# VISION2UI: A Real-World Dataset with Layout for Code Generation from UI Designs

Yi Gui<sup>1</sup>\*, Zhen Li<sup>1</sup>\*, Yao Wan<sup>1</sup>†, Yemin Shi<sup>3</sup>, Hongyu Zhang<sup>4</sup>, Yi Su<sup>5</sup>, Shaoling Dong<sup>2</sup>, Xing Zhou<sup>2</sup>†, Wenbin Jiang<sup>1</sup>

National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
Rabbitpre AI, <sup>3</sup>Peking University, <sup>4</sup>Chongqing University
School of Electrical & Information Engineering, Hubei University of Automotive Technology {guiyi, ninth, wanyao, wenbinjiang}@hust.edu.cn, shiyemin@pku.edu.cn

## **Abstract**

hyzhang@cqu.edu.cn, 20230053@huat.edu.cn, {dong, zhouxing}@tuzhanai.com

Automatically generating UI code from webpage design visions can significantly alleviate the burden of developers, enabling beginner developers or designers to directly generate Web pages from design diagrams. Currently, prior research has accomplished the objective of generating UI code from rudimentary design visions or sketches through designing deep neural networks. Inspired by the groundbreaking advancements achieved by Multimodal Large Language Models (MLLMs), the automatic generation of UI code from high-fidelity design images is now emerging as a viable possibility. Nevertheless, our investigation reveals that existing MLLMs are hampered by the scarcity of authentic, high-quality, and large-scale datasets, leading to unsatisfactory performance in automated UI code generation. To mitigate this gap, we present a novel dataset, termed VI-SION2UI, extracted from real-world scenarios, augmented with comprehensive layout information, tailored specifically for finetuning MLLMs in UI code generation. Specifically, this dataset is derived through a series of operations, encompassing collecting, cleaning, and filtering of the open-source Common Crawl dataset. In order to uphold its quality, a neural scorer trained on labeled samples is utilized to refine the data, retaining higher-quality instances. Ultimately, this process yields a dataset comprising 2,000 (Much more is coming soon) parallel samples encompassing design visions and UI code. The dataset is available at https://huggingface.co/datasets/xcodemind/vision2ui.

## 1 Introduction

The automation of generating UI code from webpage design visions (images) not only relieves developers of burdensome tasks but also empowers novices and designers to effortlessly translate design diagrams into UI code, thereby holding considerable promise for application and market value. Currently, prior research has accomplished the objective of generating UI code from rudimentary design visions or sketches by designing deep neural networks. For example, Beltramelli [1] introduced pix2code, a model combining CNN and LSTM architectures trained on a synthetic

<sup>\*</sup>These authors contributed equally to this work.

<sup>&</sup>lt;sup>†</sup>Yao Wan and Xing Zhou are the corresponding authors.

Table 1: The performance comparision on the pix2code test dataset. The pix2code model and Pix2Struct model are both finetuned on the pix2code training dataset. The ChatGPT-4V is prompted to generate HTML code in one-shot mode (the prompts is provided in Appendix 6.1).

Model	TreeBLEU	SSIM	Error
pix2code-Beam	0.98	0.85	0.02
pix2code-Gready	0.99	0.85	0.02
Pix2Struct-282M	0.79	0.86	0.0
Pix2Struct-1.3B	0.92	0.86	0.0
ChatGPT-4V	0.09	0.84	0.0

dataset featuring basic webpage components. Their objective was to generate code directly from screenshots of webpages. Similarly, Robinson [20] developed Sketch2code, which endeavors to translate hand-drawn webpage sketches into webpage code. This work also explored both computer vision-based approach and deep learning-based approach. Recently, Multimodal Large Language Models (MLLMs), exemplified by ChatGPT-4V [15], have shown impressive capabilities in understanding images. Inspired by this, the automatic generation of UI code from high-fidelity design images is now emerging as a viable possibility.

**Empirical Analysis of Existing Works.** We empirically analyze the performances of pix2code, Pix2Struct, and ChatGPT-4V to illustrate the motivation of our work. We adopt two metrics to measure the performance of code generation from design pictures. Firstly, inspired by [19], we propose a new metric, named TreeBLEU, to evaluate the matching degree of the generated HTMLs' DOM tree compared to the ground truth. TreeBLEU is defined as the proportion of all 1-height subtrees in a given tree that can be matched with that of a reference tree. Let S(.) be the set of 1-height subtrees, then it can be formulated as follows:

TreeBLEU = 
$$\frac{|S(t) \cap S(t_r)|}{|S(t_r)|},$$
 (1)

where t and  $t_r$  denote the given tree and the reference tree, respectively.

Additionally, in order to assess the visual correspondence between the rendered page derived from the generated HTML and the ground-truth design image, we adopt SSIM [27] as an additional metric. It represents the structure of objects in the image, independent of the average luminance and contrast.

Table 1 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-shot 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, as shown in Table 1. The primary explanation for this disparity lies in the fact that both models have been fine-tuned on specific task datasets, while ChatGPT-4V has not.

**Proposed Dataset.** To unleash the potential of MLLMs in code generation, we propose a real-world dataset with the layout information, termed VISION2UI, for UI code generation in this paper. We preprocess the data from the Common Crawl<sup>3</sup> dataset, download corresponding CSS code and image elements, eliminate the noise code, and generate screenshots of HTML pages. In addition, our dataset contains the layout of all the HTML elements, i.e., the size and position information. To further improve the quality of the dataset, we train a neural scorer from a manually scored subset to remove the subset with low scores. We split the curated dataset into a training set for model learning and a testing set as a benchmark. Our dataset is composed of 20,000 samples (much more is coming soon), with 16,000 being the training set, 2,000 being the validation set, and 2,000 being the test set.

Two contemporary works to ours are Design2Code [22] and WebSight [6]. The former proposed a benchmark that is meticulously selected by human experts and composed of 484 diverse Web pages for testing. The latter proposed a synthesized dataset, of which the content is generated by LLMs and the image elements are filled with selected placeholder images. Our dataset differs from them based on the following features: 1) Our dataset, sourced from the real world, features a more

<sup>&</sup>lt;sup>3</sup>https://data.commoncrawl.org/

abundant and authentic structure and image elements. It offers a superior level of diversity and authenticity compared to WebSight. 2) Compared to Design2Code, our dataset with 20,000 samples is considerably larger, providing both a training set for model learning and a test set for evaluation. 3) Our dataset not only includes pairs of HTML code and design images but also encompasses the layout of webpage elements. This facilitates LLMs to more effectively learn the ability to generate appropriate structures of webpages.

**Contribution.** We propose a real-world dataset for empowering MLLMs to train and test on the task of generating HTML code from high-fidelity images. This dataset contains the layout of design images, which further facilitates MLLMs to converge at the training stage.

# 2 VISION2UI: The Dataset

Our goal is to construct a dataset in the format of (HTML code, UI design) pairs. The screenshots of websites are basically equivalent to the corresponding designs of the websites. Consequently, we chose to crawl and clean HTML code from networks, generate screenshots, and filter the paired data to create our dataset. The overview of our data pipeline is depicted in Figure 1.

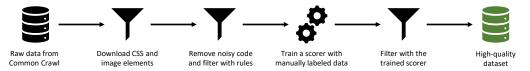


Figure 1: The pipeline of dataset construction.

#### 2.1 Dataset Construction

**Data Collection.** Since directly crawling websites is time-consuming and labor-intensive work, we opt to create our dataset on top of the Common Crawl<sup>3</sup> dataset. The Common Crawl dataset is a vast collection of Web page data from 2013 to now, acquired and updated by monthly Web crawling. The fundamental data we have selected is from November to December 2023. This partial dataset from the original Common Crawl dataset includes 3.35 billion pages, amounting to a total of 135.40TB data. Each entry of the original dataset is made up of only HTML text without CSS and image elements. We download the corresponding CSS code of each HTML and insert it into the HTML text. The image elements will be downloaded during the procedure of generating screenshots.

**Data Cleaning.** The primary objective of this study is to curate a dataset that enables neural models to acquire the capability of generating corresponding static HTML code from user interface (UI) design images. Under ideal circumstances, the HTML code at each point in the dataset should be identically translated into the respective UI design image. To better facilitate LLMs to align HTML code with the corresponding design images, we have meticulously cleaned the amalgamated text of HTML and CSS, sourced from the preceding steps, by adhering to the following heuristic rules:

- Redundancy code cleaning. Redundancy code such as comments, invisible elements, and other code that has no direct relation to the rendering effects of static HTML pages, will introduce noise in the model training procedure. Consequently, we remove them with the following rules.
  - Remove all comments in the HTML and CSS parts.
  - Remove all < meta > and < script > tags.
  - Remove the invisible (hidden, zero-sized, or outside the display range) HTML elements.
  - Delete the attributes not in (class, id, width, height, style, src) of all HTML elements.
  - Remove the href attribute of tag < a >.
  - Remove the CSS styles not effective in HTML code.
- Length filtering. Too short HTML or CSS is usually due to parsing errors or other issues of the Web pages. Meanwhile, too long input context will significantly slow down the training and inference procedure. Thus we apply the following length filters.
  - CSS text length should be between  $128 \times 5$  and  $4096 \times 5$  characters.

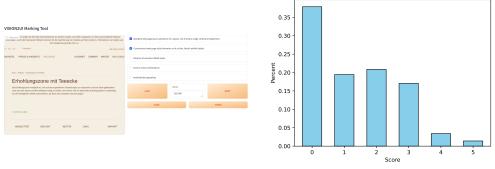
- HTML text length should be between  $128 \times 5$  and  $2056 \times 5$  characters.

This rigorous process ensures that the resultant dataset aligns closely with the UI design images and subsequently facilitates the development of more accurate and efficient neural models. All cleaning work is mainly implemented through the Python toolkit BeautifulSoup<sup>4</sup> and cssutil<sup>5</sup>, and is combined with multiple processes to speed up the processing of the massive data.

Screenshots Generation. Upon the cleaned data, we utilize Pyppeteer<sup>6</sup> to obtain webpage screenshots from the merged code which combines HTML and CSS. In the process of generating HTML code, a critical evaluation metric is the extent to which the hierarchical structure of the webpage's Document Object Model (DOM) tree aligns with the original HTML. This alignment is essential as it ensures the consistency and integrity of the Web page's structure, thereby ensuring accurate rendering and enhancing user experience. To facilitate the model's learning of webpage DOM tree structure generation, we simultaneously generate the location and size information for each HTML element when creating webpage screenshots. This information serves as one of the training labels for the model. In other words, each actual data point transforms into a triplet (HTML code, UI design, Layout).

**Filtering with Neural Scorer.** In our empirical data analysis, we note a significant portion of the collected screenshots exhibit deficiencies in aesthetics, including incompletely loaded pages stemming from factors such as invalid image links, as well as instances where the content consists predominantly of textual elements. The presence of these factors markedly undermines the overall quality of the dataset, necessitating a more rigorous examination of the acquired data. Due to the sheer volume of data, manually screening the entire dataset is unfeasible. Hence, we leverage a machine learning model as a neural scorer to assess the screenshots, subsequently eliminating data points that fall below a specified score threshold. Initially, we devise a scoring tool (Figure 2 (a)) and manually evaluate a subset of the screenshots within the dataset for training the neural scorer. The scoring criteria are meticulously delineated below, and each criterion satisfied will be awarded one point:

- Standard Web page layout (presence of a layout, not merely a single vertical arrangement)
- Conventional Web page style (elements such as lists, blocks exhibit styles)
- · Absence of excessive blank styles
- Diverse color combinations
- · Aesthetically appealing



- (a) The manually scoring tool
- (b) The distribution of manually scored result

Figure 2: The scoring tool and result

<sup>&</sup>lt;sup>4</sup>https://www.crummy.com/software/BeautifulSoup/

<sup>&</sup>lt;sup>5</sup>https://github.com/jaraco/cssutils

<sup>&</sup>lt;sup>6</sup>https://github.com/pyppeteer

Table 2: A statistical comparison between our	dataset and both	WebSight and Design2Code	. The
statistical data of the two is referred to [22].			

	WebSight	Design2Code	VISION2UI (Ours)
Purpose	Training	Testing	Training&Tesing
Source	Synthetic	Real-World (C4)	Real-World (Common Crawl)
Size	823K	484	20k (Much more is coming soon)
Avg. Len (tokens)	647±216	31216±23902	8460±7120
Avg. Tags	19±8	158±100	175±94
Avg. DOM Depth	5±1	13±5	15±5
Avg. Unique Tags	10±3	22±6	21±5

The score distribution of the manually labeled subset is depicted in Figure 2 (b). Utilizing the rated subset of data, we trained a ResNet-50 model to serve as a scorer, which achieved a 78% accuracy on the test part of the manually scored subset. Employing these scoring criteria, we evaluated the cleaned dataset and ultimately retained data points with scores of two or above. The scorer achieved nearly 95% accuracy in binary classification which determines whether the score is greater than or equal to 2. Empirical evidence demonstrates that this methodology, based on neural network models, not only accelerates the screening process significantly but also markedly enhances the quality of the dataset.

#### 2.2 Data Statistics

To quantitatively measure the diversity of our dataset, we adopt the same statistical metrics as those in Design2Code, with the results presented in Table 2. The *Avg. Len* indicates the token length obtained through the GPT-2 tokenizer; *Avg. Tags* refers to the number of tags in the HTML code; *Avg. Unique Tags* denotes the number of unique tags in the HTML code, and Avg. *DOM Depth* signifies the maximum depth of the HTML's DOM Tree.

As shown in Table 2, our dataset exhibits significant advantages in terms of diversity compared to WebSight, which is due to the fact that WebSight is generated by LLMs, and thus its distribution deviates from that of real-world datasets. Compared to Design2Code, our dataset is essentially consistent in terms of diversity, as the C4 dataset it uses also originates from Common Crawl; However, the scale of our dataset far exceeds it (the figure of 20,000 is only a temporary release quantity); HTML text and CSS text are shorter due to the cleaning and filtering rules applied, allowing for better alignment between the two modalities. We will be releasing a dataset with over one million samples soon.

In Figure 3, we also showcase twelve screenshots of webpages from our dataset VISION2UI, demonstrating its exceptional diversity across various dimensions including element types, layout, structure, and coloration.

# 3 Related Work

**Code Generation.** Recently, notable advancements have been made in code generation by various pre-trained code language models. For instance, CodeGPT [11], a Transformer-based model, has undergone training utilizing a corpus tailored for program synthesis, following a similar architecture to GPT-2. Another model, CodeT5 [25], is grounded on T5 [17] and encompasses pre-training across eight programming languages, integrating an identifier-aware objective during its pre-training phase. Additionally, Codex [2], a GPT model, has been trained on a code corpus derived from GitHub and has notably served as the foundational framework for Copilot<sup>7</sup>. Moreover, AlphaCode, introduced by Li et al. [10], stands out as a code generation system tailored to produce unique solutions for intricate problems necessitating deep cognitive engagement. More recently, the landscape of code generation has been largely influenced by LLMs, such as CodeGen [13], CodeT5+ [26], InCoder [4], GPT-3.5 [14], StarCoder [9], Code Llama [21], and WizardCoder [12].

<sup>&</sup>lt;sup>7</sup>https://github.com/features/copilot

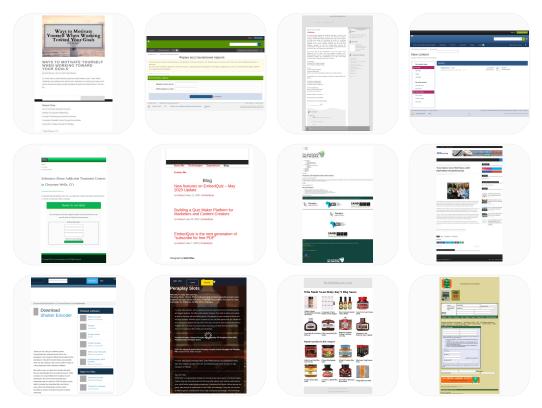


Figure 3: Sample screenshots of Web pages in VISION2UI.

Image to Code. To reverse engineer code from Graphical User Interface (GUI), Beltramelli [1] proposed pix2code, a model employing CNN [7] and Long Short-Term Memory [5] (LSTM) as image encoder and code decoder respectively. The model was trained on a synthesized dataset comprising both GUI screenshots and associated source code for three platforms, enabling the generation of Domain Specified Language (DSL) code which can be compiled into IOS, Andriod or Website-based GUI code. Microsoft's Sketch2Code [20] generates website codes from wireframe sketches exploring two methods: a computer vision-based approach by identifying elements and structures, and a deep learning-based approach by translating website sketches into normalized websites to generate HTML code. Wu et al. [28] formulated the problem of screen parsing, predicting UI hierarchy graphs from screenshots using Faster-RCNN [18] for encoding screenshot image and LSTM attention mechanism for graph codes and edges construction. Pretrained by learning to predict simplified HTML from masked screenshots of websites, Pix2Struct [8] significantly improved visual language understanding on nine tasks across four domains. To mitigate the burden and nondifferentiable issue of website rendering, Soselia et al. [23] applied reinforcement learning to finetune a vision-code Transformer (ViCT), comprising a visual Transformer [3] (ViT) and a GPT-2/Llamabased [24, 16] code decoder, by minimizing visual discrepancy between the original and generated HTML code without rendering.

## 4 Discussion

#### 4.1 Practical Challenges of Automatic HTML Code Generation from Designs

**Lenghy CSS.** CSS is lengthy and complex, with all elements aggregated together. As shown in Table 2, despite our efforts to clean up noise in the CSS, such as invisible elements, the CSS text remains lengthy, reflecting its complexity to some extent. In fact, we have attempted to train a MLLM to directly generate HTML code from images in the dataset, but this large block of CSS still presents significant challenges to the model's training. Another potential solution is to create a dataset entirely

in the Tailwind CSS style. In this case, with style and elements more concentrated, the model might be easier to train.

Generation of HTML DOM Tree Structure. 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 is indeed a challenging task. Based on our empirical study of ChatGPT-4V, we found that it performs well in capturing the text and color from images when generating webpage code. However, the structure of the generated code can be problematic. For instance, it might incorrectly convert a vertical list into a horizontal one. The empirical test results presented in Table 1 highlight its deficiencies in generating HTML structures. Therefore, designing a model that is more proficient in generating the hierarchy structure for the translation of design diagrams into webpage code is promising.

#### 4.2 Future Work

The size of the dataset released this time is 20,000, which is only a small portion of the total dataset. We will soon release datasets on a scale exceeding 1 million. Moreover, we will further enhance the quality of the dataset by introducing human evaluation. Meanwhile, we have started to use the dataset to finetune MLLMs, but due to limitations in computational resources, we have not yet finished, so we have not released the final evaluation results. We are also attempting to address the challenges mentioned above from the aspects of the dataset, model design, and engineering methods. Ultimately, our goal is to propose a practical and effective tool for directly generating UI code from high-definition design diagrams, which not only alleviates the burden of developers but also enables beginner developers or designers to directly generate Web pages from design diagrams.

## 5 Conclusion

In this paper, we have proposed VISION2UI, a real-world dataset with layout information for generating UI code from design images. This dataset consists of 20,000 samples for both training and testing, facilitating the model training and evaluation. 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 believe that the dataset proposed in this work can further advance related research in automatic UI code generation from design images.

## References

- [1] 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. 1, 6
- [2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 5
- [3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua 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. 6
- [4] 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. 5
- [5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 1997. URL https://api.semanticscholar.org/CorpusID:1915014. 6
- [6] Hugo Laurenccon, 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. 2

- [7] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989. URL https://api.semanticscholar.org/CorpusID:41312633. 6
- [8] Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang 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. 6
- [9] 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. 5
- [10] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, et al. Competition-level code generation with alphacode. *Science*, 378(6624): 1092–1097, 2022. 5
- [11] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *NeurIPS Datasets and Benchmarks*, 2021. 5
- [12] 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. 5
- [13] 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. 5
- [14] OpenAI. ChatGPT. https://openai.com/blog/chatgpt/, 2022. 5
- [15] 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. 2
- [16] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL https://api.semanticscholar. org/CorpusID:160025533. 6
- [17] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21: 1–67, 2020. 5
- [18] 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. 6
- [19] 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. 2
- [20] Alex Robinson. Sketch2code: Generating a website from a paper mockup. CoRR, abs/1905.13750, 2019. URL http://arxiv.org/abs/1905.13750. 2, 6

- [21] Baptiste Roziere, 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. 5
- [22] 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. 2, 5
- [23] 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. 6
- [24] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023. URL https://api.semanticscholar.org/CorpusID:257219404. 6
- [25] 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. In *EMNLP*, pages 8696–8708, 2021. 5
- [26] 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. 5
- [27] Zhou Wang, Alan Conrad Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612, 2004. URL https://api.semanticscholar.org/CorpusID:207761262. 2
- [28] 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. 6

# 6 Appendix

# 6.1 The Prompt Used for ChatGPT-4V in UI Code Generation from Images

The prompt presented here is primarily referenced from the open-source project screenshot-to-code<sup>8</sup>.

You are an expert Tailwind developer. You take screenshots of a reference Web page 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 gereated 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.

<sup>&</sup>lt;sup>8</sup>https://github.com/abi/screenshot-to-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://placehold.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.