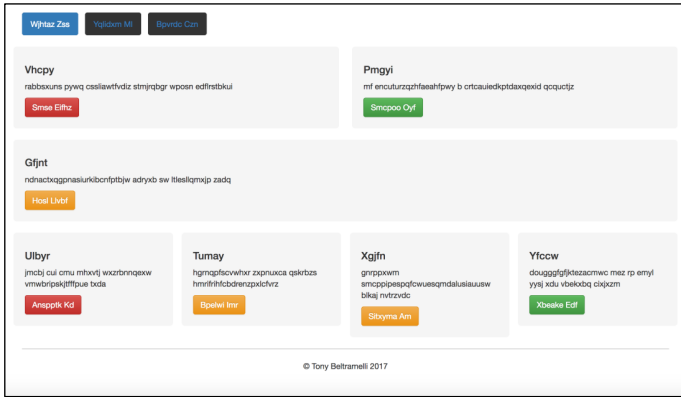
Test Set	Model	Training	Block	Text	Position	Text Color	CLIP
WebCode2M-Short	CogAgent-Chat-18B	-	0.25 ( $\pm 0.33$ )	0.54 ( $\pm 0.45$ )	0.41 ( $\pm 0.35$ )	0.45 ( $\pm 0.42$ )	0.67 ( $\pm 0.29$ )
	WebSight VLM-8B	WebSight	0.19 ( $\pm 0.26$ )	0.65 ( $\pm 0.33$ )	0.60 ( $\pm 0.32$ )	0.63 ( $\pm 0.36$ )	0.78 ( $\pm 0.20$ )
	Design2Code-18B	WebSight	0.65 ( $\pm 0.31$ )	0.92 ( $\pm 0.16$ )	0.72 ( $\pm 0.16$ )	0.68 ( $\pm 0.24$ )	0.80 ( $\pm 0.10$ )
	LLaVA-v1.5-7B	-	0.07 ( $\pm 0.11$ )	0.53 ( $\pm 0.38$ )	0.43 ( $\pm 0.33$ )	0.50 ( $\pm 0.41$ )	0.60 ( $\pm 0.33$ )
	LLaVA-onevision-0.5B	-	0.07 ( $\pm 0.18$ )	0.31 ( $\pm 0.42$ )	0.25 ( $\pm 0.35$ )	0.25 ( $\pm 0.39$ )	0.33 ( $\pm 0.38$ )
	LLaVA-onevision-7B	-	0.16 ( $\pm 0.27$ )	0.40 ( $\pm 0.44$ )	0.34 ( $\pm 0.39$ )	0.29 ( $\pm 0.36$ )	0.52 ( $\pm 0.37$ )
	Gemini	-	0.32 ( $\pm 0.44$ )	0.42 ( $\pm 0.48$ )	0.36 ( $\pm 0.42$ )	0.30 ( $\pm 0.38$ )	0.35 ( $\pm 0.40$ )
	Claude	-	0.51 ( $\pm 0.47$ )	0.57 ( $\pm 0.48$ )	0.52 ( $\pm 0.44$ )	0.48 ( $\pm 0.43$ )	0.50 ( $\pm 0.41$ )
	GPT-4V	-	0.64 ( $\pm 0.39$ )	0.79 ( $\pm 0.37$ )	0.67 ( $\pm 0.32$ )	0.59 ( $\pm 0.34$ )	0.67 ( $\pm 0.31$ )
	GPT-4o	-	0.84 ( $\pm 0.28$ )	0.95 ( $\pm 0.17$ )	0.85 ( $\pm 0.16$ )	0.79 ( $\pm 0.20$ )	0.81 ( $\pm 0.15$ )
WebCode2M-Mid	WebCoder *	WebSight	0.17 ( $\pm 0.26$ )	0.50 ( $\pm 0.42$ )	0.44 ( $\pm 0.38$ )	0.41 ( $\pm 0.40$ )	0.57 ( $\pm 0.36$ )
	WebCoder	WebCode2M	0.57 ( $\pm 0.38$ )	0.81 ( $\pm 0.31$ )	0.70 ( $\pm 0.28$ )	0.66 ( $\pm 0.32$ )	0.73 ( $\pm 0.27$ )
	CogAgent-Chat-18B	-	0.19 ( $\pm 0.29$ )	0.45 ( $\pm 0.46$ )	0.35 ( $\pm 0.37$ )	0.36 ( $\pm 0.41$ )	0.64 ( $\pm 0.30$ )
	WebSight VLM-8B	WebSight	0.13 ( $\pm 0.20$ )	0.61 ( $\pm 0.32$ )	0.56 ( $\pm 0.32$ )	0.59 ( $\pm 0.35$ )	0.73 ( $\pm 0.25$ )
	Design2Code-18B	WebSight	0.55 ( $\pm 0.35$ )	0.84 ( $\pm 0.28$ )	0.66 ( $\pm 0.24$ )	0.62 ( $\pm 0.28$ )	0.75 ( $\pm 0.23$ )
	LLaVA-v1.5-7B	-	0.03 ( $\pm 0.07$ )	0.26 ( $\pm 0.37$ )	0.22 ( $\pm 0.33$ )	0.23 ( $\pm 0.36$ )	0.29 ( $\pm 0.38$ )
	LLaVA-onevision-0.5B	-	0.06 ( $\pm 0.16$ )	0.35 ( $\pm 0.43$ )	0.28 ( $\pm 0.36$ )	0.29 ( $\pm 0.40$ )	0.41 ( $\pm 0.39$ )
	LLaVA-onevision-7B	-	0.11 ( $\pm 0.22$ )	0.34 ( $\pm 0.42$ )	0.27 ( $\pm 0.34$ )	0.28 ( $\pm 0.36$ )	0.51 ( $\pm 0.38$ )
	Gemini	-	0.39 ( $\pm 0.45$ )	0.46 ( $\pm 0.48$ )	0.38 ( $\pm 0.41$ )	0.28 ( $\pm 0.35$ )	0.39 ( $\pm 0.41$ )
	Claude	-	0.35 ( $\pm 0.45$ )	0.40 ( $\pm 0.48$ )	0.35 ( $\pm 0.42$ )	0.31 ( $\pm 0.40$ )	0.35 ( $\pm 0.41$ )
WebCode2M-Long	GPT-4V	-	0.61 ( $\pm 0.38$ )	0.78 ( $\pm 0.38$ )	0.65 ( $\pm 0.32$ )	0.58 ( $\pm 0.33$ )	0.66 ( $\pm 0.33$ )
	GPT-4o	-	0.79 ( $\pm 0.30$ )	0.92 ( $\pm 0.22$ )	0.79 ( $\pm 0.21$ )	0.77 ( $\pm 0.22$ )	0.78 ( $\pm 0.19$ )
	WebCoder *	WebSight	0.10 ( $\pm 0.18$ )	0.45 ( $\pm 0.42$ )	0.40 ( $\pm 0.38$ )	0.33 ( $\pm 0.35$ )	0.51 ( $\pm 0.38$ )
	WebCoder	WebCode2M	0.46 ( $\pm 0.34$ )	0.83 ( $\pm 0.22$ )	0.72 ( $\pm 0.22$ )	0.70 ( $\pm 0.26$ )	0.77 ( $\pm 0.21$ )
	CogAgent-Chat-18B	-	0.22 ( $\pm 0.32$ )	0.43 ( $\pm 0.45$ )	0.33 ( $\pm 0.36$ )	0.32 ( $\pm 0.38$ )	0.66 ( $\pm 0.26$ )
	WebSight VLM-8B	WebSight	0.13 ( $\pm 0.22$ )	0.59 ( $\pm 0.37$ )	0.50 ( $\pm 0.33$ )	0.54 ( $\pm 0.36$ )	0.65 ( $\pm 0.30$ )
	Design2Code-18B	WebSight	0.49 ( $\pm 0.36$ )	0.77 ( $\pm 0.35$ )	0.58 ( $\pm 0.28$ )	0.55 ( $\pm 0.28$ )	0.66 ( $\pm 0.29$ )
	LLaVA-v1.5-7B	-	0.04 ( $\pm 0.11$ )	0.23 ( $\pm 0.36$ )	0.19 ( $\pm 0.30$ )	0.20 ( $\pm 0.32$ )	0.28 ( $\pm 0.37$ )
	LLaVA-onevision-0.5B	-	0.06 ( $\pm 0.16$ )	0.27 ( $\pm 0.40$ )	0.22 ( $\pm 0.34$ )	0.22 ( $\pm 0.35$ )	0.35 ( $\pm 0.37$ )
	LLaVA-onevision-7B	-	0.11 ( $\pm 0.22$ )	0.35 ( $\pm 0.42$ )	0.28 ( $\pm 0.35$ )	0.28 ( $\pm 0.37$ )	0.47 ( $\pm 0.37$ )
WebCode2M-Long	Gemini	-	0.35 ( $\pm 0.45$ )	0.40 ( $\pm 0.48$ )	0.34 ( $\pm 0.40$ )	0.28 ( $\pm 0.34$ )	0.33 ( $\pm 0.40$ )
	Claude	-	0.39 ( $\pm 0.46$ )	0.42 ( $\pm 0.48$ )	0.36 ( $\pm 0.42$ )	0.33 ( $\pm 0.39$ )	0.35 ( $\pm 0.41$ )
	GPT-4V	-	0.60 ( $\pm 0.40$ )	0.74 ( $\pm 0.41$ )	0.62 ( $\pm 0.36$ )	0.53 ( $\pm 0.32$ )	0.60 ( $\pm 0.34$ )
	GPT-4o	-	0.84 ( $\pm 0.26$ )	0.93 ( $\pm 0.20$ )	0.79 ( $\pm 0.19$ )	0.75 ( $\pm 0.19$ )	0.78 ( $\pm 0.19$ )
	WebCoder *	WebSight	0.14 ( $\pm 0.23$ )	0.47 ( $\pm 0.42$ )	0.40 ( $\pm 0.36$ )	0.32 ( $\pm 0.32$ )	0.55 ( $\pm 0.35$ )
	WebCoder	WebCode2M	0.46 ( $\pm 0.34$ )	0.80 ( $\pm 0.25$ )	0.66 ( $\pm 0.23$ )	0.62 ( $\pm 0.26$ )	0.73 ( $\pm 0.22$ )

**Table 8: The performance breakdown on classic metrics.**

Model	WebCode2M-Short				WebCode2M-Mid				WebCode2M-Long			
	BLEU	Rough-1	MSE	SSIM	BLEU	Rough-1	MSE	SSIM	BLEU	Rough-1	MSE	SSIM
CogAgent-Chat	.33 $\pm$ .33	.40 $\pm$ .33	.35 $\pm$ .38	.59 $\pm$ .15	.27 $\pm$ .31	.31 $\pm$ .30	.32 $\pm$ .30	.58 $\pm$ .14	.22 $\pm$ .28	.26 $\pm$ .28	.36 $\pm$ .30	.60 $\pm$ .13
WebSight VLM	.17 $\pm$ .19	.20 $\pm$ .20	.32 $\pm$ .36	.62 $\pm$ .17	.10 $\pm$ .14	.11 $\pm$ .14	.37 $\pm$ .38	.59 $\pm$ .16	.11 $\pm$ .16	.13 $\pm$ .17	.39 $\pm$ .27	.61 $\pm$ .15
Design2Code	.50 $\pm$ .32	.56 $\pm$ .29	.39 $\pm$ .36	.58 $\pm$ .15	.45 $\pm$ .31	.51 $\pm$ .28	.35 $\pm$ .28	.56 $\pm$ .14	.33 $\pm$ .29	.41 $\pm$ .26	.37 $\pm$ .28	.61 $\pm$ .11
LLaVA-v1.5-7B	.04 $\pm$ .07	.06 $\pm$ .06	.35 $\pm$ .37	.65 $\pm$ .17	.03 $\pm$ .05	.04 $\pm$ .05	.33 $\pm$ .35	.65 $\pm$ .14	.03 $\pm$ .05	.05 $\pm$ .05	.34 $\pm$ .28	.66 $\pm$ .12
LLaVA-onevision-0.5B	.11 $\pm$ .17	.17 $\pm$ .18	.38 $\pm$ .42	.53 $\pm$ .24	.07 $\pm$ .11	.12 $\pm$ .15	.35 $\pm$ .27	.49 $\pm$ .22	.08 $\pm$ .13	.13 $\pm$ .15	.37 $\pm$ .27	.55 $\pm$ .20
LLaVA-onevision-7B	.13 $\pm$ .21	.20 $\pm$ .22	.31 $\pm$ .33	.63 $\pm$ .15	.06 $\pm$ .14	.12 $\pm$ .16	.30 $\pm$ .28	.60 $\pm$ .14	.05 $\pm$ .12	.11 $\pm$ .15	.33 $\pm$ .27	.63 $\pm$ .12
Gemini	.64 $\pm$ .28	.75 $\pm$ .20	.32 $\pm$ .30	.62 $\pm$ .15	.68 $\pm$ .23	.74 $\pm$ .21	.33 $\pm$ .26	.60 $\pm$ .14	.63 $\pm$ .22	.69 $\pm$ .19	.39 $\pm$ .31	.61 $\pm$ .12
Claude	.74 $\pm$ .27	.81 $\pm$ .22	.29 $\pm$ .29	.61 $\pm$ .14	.74 $\pm$ .23	.79 $\pm$ .20	.30 $\pm$ .23	.59 $\pm$ .12	.68 $\pm$ .22	.73 $\pm$ .18	.33 $\pm$ .28	.61 $\pm$ .12
GPT-4V	.69 $\pm$ .28	.74 $\pm$ .24	.30 $\pm$ .25	.61 $\pm$ .14	.55 $\pm$ .32	.62 $\pm$ .28	.34 $\pm$ .29	.55 $\pm$ .12	.49 $\pm$ .29	.58 $\pm$ .23	.38 $\pm$ .28	.57 $\pm$ .11
GPT-4o	.73 $\pm$ .26	.80 $\pm$ .21	.27 $\pm$ .19	.60 $\pm$ .13	.70 $\pm$ .25	.74 $\pm$ .23	.27 $\pm$ .20	.58 $\pm$ .12	.67 $\pm$ .23	.71 $\pm$ .20	.30 $\pm$ .21	.60 $\pm$ .11
WebCoder *	.23 $\pm$ .23	.26 $\pm$ .22	.37 $\pm$ .33	.58 $\pm$ .17	.16 $\pm$ .17	.17 $\pm$ .18	.34 $\pm$ .28	.56 $\pm$ .16	.15 $\pm$ .17	.17 $\pm$ .17	.44 $\pm$ .33	.59 $\pm$ .15
WebCoder	.58 $\pm$ .26	.61 $\pm$ .24	.41 $\pm$ .44	.59 $\pm$ .17	.46 $\pm$ .27	.47 $\pm$ .26	.39 $\pm$ .37	.56 $\pm$ .16	.40 $\pm$ .26	.41 $\pm$ .24	.47 $\pm$ .44	.58 $\pm$ .16

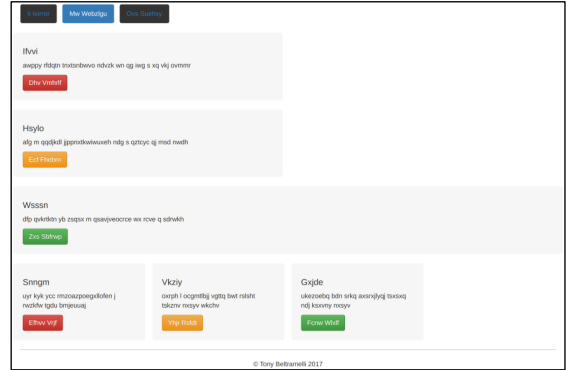
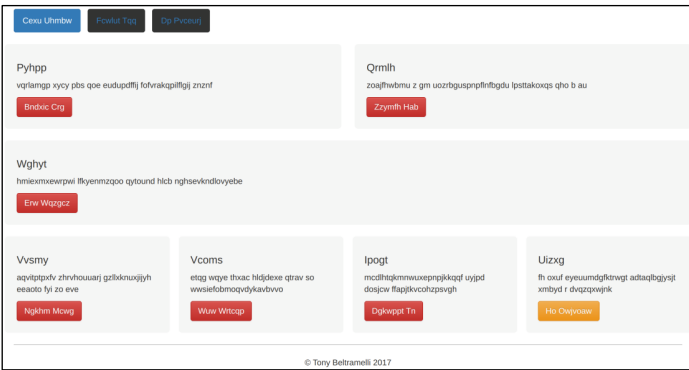
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566

1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624



Reference

GPT-4V



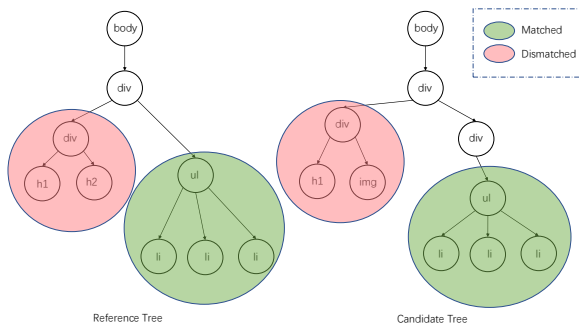
pix2code

Pix2struct

Figure 7: A representative example on the pix2code test dataset.

Table 10: The filtering performance across different NSFW threshold values.

Threshold	Total Num	Removed by nsfw filter			Retained after nsfw filter				Retained after nsfw filter bad word filter			
		total num	good num	miskill ratio	total num	bad num	retention ratio	toxic ratio	total num	bad num	retention ratio	toxic ratio
0.02	1536	891	883	57.49%	645	0	41.99%	0.00%	645	0	41.99%	0.00%
0.03		324	318	20.70%	1212	2	78.91%	0.17%	1210	0	78.78%	0.00%
0.04		133	128	8.33%	1403	3	91.34%	0.21%	1400	0	91.15%	0.00%
0.05		70	66	4.30%	1466	4	95.44%	0.27%	1463	1	95.25%	0.07%
1.00		0	0	0.00%	1536	8	100.00%	0.52%	1530	2	100.00%	0.13%



images were processed using OCR, extracted from the original screenshots, and embedded into the predicted HTML code. The results indicate that our model effectively preserves both structural and stylistic consistency while achieving high accuracy in text content prediction.

Figure 8: A subtree matching example in the metric of Tree-BLEU.

1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682

1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740

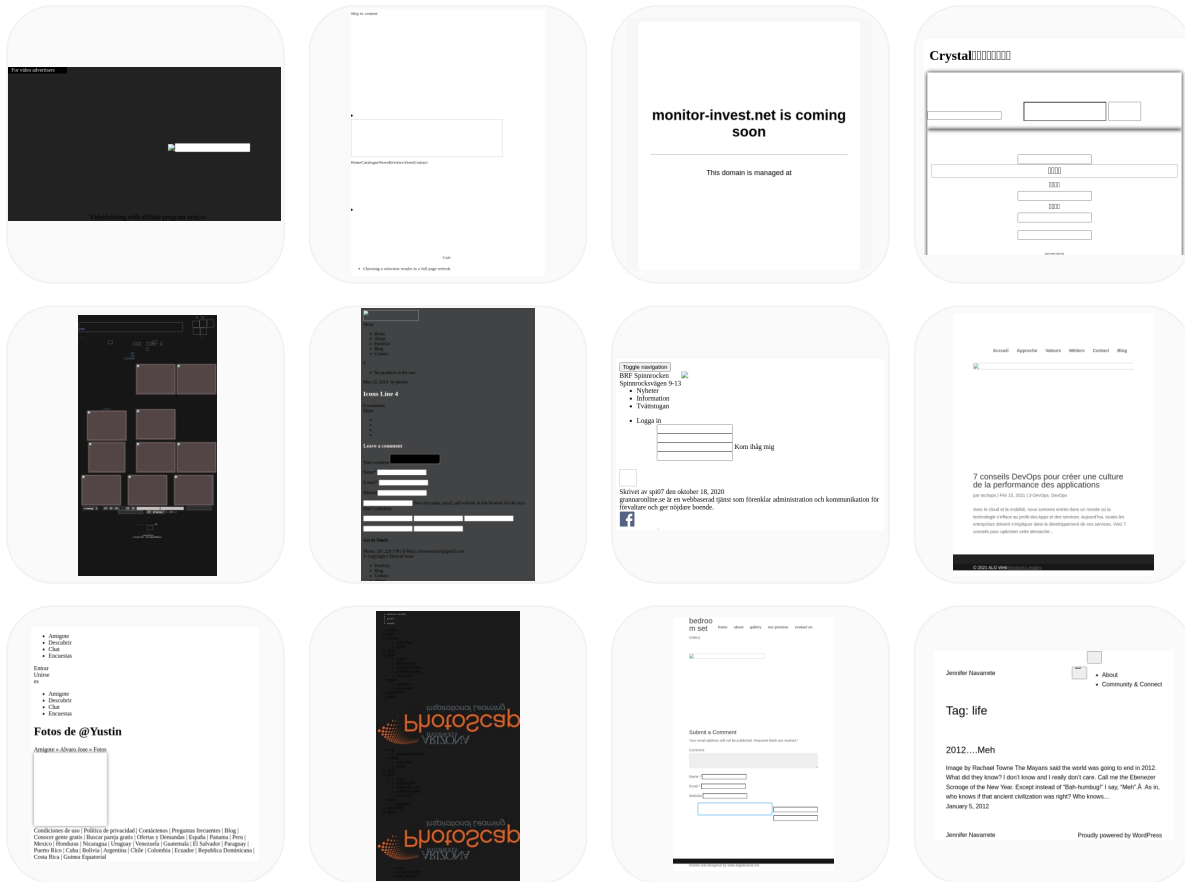


Figure 9: Low-quality samples.

Vision2UI Scoring Tool

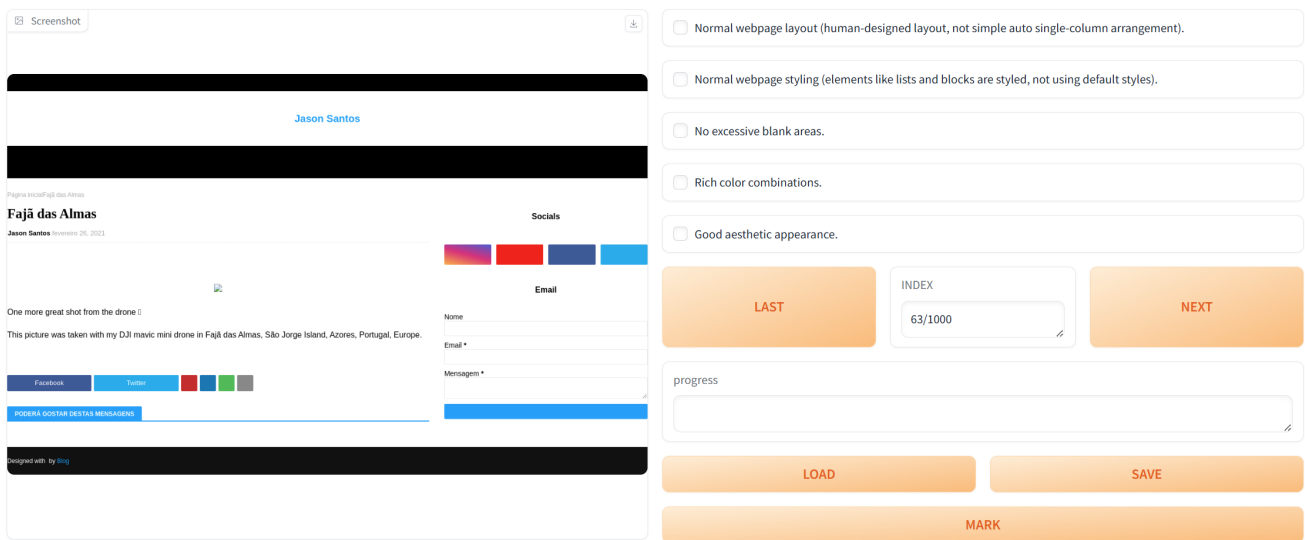


Figure 10: The manual scoring tool.

1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798

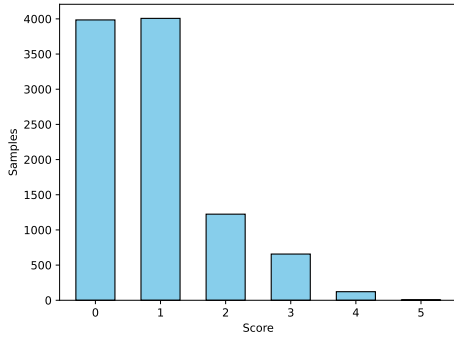


Figure 11: The distribution of manually scored results.

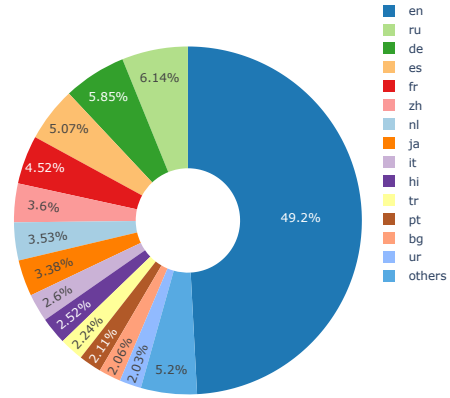


Figure 12: Language distribution in the training dataset.

### H Our Dataset on HuggingFace

We present a screenshot of our dataset uploaded to HuggingFace in Figure 14 and Figure 15. Detailed data cards and specifications for the dataset are available on HuggingFace.

1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856



1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914

1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972



Figure 13: Comparison of an original webpage (input) on the left, and the rendering of the HTML generated by our model, WebCoder, (output) on the right.

1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030

2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088

image	bbox	text
image · width (px)  1.28k 1.97k	string · lengths  4 140k	string · lengths  1.47k 32.2k
	<code>{"type": "body", "content": "", "style": "color: #111; font-family: \\"Arimo\\", Helvetica, Arial,..."}</code>	<code>&lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt;&lt;title&gt;Mercado CE archivos - Certiberia&lt;/title&gt;&lt;/head&gt;&lt;body...</code>
	<code>{"type": "body", "content": "", "style": "font-family: Trebuchet MS, Helvetica, sans-serif; line-..."}</code>	<code>&lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt;&lt;title&gt;Форекс Тренд - претензия потерпевшего (видео с объективным...</code>
	<code>{"type": "body", "content": "", "style": "margin: 0; padding-top: 0; padding-right: 0; padding-..."}</code>	<code>&lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt;&lt;title&gt;Covid-19 Archives - Freelance writer: leadership, personal...</code>
	<code>{"type": "body", "content": "", "style": null, "bbox": [8, 8, 1264, 2126], "children": [{"type": "..."</code>	<code>&lt;!DOCTYPE html&gt; &lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt;&lt;title&gt;Dr. Brooke Wooten   Foothills Veterinary...</code>
	<code>{"type": "body", "content": "", "style": "font: 100% Verdana, Arial, Helvetica, sans-serif; margin-..."}</code>	<code>&lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt;&lt;title&gt;E-Flora BC Contact Us Form&lt;/title&gt;&lt;/head&gt;&lt;body...</code>
	<code>{"type": "body", "content": "", "style": "text-size-adjust: 100%; margin: 0; box-sizing: inherit;..."}</code>	<code>&lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt;&lt;title&gt;How to Get Started in Online Poker - ...</code>

Figure 14: Examples of WebCode2M dataset uploaded to HuggingFace (partial columns)






score	scale	lang	tokens	hash
int64  3 3	sequence · lengths  2 2	string · classes  20 values	sequence · lengths  2 2	string · lengths  64 64
3	[ 1280, 1561 ]	es	[ 678, 3344 ]	370d891e8aa1b9eb1d935a6a76bccbf21a7329f0f0a6b3c4d05cb9fc4687318e
3	[ 1280, 1914 ]	ru	[ 1742, 4326 ]	141a6998fe0b9578f5b2fd482bc44bf400e5ee950775d5c8a3da10e1ff406500
3	[ 1280, 1705 ]	en	[ 4081, 3733 ]	6ba400fc0d1912d8ce2b9304d769329493c2b89e82ca518a00cdfcc0fcc1cd2
3	[ 1302, 2142 ]	en	[ 4063, 4729 ]	02bc15699aa9151927d94f2e05e0ae0d987f648e6a3fccd009df626c563f061d
3	[ 1280, 830 ]	en	[ 199, 796 ]	e422705a884c0c73fd977af39328a00f1b69dedee61296f0e1fa6848b0ea1bc
3	[ 1370, 1683 ]	en	[ 2144, 3474 ]	a35a1c4750f514ea87e23f08c213de2e0e9ae284645b6cfc2545f1682498f813

Figure 15: Examples of WebCode2M dataset uploaded to HuggingFace (partial columns)