WebUIBench: A Comprehensive Benchmark for Evaluating Multimodal Large Language Models in WebUI-to-Code

Zhiyu Lin[1,3*†], Zhengda Zhou[1,4*†], Zhiyuan Zhao[1*], Tianrui Wan[2], Yilun Ma[2], Junyu Gao[1,2‡], Xuelong Li[1‡]

[1]Institute of Artificial Intelligence (TeleAI), China Telecom,
[2]Northwestern Polytechnical University, [3]Beijing Jiaotong University, [4]Nanjing University
zyllin@bjtu.edu.cn; zhengdazhou@smail.nju.edu.cn; tuzixini@gmail.com
tianrui@mail.nwpu.edu.cn; mayilun@mail.nwpu.edu.cn; gjy3035@gmail.com; xuelong_li@ieee.org
: https://github.com/MAIL-Tele-AI/WebUIBench

Abstract

With the rapid advancement of Generative AI technology, Multimodal Large Language Models(MLLMs) have the potential to act as AI software engineers capable of executing complex web application development. Considering that the model requires a confluence of multidimensional sub-capabilities to address the challenges of various development phases, constructing a multi-view evaluation framework is crucial for accurately guiding the enhancement of development efficiency. However, existing benchmarks usually fail to provide an assessment of sub-capabilities and focus solely on webpage generation outcomes. In this work, we draw inspiration from the principles of software engineering and further propose WebUIBench, a benchmark systematically designed to evaluate MLLMs in four key areas: *WebUI Perception*, *HTML Programming*, *WebUI-HTML Understanding*, and *WebUI-to-Code*. WebUIBench comprises 21K high-quality question-answer pairs derived from over 0.7K real-world websites. The extensive evaluation of 29 mainstream MLLMs uncovers the skill characteristics and various weakness that models encountered during the development process.

1 Introduction

The emergence of Large Language Models(LLMs) has rapidly reshaped the landscape of software engineering. AI code generation (Chen et al., 2024a; Shin and Nam, 2021; Dehaerne et al., 2022) evolves from assisting developers to independently completing the entire development lifecycle (*i.e.,* AI software engineer). Automatic website development is a challenging and widely discussed multimodal code generation scenario(Si et al., 2024; Yun et al., 2024; Beltramelli, 2018): Multimodal Large Language Models(MLLMs) are required to generate front-end code projects based on user-provided WebUI images (WebUI-to-Code).

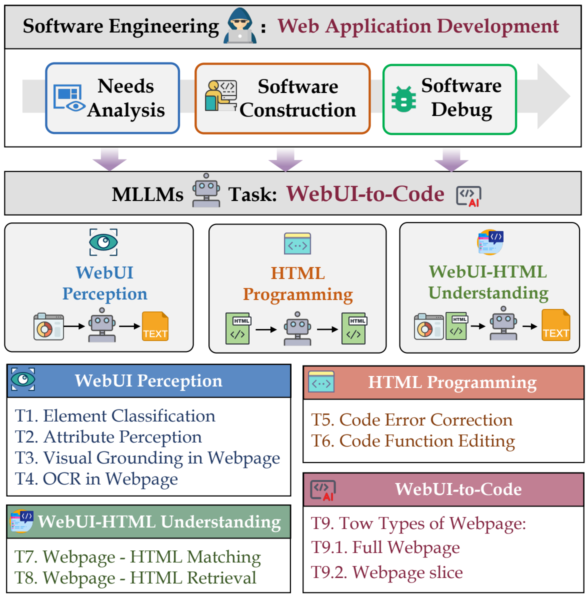


Figure 1: Evaluation taxonomy of WebUIBench.

Recent works(Si et al., 2024; Yun et al., 2024; Beltramelli, 2018) have evaluated MLLMs and reached a consensus that MLLMs struggle to generate complex websites, revealing a significant gap between solutions and practical applications. Therefore, it is essential to identify the challenges across various development stages and evaluate the corresponding sub-capabilities of models. However, current benchmarks(Guo et al., 2024; Si et al., 2024) typically focus on assessing the output quality of generated website (*e.g.,* webpage elements and layout) and mostly lack evaluation for sub-capabilities. To address this issue, (Yun et al., 2024) propose webpage understanding benchmark, but they are restricted to only one type of sub-capability and limited to the dataset quality annotated by LLMs. Inspired by the main activities of software engineering(Biolchini et al., 2005), we initially pro-

[*]Equal Contribution
[†]Work done during an internship at TeleAI
[‡]Corresponding Authors: X. Li and J. Gao

Findings of the Association for Computational Linguistics: ACL 2025, pages 15780–15797
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

pose a taxonomy for the capability evaluation of sub-capability as depicted in Figure 1.

Our core idea is to align the evaluation criteria of models' capabilities with the task requirements of software engineering. To this end, we propose the following sub-capability evaluation dimensions: **(i) WebUI Perception:** WebUI images serve as visual carriers of development requirements. The fundamental skill of the model is to accurately perceive the visual semantic information in the webpage, including both text and images. **(ii) HTML Programming:** During the software construction phase, the model's knowledge reservoir and programming skills in front-end code are essential for assisting or substituting developers for efficient development. and **(iii) WebUI-HTML Understanding:** Post-software development, code testing and adjustments are necessary to ensure requirement accuracy. This necessitates the model's ability to perform cross-modality reasoning between design images and code functionalities. Furthermore, we draw from current leading MLLMs evaluation benchmarks(Liu et al., 2025; Li et al., 2023; Yue et al., 2024; Li et al., 2024) to design multiple sub-tasks for each evaluation dimension, tailored to the characteristics of web data. In summary, our main contributions are three-fold:

- Construction of WebUIBench Dataset: The raw data is collected from 5 categories of frequently used real-world websites, including 719 complete webpage screenshots, source code and fine-grained information of all page elements. Based on this, WebUIBench consists of 2,488 webpage slices and 21,793 question-answer pairs across 9 sub-tasks.

- Evaluation of Mainstream MLLMs: The evaluation process is conducted in 29 mainstream MLLMs, including 22 open-source models such as the InternVL2.5 series and the Qwen2-VL series with parameters ranging from 2B to 78B, and 7 closed-source MLLMs, such as GPT-4o, Gemini-1.5 Pro, and Claude-3.5-Sonnet.

- Analysis of Challenges: The primary conclusion is that most MLLMs are not capable of performing the complete front-end software development process as effectively as humans. The observed positive correlation between sub-capabilities and WebUI-to-Code performance validates our evaluation approach. It also reveals that the primary challenge for current MLLMs is to enhance and balance sub-capabilities across different dimensions.

2 Taxonomy of Evaluation

Inspired by **Web Application Development**, our benchmark evaluates *WebUI-to-Code(**Task9**)* capability, and three essential sub-capabilities: *WebUI Perception*, *HTML Programming*, and *WebUI-HTML Understanding*. For WebUI-to-Code, we provide two types of webpage: full webpage and webpage slice. For sub-capability evaluation, we designed various sub-tasks as follows:

2.1 WebUI Perception

Inspiration: The WebUI design is a visual representation of **Needs Analysis**. WebUI Perception helps developers accurately grasp the requirements.

Task1. Element Classification. This task evaluates the model's capability to identify elements within webpage screenshots. The model must ascertain the presence of specific element types or combinations by fully understanding the screenshot.

Task2. Attribute Recognition. This task assesses the model's ability to discern detailed visual attributes of webpage elements, including text and background colors, font styles, and border styles.

Task3. Visual Grounding in Webpage. This task assess the model's capability to spatially locate elements on a webpage. We developed two levels of granularity for visual grounding tasks: *(i)* At a coarse granularity, after evenly dividing the web page into a grid, the model identifies the grid region number of the specified element; *(ii)* At a fine granularity, the model accurately returns the coordinates of the element's bounding box.

Task4. OCR in Webpage. This task tests the model's proficiency in extracting text from webpage screenshots. The model is required to detect and extract text content from a designated area framed by a red bounding box.

2.2 HTML Programming

Inspiration: The coding skills(e.g., HTML Programming) of developers ensure efficiency and stability throughout the **Software Construction** lifecycle.

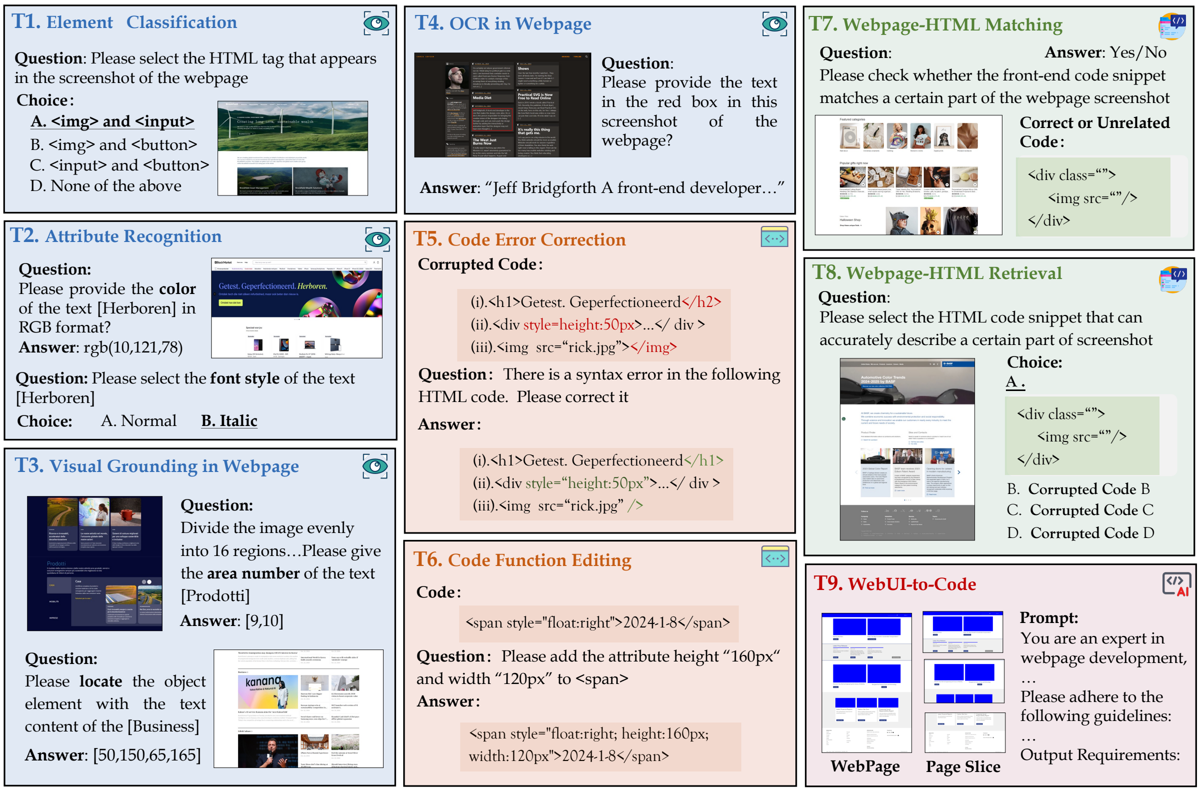


Figure 2: Task examples in the WebUI benchmark, from the WebUI Perception, HTML Programming, and WebUI-HTML Understanding and WebUI-to-Code task.

Task5. Code Error Correction. This task assesses the model's ability to correct syntax errors in front-end code. The model needs to identify errors in code snippets and return corrected versions.

Task6. Code Function Editing. This task evaluates the model's capability to implement static webpage functionalities through code. The model must edit and adjust code snippets according to the provided natural language instructions.

2.3 WebUI-HTML Understanding

Inspiration: Software Debugging aims at ensuring consistency between the HTML code and the WebUI, reducing functional deficiencies through cross-modality understanding.

Task7. Webpage-HTML Matching. The model determines whether the provided webpage screenshot and code snippet are correctly matched.

Task8. Webpage-HTML Retrieval. The model selects the appropriate code snippet that corresponds to the given webpage screenshot from a selection of multiple snippets.

WebUI-HTML Understanding tasks directly simulate a critical prerequisite in debugging: developers must identify and locate relevant code sections based on visual interface elements before fixing bugs. MLLMs need cross-modally understand the matching relationship between WebUI images and HTML code ("MLLMs cannot directly use F12").

3 Dataset

3.1 Raw Data Collection

WebUIBench consists of 5 categories of websites commonly visited by users: enterprise portals, background management systems, personal blogs, news sites, and e-commerce platforms. Firstly, we gather 1K websites (0.2K websites for each category) from the Internet. By using browser extension tools and manual collection, we collect the source HTML code and screenshot of these websites. Additionally, we extract detailed information of webpage elements, including tag categories, text content, CSS and spatial locations.

Quality Control. The incompletely or incorrectly loaded website are firstly reviewed and removed by human annotators . For excessively long pages, often found in news sites and e-commerce platforms categories due to repetitive elements, we develop a page simplification algorithm to refactor source HTML code. The algorithm can streamline

Statistic	Number
Total Website - HTML Code Samples	719
Total Question - Answer Samples	21793
Total Website Types	5
Total Task Types	9
Webpage Screenshots	
◇ Full page	719
◇ Slice page	2488
Screenshot Resolution	
◇ Maximal	1800×6802
◇ Minimal	1800×386
◇ Average	≈1800×1235
Tokens of Question Captions	
◇ Maximal	3582
◇ Minimal	42
◇ Average	≈287
Average slices per site	3.46
Average QA samples per slice	10.68

Table 1: Key statistics of WebUIBench.

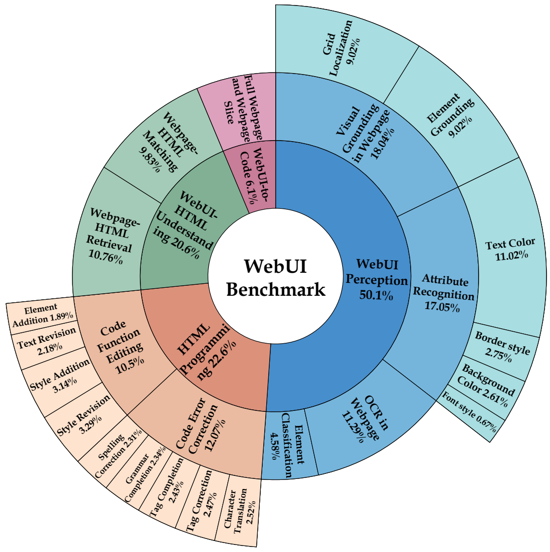


Figure 3: Question-Answer distribution of WebUIBench

webpage elements and shorten page length while ensuring the quality and diversity of elements. Detailed information about the algorithm is provided in the Appendix A.3.

3.2 Question and Answer Pairs Collection

Users usually browse websites by scrolling up and down, similar to viewing through a "sliding window." Inspired by this observation, webpage slice is designed as the fundamental image data for constructing dataset. We segment the screenshot of webpage into slices of varying sizes based on the page layout and browser window size, ensuring each slice is relatively independent and semantically complete. The detailed segmentation algorithm is introduced in the Appendix A.3. While collecting sliced screenshots, we also save the webpage element information within the slices and capture the corresponding code snippets to support the next step of the annotation process.

Automatic Labeling. We first design QA templates for each task mentioned in Section 2, including question caption, options and answers. To enhance the diversity and challenge of the evaluation data, the QA templates for each task can be transformed into various forms based on the designed strategies (as shown in Appendix A.4). By retrieving element information and code snippets, QA templates are automatically filled as the complete evaluation samples. To ensure the standards and quality of the dataset, automatic labeling is conducted in multiple batches. Each task within a batch undergoes sampling inspection, and the generation process will be optimized until all sampled data pass the inspection.

3.3 Dataset Statistics

For webpage data collection, our dataset consists of 719 full webpage and 2488 webpage slices from 5 categories, covering a variety of resolution modes. We open-source the screenshot (*.png* files), source HTML code (*.html* files), and element information (*.json* files) for these webpage. Based on this, WebUIBench includes 21,793 question-answer pairs, with an average of 10.68 question-answer pairs per webpage screenshot. Table 1 shows key statistics of dataset and Figure 3 shows the question-answer pairs distribution across different evaluation dimensions and tasks.

4 Metric

4.1 Automatic-metric Designs

Objective Question Scoring. For multiple-choice tasks, we employ accuracy as the scoring metric. For open-ended tasks such as OCR, the score is given based on the text similarity between the generated string and the ground-truth string(character-level Sørensen-Dice similarity). It is also important to note that in code correction and code editing tasks, the clarity of the questions and prompts ensures unambiguous answers. Therefore, we also score these tasks by calculating the string similarity between the generated HTML code and the standard answer. Additionally, for tasks involving the identification of element color attributes, we use the

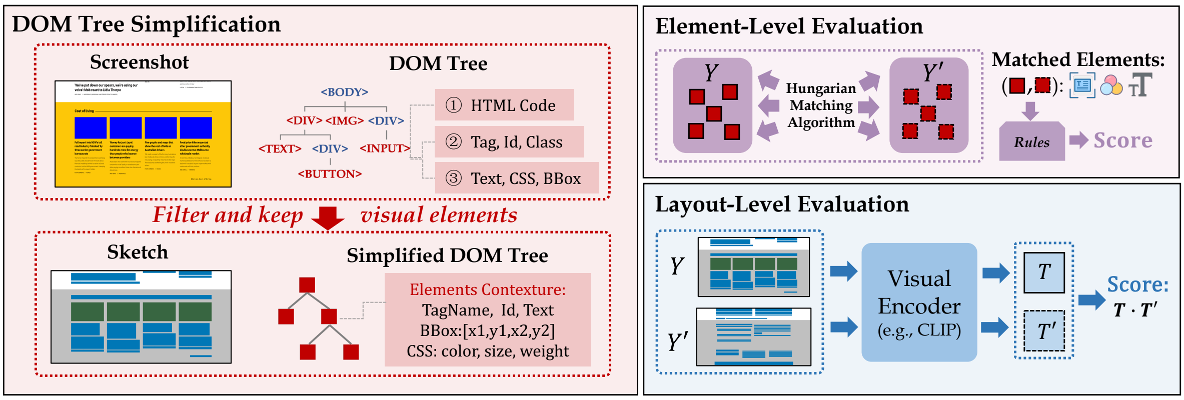


Figure 4: Schematic diagram of fine-grained WebUI-to-Code task evaluation process.

CIEDE2000 color difference formula for scoring, following (Luo et al., 2001).

WebUI-to-Code Task Scoring. Evaluation is approached from two levels of granularity:

Coarse-Grained Evaluation: This method involves calculating the visual similarity between the original webpage screenshot and the generated webpage screenshot to assess the overall visual quality. We utilize the visual pre-training backbone(e.g., CLIP(Radford et al., 2021)) to extract feature vectors and compute the cosine similarity as a measure of visual similarity.

Fine-Grained Evaluation: We separate the fine-grained evaluation into element-level and layout-level assessments as shown in Figure 4. The process includes: (i) Simplifying and restructuring the DOM tree of the webpage to preserve visual elements; (ii) Conducting evaluations separately at both the element and layout levels. The specific evaluation details are as follows:

- *DOM Tree Simplification:* We parse the original DOM tree to extract all elements related to visual presentation, including images, text, input fields, buttons, and areas with background colors. This results in a filtered set of webpage elements $R = \{r_1, r_2, \ldots, r_m\}$. Based on the data preparation outlined in Section 3.1, we gather corresponding information for each element in the set.

- *Element-Level Evaluation:* Given the sets of elements for the real and generated webpages, $R = \{r_1, r_2, \ldots, r_m\}$ and $G = \{g_1, g_2, \ldots, g_n\}$, we construct a cost matrix based on the similarity of the text content of the elements, as referenced in (Si et al., 2024). The Hungarian algorithm is then used to find the optimal matching of elements. We evaluate similarity metrics for successfully matched elements from the real and generated webpages, including**text content, font color and background color**.

- *Layout-Level Evaluation:* To maximize the decoupling of element-level and layout-level evaluations, we first remove content and style attributes of elements from the original webpage screenshot, preserving only size and spatial location information. Servel color blocks are used to distinguish elements of various tag categories, resulting in a sketch image of the webpage, as shown in Figure 4. We then use visual pre-training backbone to extract visual features from this sketch image, quantifying the webpage layout information. Finally, we evaluate the effectiveness of layout generation by calculating the cosine similarity between the visual features of the real and generated webpage layouts.

Table 2: We evaluate the correlation between the rankings produced by our evaluation framework and those given by human experts. In our ablation study, we conducted three groups of experiments: (i) removing element-level evaluation (w/o Element), (ii) removing layout-level evaluation (w/o Layout), and (iii) retaining only the visual features from CLIP (CLIP).

	Ours	w/o Element	w/o Layout	CLIP
Human-1	**0.83**	0.75	0.81	0.76
Human-2	**0.81**	0.74	0.76	0.75
Human-3	**0.85**	0.78	0.81	0.78
Average	**0.83**	0.76	0.77	0.77

Table 3: Evalutaion results of different MLLMs on the WebUIBench testset. Bold entries represent the best performance in each category and the underline entries represent the second-best performance. Task name: EC=Element Classification, AP=Attribute Perception, VG=Visual Grounding, CEC=Code Error Correcting, CFE=Code Function Editing, WHM=WebUI-HTML Matching, WHR=WebUI-HTML Retrieval, W2C=WebUI-to-Code.

Model	Size	EC	OCR	AP	VG	*Avg.*	CEC	CFE	*Avg.*	WHM	WHR	*Avg.*	***W2C***
Closed Source Model													
GPT-4o	-	83.3	79.1	79.8	44.4	**57.3**	91.8	90.4	91.1	65.7	41.9	53.8	**82.0**
GPT-4o-mini	-	42.4	72.9	70.8	38.9	45.0	92.0	90.7	**91.4**	50.1	46.4	48.2	74.9
Claude-3.5-Sonnet	-	78.9	77.3	80.7	42.9	55.9	88.6	86.9	87.8	73.7	43.6	**58.7**	80.2
Gemini-1.5-pro	-	63.9	76.8	70.3	26.1	47.4	87.7	84.8	86.2	65.0	47.0	56.0	80.0
Yi-Vision	-	59.2	37.7	68.9	28.5	38.9	84.3	81.5	82.9	46.3	48.9	47.6	77.0
GLM-4v	-	62.4	62.7	62.6	29.4	43.5	53.3	32.3	42.8	48.4	65.0	56.7	72.0
Step-1.5v-mini	-	62.1	49.4	57.1	17.6	46.5	86.6	84.9	85.7	44.8	53.8	49.3	67.9
Open Source Model													
Qwen2-VL	2B	28.2	49.6	54.4	39.9	34.4	16.2	15.6	15.9	24.7	29.8	27.2	62.1
InternVL2	2B	46.8	34.5	49.4	29.8	32.1	16.9	16.2	16.5	28.2	39.1	33.6	55.9
InternVL2.5	2B	43.8	45.5	41.6	37.8	33.7	61.9	56.5	59.2	31.0	58.7	44.8	55.9
Ovis1.6-Llama3.2	3B	64.8	45.4	51.4	38.6	40.0	35.6	50.8	43.2	18.5	37.4	27.9	**65.9**
InternVL2	4B	56.7	47.4	57.3	39.7	**40.2**	83.2	78.4	80.8	43.6	60.3	51.9	63.4
InternVL2.5	4B	56.1	50.6	55.8	36.3	39.8	92.1	86.9	**89.5**	71.7	58.6	**65.2**	63.6
Qwen2-VL	7B	78.3	76.1	67.3	16.1	**47.6**	41.9	69.9	55.9	55.9	36.2	46.1	65.8
InternVL2	8B	32.4	54.8	59.2	40.9	37.4	75.6	72.4	74.0	57.9	62.9	60.4	70.7
InternVL2.5	8B	26.0	47.5	60.2	42.4	35.3	83.8	84.1	**83.9**	75.4	62.9	**69.1**	**71.9**
MiniCPM-V-2.6	8B	49.9	54.7	54.5	23.3	36.5	66.2	62.3	64.2	21.6	34.7	28.1	70.4
Phi-3-vision	8B	62.9	16.4	57.3	40.5	35.5	58.1	44.5	51.3	25.7	36.7	31.2	56.0
Phi-3.5-vision	8B	12.2	3.8	53.7	29.4	19.8	67.9	45.2	56.6	25.4	35.4	30.4	53.5
Ovis1.6-Gemma2	9B	57.5	51.0	70.4	21.1	40.0	65.3	85.9	75.6	42.2	45.3	43.8	69.8
InternVL2	26B	31.1	65.0	65.4	35.3	39.4	80.8	82.3	81.6	52.7	55.0	53.8	67.6
InternVL2.5	26B	65.9	57.1	67.9	53.6	**49.1**	91.5	89.9	90.7	80.3	64.2	72.2	**75.3**
Ovsi1.6-Gemma2	27B	42.8	42.9	46.1	18.9	30.2	89.9	88.9	89.4	52.7	42.5	47.6	74.0
InternVL2.5	38B	36.1	55.9	49.5	40.5	36.5	92.4	90.5	**91.5**	89.7	65.9	**77.9**	74.6
InternVL2	40B	50.8	66.1	61.6	29.7	41.8	76.7	73.1	74.9	65.7	60.9	63.3	74.3
Qwen2-VL	72B	71.1	83.4	78.9	24.5	**51.6**	59.5	64.7	62.1	74.7	49.8	62.2	**77.3**
NVLM-D	72B	81.5	14.4	63.9	57.7	43.5	30.7	25.7	28.2	66.0	44.9	55.5	62.3
InternVL2	76B	42.3	71.3	66.0	39.3	43.8	83.5	86.1	84.8	61.9	57.8	59.9	74.9
InternVL2.5	78B	41.6	62.0	73.8	43.5	44.2	92.5	88.9	**90.7**	83.7	59.7	**71.7**	75.7

4.2 Human-centric Evaluations

We conduct an experiment to validate our evaluation framework: we randomly select 100 samples from the WebUI-to-Code task, and invite three front-end experts to rank the webpages generated by five models (GPT-4o, Qwen2-VL-72B, InternVL2.5-26B, Internvl2.5-8B, Ovis-Gemma2-3B). We calculate the correlation between human rankings and our evaluation framework rankings. Additionally, we conduct ablation experiments on different evaluation dimensions of our framework. The correlation results are as follows:

As shown in the Table 2, **the strong correlation (correlation>0.8) with human expert confirms the validity and effectiveness of our evaluation framework**. Ablation experiment results also confirmed that adding element and layout dimensions significantly improved the webpage similarity comparison.

5 Experiments

5.1 Models

We select both the latest and top-performing MLLMs for evaluation, including closed-source models: GPT-4o (Hurst et al., 2024), GPT-4o-mini, Gemini-1.5-Pro-002(Team et al., 2024), Claude-3.5-Sonnet(Anthropic), GLM-4V(GLM et al., 2024), Yi-Vision(Young et al., 2024) and Step-1.5v(ste); open-source models: InternVL2.5 series(Chen et al., 2024b), InternVL2 series(Chen et al., 2024b), Qwen2-VL series(Wang et al., 2024), Ovis-Gemma2 series(Lu et al., 2024), Phi-Vision series(Abdin et al., 2024), NVLM-D-72B(Dai et al., 2024) and MiniCPM-V-2.6(Yao et al., 2024). For open-source models, all parameter sizes within the same series are included in the evaluation process, ranging from the smallest model at 2 billion parameters to the largest at 78 billion parameters.

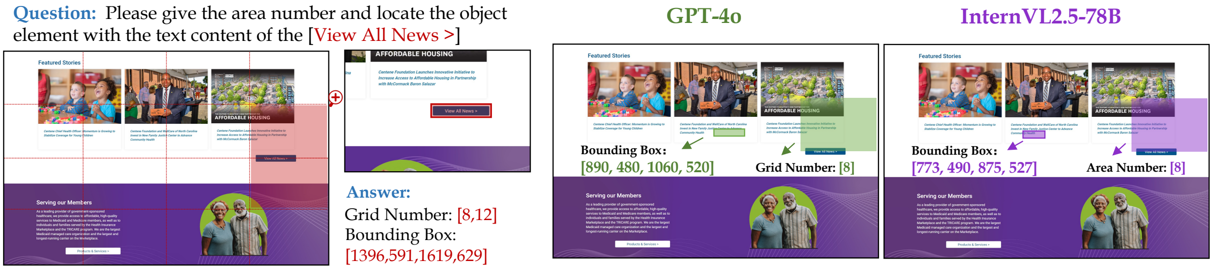


Figure 5: Visual grounding task examples (GPT-4o and InternVL2.5-78B).

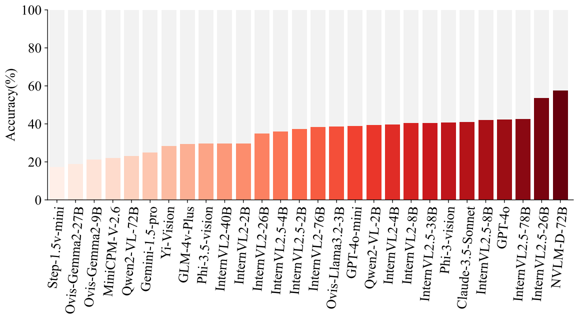


Figure 6: Results of grid number prediction (coarse-grained visual grounding)

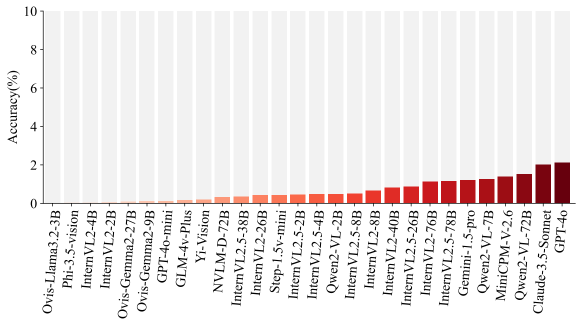


Figure 7: Results of bounding box prediction (fine-grained visual grounding)

5.2 Main Results

As illustrated in Table 3, we report the performance of all models across 9 tasks, with mean statistics calculated across various evaluation dimensions. To further provide a fine-grained evaluation and insights into the models' capabilities, we conduct a detailed analysis of both quantitative statistics and qualitative examples below.

5.2.1 Analysis of Capability Characteristics

The sub-capability assessment involves the results of *WebUI Perception*, *HTML Programming*, and *WebUI-HTML Understanding* (*i.e.,* from Task1 to Task8)

MLLMs exhibit personalized development capability advantages. For instance, *Qwen2-VL* series models generally performs better on WebUI Perception dimensions, indicating proficiency in addressing challenges from the visual modality. Conversely, *InternVL2.5* series demonstrates a more pronounced advantage in HTML programming tasks. Overall, *GPT-4o* exhibits a more comprehensive capability, yet remains weaker in WebUI-HTML Understanding tasks compared to the *Claude-3.5-sonnet*. This phenomenon indicates that our proposed evaluation taxonomy can uncover finer-grained differences between models, which is beneficial for leveraging and enhancing personalized capabilities.

Limitations of MLLMs in visual grounding task on Webpage. A common weakness exhibited by most MLLMs is their difficulty in performing visual grounding tasks, with predicting bounding boxes being more challenging than predicting grid numbers. Figure 5 shows a qualitative example where *GPT-4o* and *InternVL2.5-78B* can partially predict the area number of button locations correctly. However, the prediction of bounding boxes by both models completely deviated from the groundtruth. We attribute this phenomenon to the current MLLMs' lack of pixel-level understanding(Peng et al., 2024), necessitating more fine-grained annotation and training with webpage images.

MLLMs are poor at WebUI-HTML Understanding. Recalling results in Table 3, MLLMs (*e.g.,* parameters<40B) perform a random guessing (*i.e.,* score $\leq$ 50%) in Webpage-HTML Matching task, and performance in Webpage-HTML Retrieval tasks is also relatively lower compared to single-modality sub-tasks (*e.g.,* HTML Programming task). We hypothesize that this capability deficiencies stems from the increased information density in cross-modality reasoning scenarios and a lack of high-quality WebUI-HTML matching training data.

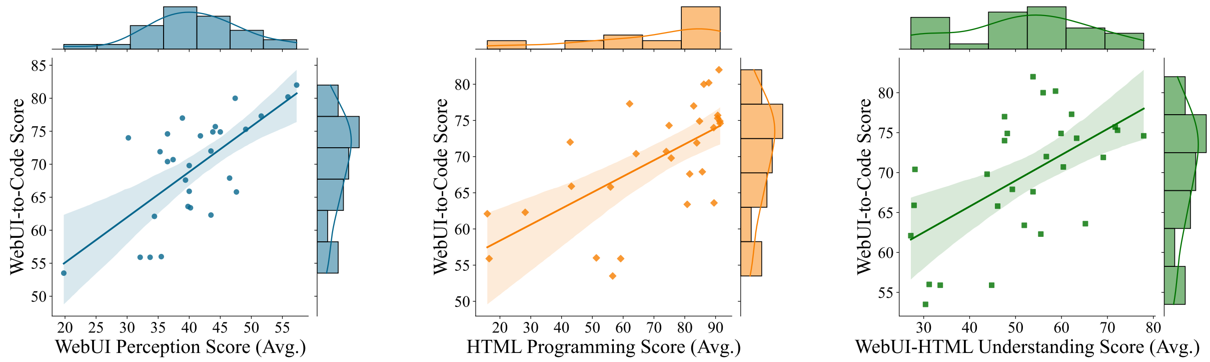


Figure 8: Positive correlations between WebUI-to-Code performance and sub-capability performance.

Table 4: The results of the instruction-following failure rate and code compilation success rate for the small MLLMs.

Model	Size	♯Samples	Accuracy
***Instruction-Following Evaluation* (↓)**			
InternVL2	4B	49	2.39%
InternVL2.5	4B	71	3.46%
InternVL2	2B	811	3.96%
Qwen2-VL	2B	1,085	5.17%
InternVL2.5	2B	1,393	6.81%
Ovis1.6-Llama3.2	3B	2,895	14.14%
***HTML Code Compilation* (↑)**			
Ovis1.6-Llama3.2	3B	823	62.37%
InternVL2	4B	504	38.23%
InternVL2.5	4B	247	18.74%
InternVL2.5	2B	239	18.13%
Qwen2-VL	2B	155	11.76%
InternVL2	2B	53	4.02%

5.2.2 Results for WebUI-to-Code Task

The more results of element-level and layout-level evaluation are reported in Appendix.

Positive correlations between WebUI-to-Code performance and personalized sub-capability. As shown in Figure 8, the positive correlation indicates that our evaluation taxonomy effectively reveals the model's WebUI-to-Code capabilities across different sub-capability dimensions. We can initially use this phenomenon to analyze and explain the performance gap. For example, the low competitiveness of *NVLM-D-72B* among 70B+ models may be due to its deficiency in HTML programming capabilities (*i.e,* CEC Score is 30.7% and CFE Score is 25.7%). It validates our idea of evaluating sub-capabilities according to software engineering principles.

Small MLLMs face increasing inference cost in HTML generation. HTML typically describes webpage content in long text form, posing challenges to the model's inference process. As shown in Table 4, although the outputs from the smaller models generally passed the instruction-following tests, the code content within the output often failed to compile successfully. We notice that small models tend to output repetitive content or incomplete code, affecting the proper closure of HTML tags. It hinders their ability to perform generation tasks well, despite having decent performance in some sub-capabilities.

Visualization for qualitative examples. As shown in Figure 9, we present the generation results of *GPT-4o* and *QwenVL2-72B* on complex webpages. By observing and comparing visual differences between generated webpage screenshots and WebUI images, we observe some interesting phenomena: (i)MLLMs demonstrates the ability to recognize vertical layouts of pages but struggles to identify and generate horizontal layouts. (ii) MLLMs can count elements effectively, yet performs poorly in generating the shapes and sizes of these elements. (iii) As the content of the webpage increases, these deficiencies become more pronounced.

5.3 Explorations of Future Benchmarking

Front-end Frameworks Evaluation. We select 100 WebUI images and used the Qwen2-VL-7B model to generate code, and setting whether to use the Tailwind framework (a lightweight front-end CSS framework) in the prompts. We present the compilation success rate and the webpage generation quality as follows:

Using the Tailwind framework improve the model's coding efficiency and success rate (47% to 92%), but unexpectedly, the quality of the generated webpages declined (0.718 to 0.697). Upon

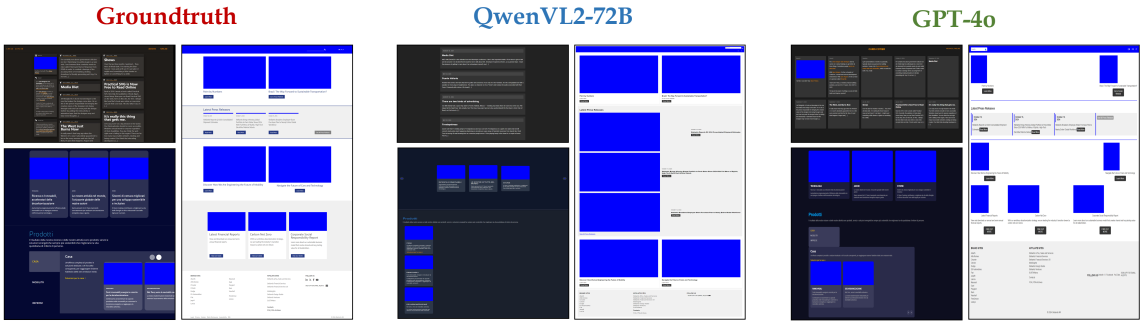


Figure 9: Examples of generated webpage by Qwen2-VL-72B and GPT4o on complex webpage and slices. Compared to implementing the styling of webpage elements, webpage layout remains a huge challenge for MLLMs.

reviewing the code, we find syntax errors related to the this framework and instances where Tailwind was mixed with raw HTML inconsistently. This suggests that coding with Tailwind remains challenging for MLLMs. Additionally, deficiencies in HTML Programming also impact webpage generation quality, aligning with our findings on NLVM-D-72B.

Dynamic Interaction Evaluation. A comprehensive evaluation of JavaScript functionality is a major challenge. To explore the model's capabilities in this area, we conducted a preliminary experiment to verify our hypothesis regarding their limitations: we designed 100 web development instructions requiring complex interactive functions, including form validation, event handling, and state management. These instructions covered various scenarios, from simple click events to complex data interactions.

We then asked the leading models, Claude-3.7-Sonnet and GPT-4o, to generate the required code. Through manual interaction and testing, we observed that the models' success rate in completing these interactive functions was extremely low: Claude scored 0.26, while GPT-4o scored 0.20. These low scores confirm that current models significantly under-perform in generating functional JavaScript code.

5.4 Discussions of Solutions

In the front-end software development process, engineers usually construct the page layout first and then fill in the element information based on the completed layout. However, it seems that MLLMs internally couple the generation processes of both. Although the generation process is a black box, the similar rankings of MLLMs regarding element-level and layout-level scores may support this observations. Intuitively, element-level and layout-level information represent two types of patterns: local fine-grained features and global spatial features. Therefore, a reasonable hypothesis is that generating element and layout features simultaneously may not fully leverage the model's capabilities.

The above and discussions may suggest a future solution: decoupling webpage lay out and element content generation into two steps, using methods like multimodal chain-of-thought(Wei et al., 2022) to incrementally generate webpages.

6 Conclusion

In this study, we introduce WebUIBench, a large-scale and comprehensive benchmark designed to evaluate the WebUI-to-Code capabilities of Multimodal Large Language Models (MLLMs). WebUIBench comprises over 21K question-answer pairs derived from more than 0.7K real-world websites, encompassing 9 distinct subtasks. We conducted extensive experiments on 7 state-of-the-art closed-source and 22 prominent open-source MLLMs. Our key findings highlight the models' deficiencies in webpage generation tasks across various dimensions, including cross-modality reasoning, element localization, and webpage layout generation. This benchmark provides critical insights and guidance for future research aimed at improving webpage generation performance.

Limitations

WebUIBench currently has the following limitations: (i) Imbalanced Data Distribution Across Subtasks: After manual and algorithmic filtering, some evaluation dimensions have only a few question-answer pairs remaining (*e.g.,* the evaluation data for font-style accounts for only 0.67%). In future work, we plan to address this by collecting new website data targeted at these missing categories. (ii) Lack of Mobile Webpage Evaluation Datasets: We have not yet constructed evaluation datasets for mobile platforms (*e.g.,* smartphones). Considering that mobile web development is a prevalent task in software engineering, we plan to supplement our current data to include mobile evaluation datasets and results. (iii) Absence of Page Functionality Interaction Evaluation: In practical development, both static page and dynamic interaction functionalities need to be considered. In future work, it will also be important to evaluate MLLMs in generating interactive functionality code (*e.g.,* JavaScript).

References

step-1.5v-mini.

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Anthropic. The claude 3 model family: Opus, sonnet, haiku.

Tony Beltramelli. 2018. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems*, pages 1–6.

Jorge Biolchini, Paula Gomes Mian, Ana Candida Cruz Natali, and Guilherme Horta Travassos. 2005. Systematic review in software engineering. *System engineering and computer science department COPPE/UFRJ, Technical Report ES*, 679(05):45.

Liguo Chen, Qi Guo, Hongrui Jia, Zhengran Zeng, Xin Wang, Yijiang Xu, Jian Wu, Yidong Wang, Qing Gao, Jindong Wang, et al. 2024a. A survey on evaluating large language models in code generation tasks. *arXiv preprint arXiv:2408.16498*.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. 2024b. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*.

Wenliang Dai, Nayeon Lee, Boxin Wang, Zhuolin Yang, Zihan Liu, Jon Barker, Tuomas Rintamaki, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2024. Nvlm: Open frontier-class multimodal llms. *arXiv preprint*.

Enrique Dehaerne, Bappaditya Dey, Sandip Halder, Stefan De Gendt, and Wannes Meert. 2022. Code generation using machine learning: A systematic review. *Ieee Access*, 10:82434–82455.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.

Hongcheng Guo, Wei Zhang, Junhao Chen, Yaonan Gu, Jian Yang, Junjia Du, Binyuan Hui, Tianyu Liu, Jianxin Ma, Chang Zhou, et al. 2024. Iw-bench: Evaluating large multimodal models for converting image-to-web. *arXiv preprint arXiv:2409.18980*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. 2024. Unlocking the conversion of web screenshots into html code with the websight dataset. *arXiv preprint arXiv:2403.09029*.

Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2023. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning*, pages 18893–18912. PMLR.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. 2023. Seed-bench: Benchmarking multimodal llms with generative comprehension. *arXiv preprint arXiv:2307.16125*.

Lin Li, Guikun Chen, Hanrong Shi, Jun Xiao, and Long Chen. 2024. A survey on multimodal benchmarks: In the era of large ai models. *arXiv preprint arXiv:2409.18142*.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. 2025. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pages 216–233. Springer.

Shiyin Lu, Yang Li, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Han-Jia Ye. 2024. Ovis: Structural embedding alignment for multimodal large language model. *arXiv:2405.20797*.

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

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

Wujian Peng, Lingchen Meng, Yitong Chen, Yiweng Xie, Yang Liu, Tao Gui, Hang Xu, Xipeng Qiu, Zuxuan Wu, and Yu-Gang Jiang. 2024. Inst-it: Boosting multimodal instance understanding via explicit visual prompt instruction tuning. *arXiv preprint arXiv:2412.03565*.

Tony Beltramelli Pix2code. Generating code from a graphical user interface screenshot [electronic resource]. *arXiv preprint arXiv:1705.07962*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Jiho Shin and Jaechang Nam. 2021. A survey of automatic code generation from natural language. *Journal of Information Processing Systems*, 17(3):537–555.

Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2code: How far are we from automating front-end engineering? *arXiv preprint arXiv:2403.03163*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *Preprint*, arXiv:2409.12191.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, et al. 2024. Minicpm-v: A gpt-4v level mllm on your phone. *arXiv preprint arXiv:2408.01800*.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. 2024. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9556–9567.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, et al. 2024. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *arXiv preprint arXiv:2406.20098*.

A Appendix

A.1 Related Work

Before the advent of Large Language Models(LLMs), a series of works(Beltramelli, 2018; Lee et al., 2023; Nguyen and Csallner, 2015; Pix2code) have already begun exploring how to convert webpage screenshots into HTML code. With the development of Multimodel Large Language Models(MLLMs), this field has seen the emergence of several works aimed at evaluating and addressing webpage code generation issues: WebSight(Laurençon et al., 2024) introduced a large-scale synthetic dataset to train models in the code generation domain, but did not provide an evaluation dataset or methodology. Design2Code(Si et al., 2024) was the first to systematically evaluate both open-source and closed-source MLLMs using real webpage data. Web2Code(Yun et al., 2024) proposed an evaluation task for webpage understanding, expanding previous evaluation frameworks and introducing a high-quality code instruction dataset. IWBench(Guo et al., 2024) presented an evaluation method focused on webpage layout and improved generation algorithms using CoT.

Table 5: Comparison of WebUIBench with previous works

Benchmark	Source	♯Size	Sub-cap.
Websight	Synthetic	823K	✗
Pixel2Code	Synthetic	1.7K	✗
Web2Code	Synthetic	884.7k	✓
Design2Code	Real-World	484	✗
IWBench	Real-World	1.2K	✗
WebUIBench(Ours)	Real-World	21K	✓

A.2 Examples for Different Tasks

As shown in Figure 10 to 16 , we provide specific examples for each evaluation task under the three major capability dimensions to illustrate the sample dataset.

A.3 Data Collection

Webpage Simplification Algorithm: In the collected dataset of raw web pages, some pages are overly long, particularly those from news or e-commerce sites. These pages surpass the maximum input length of current multimodal large models, creating challenges for subsequent evaluations. Additionally, these pages contain numerous redundant and duplicate elements. To address this, we developed a web page simplification algorithm that analyzes the DOM tree structure to identify and remove redundant nodes, thereby facilitating more effective evaluations. The specific process of the web page simplification algorithm is shown in Alg 19.

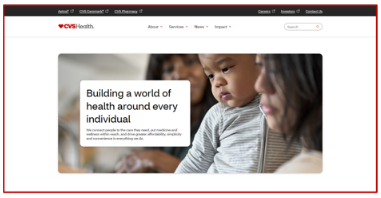
Webpage Segmentation Algorithm: The web page segmentation algorithm is designed to overcome the input length limitations of large models. By dividing web pages into multiple slices, we can incrementally process and analyze web content, ensuring each slice is manageable by the model. This approach not only enhances the model's processing efficiency but also improves evaluation accuracy. The process of the web page segmentation algorithm is shown in Alg 20.

A.4 Automatic Labeling Strategy

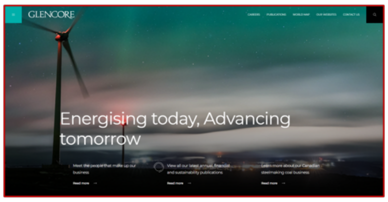
Task 1: Element Classification We first use the front-end code to determine whether the following three types of tags and their combinations exist on the web page: **<input />, <button />, <img />**. We then construct them into correct options, such as **B. <img /> and <input />**. At the same time, we construct error interference options by permuting and combining elements that do not exist on the page; for example: **A. <img /> and <button />, C. <input /> and <button />, D. None of the above.**

Task 2: Attribute Perception To prevent interference in the testing process from elements with identical text content, we initially selected web page elements with unique text. Using the element IDs, we extracted the following four types of style content from the CSS file to create question-answer pairs.

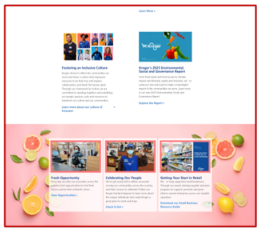
- **background-color** We construct questions and correct answers based on the RGB color format requirements, such as **rgb(19,25,36)**.
- **color** We construct questions and correct answers based on the RGB color format requirements, such as **rgb(19,25,36)**.
- **font-style** We construct the following multiple-choice questions: **A. Italic, B. Oblique.**
- **border-radius** We construct the following multiple-choice questions: **A. Rounded corners, B. Square corners.**



Question: Please select the HTML tag that appears in the screenshot of the web page.
A. <input>,<img>
B. <input>,<button>,<img>
C. <button>,<img>
D. <input>,<button>
Answer: B



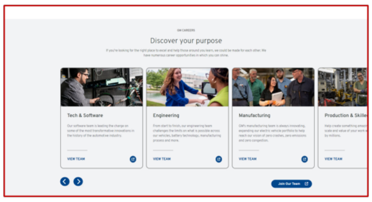
Question: Please select the HTML tag that appears in the screenshot of the web page.
A. <input>,<img>
B. <button>,<input>
C. <img>,<button>
D. None of the above
Answer: C



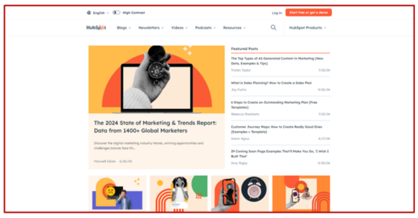
Question: Please select the HTML tag that appears in the screenshot of the web page.
A. <button>
B. <input>
C. <img>
D. None of the above
Answer: C

Figure 10: Samples of Element Classification.

Question: Please select the style of the block with the text [三星随享] in the screenshot of the web page.
A. Rounded corners
B. Square corners
Answer: A

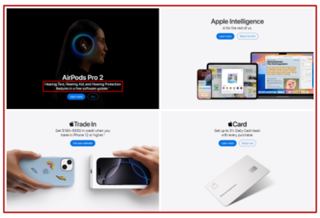


Question: Please provide the **background color** of the text [**Join Our Team**] in the webpage screenshot in RGB format.
Answer: rgb (0, 71, 140)



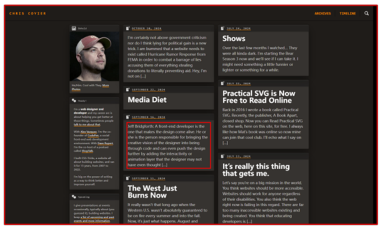
Question: Please provide the **background color** of the text [**Start free or get a demo**] in the webpage screenshot in RGB format.
Answer: rgb (255, 92, 53)

Figure 11: Samples of Attribute Perception.



Question: Please provide the text in the red box in this screenshot of the webpage?
Answer: Hearing Test, Hearing Aid, and Hearing Protection features in a free software update.

Question: Please provide the text in the red box in this screenshot of the webpage?
Answer: 24 Apr 6.00 am UTC Declaration

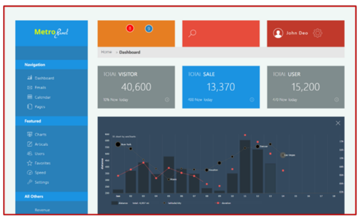


Question: Please provide the text in the red box in this screenshot of the webpage?

Answer: Jeff Bridgforth: A front-end developer is the one that makes the design come alive. He or she is the person responsible for bringing the creative vision of the designer into being through code and can even push the design further by adding the interactivity or animation layer that the designer may not have even thought [...]

Figure 12: Samples of OCR in the Webpage.

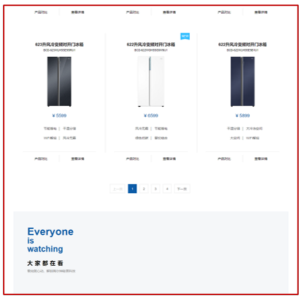
Question: Please locate the object element with the text content of the [**JA Leaders Condemn Trump's Comparison Between Treatment of Jan. 6 Rioters and WWII Incarcerees**] in the screenshot of the web page, return the bounding box [left, top, right, bottom] in an array as json format.
Answer: [390,792,596,875]



Question: Please locate the object element with the text content of the [**Emails**] in the screenshot of the web page, return the bounding box [left, top, right, bottom] in an array as json format.
Answer: [156,401,213,425]

Question: Please locate the object element with the text content of the [**960GB – 1TB**] in the screenshot of the web page, return the bounding box [left, top, right, bottom] in an array as json format.
Answer: [32,822,132,846]

Figure 13: Samples of Visual Grounding(fine granularity).



Question: Divide the image evenly into 16 regions, forming a 4x4 grid. The numbering rules are as follows:
- Each row is numbered from left to right.
- The rows are numbered from top to bottom.

The specific numbering order is:

[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 16]

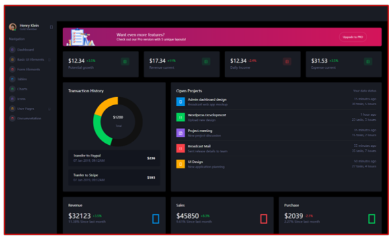
This way, each region is numbered starting from the top-left corner, increasing row by row.
Please give the area number where the text content of the [BCD-623WLHSSE5SRU1] element in the screenshot of the web page is located. If the text string spans multiple areas, please give the numbers of all these areas. The region numbers are stored in an array and returned as json format.
Answer: [1,2]

Question: Divide the image evenly into 16 regions, forming a 4x4 grid. The numbering rules are as follows:
- Each row is numbered from left to right.
- The rows are numbered from top to bottom.

The specific numbering order is:

[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 16]

This way, each region is numbered starting from the top-left corner, increasing row by row.
Please give the area number where the text content of the [My GDC '24 Talk: The Playdate Story] element in the screenshot of the web page is located. If the text string spans multiple areas, please give the numbers of all these areas. The region numbers are stored in an array and returned as json format.
Answer: [1,2,3,4]



Question: Divide the image evenly into 16 regions, forming a 4x4 grid. The numbering rules are as follows:
- Each row is numbered from left to right.
- The rows are numbered from top to bottom.

The specific numbering order is:

[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 16]

This way, each region is numbered starting from the top-left corner, increasing row by row.
Please give the area number where the text content of the [$31.53] element in the screenshot of the web page is located. If the text string spans multiple areas, please give the numbers of all these areas. The region numbers are stored in an array and returned as json format.
Answer: [8]

Figure 14: Samples of Visual Grounding(coarse granularity).

Question: There is a syntax error in the following HTML code. Please correct it and return only the corrected HTML code in json format.
HTML Code:

```
<button aria-label="Play/Pause" class=homeHero--playBtn icon-pause-btn id=c5f63018-4497-4419-a2dc-28f820260bd0>
    <span class="visibility-hidden" id="932dd9bb-ef9b-4ffd-857c-477a444042f0">Play/Pause</span>
</button>
```

Answer:

```
<button aria-label="Play/Pause" class="homeHero--playBtn icon-pause-btn" id="c5f63018-4497-4419-a2dc-28f820260bd0">
   <span class="visibility-hidden" id="932dd9bb-ef9b-4ffd-857c-477a444042f0">Play/Pause</span>
</button>
```

Question: There is a syntax error in the following HTML code. Please correct it and return only the corrected HTML code in json format.
HTML Code:

```
<img data-savepage-currentsrc="default-url" data-savepage-src=" default-url " id="aeed1dc7-452c-4a77-817e-e87d3cec763d" src="default-url"/></img>
<p id="70f1b345-51cf-4f0d-abfd-8e4c9e026e12">October 7, 2024
```

Answer:

```
<img data-savepage-currentsrc="default-url" data-savepage-src=" default-url" id="aeed1dc7-452c-4a77-817e-e87d3cec763d" src="default-url"/>
<p id="70f1b345-51cf-4f0d-abfd-8e4c9e026e12">October 7, 2024</p>
```

Question: There is a syntax error in the following HTML code. Please correct it and return only the corrected HTML code in json format.
HTML Code:

```
<span class="button-primary__label" id="29709cd0-8baa-414f-a5ee-53b40fbc03c2"> > Contact us</span>
<span class="button-primary__label" id="29709cd0-8baa-414f-a5ee-53b40fbc03c2"> Contact us <
</span>
```

Answer:

```
<span class="button-primary__label" id="29709cd0-8baa-414f-a5ee-53b40fbc03c2"> Contact us</span>
```

Figure 15: Samples of Code Error Correction.

Question: Please edit the code according to the given html code and instructions, and only return the edited HTML code in json format.
Instruction: Please change the id attribute of element to\"kzcrqbnoov\
HTML Code:

```
<p class="section-name c-blue bold" id="9251da8f-9475-4e0c-b13d-363ffb9b2390">What we do</p>
```

Answer:

```
<p class="section-name c-blue bold" id=" kzcrqbnoov ">What we do</p>
```

Question: Please edit the code according to the given html code and instructions, and only return the edited HTML code in json format.
Instruction: Change the text content of the element's first text child node to "读取和写入速度分别最高可达 500MB/秒和 450MB/秒“
HTML Code:

```
<li id=\"e52bbdc4-ec84-4247-8dc9-e2f1e3e0d639\"> This is a text placeholder</li>
```

Answer:

```
<li id=\"e52bbdc4-ec84-4247-8dc9-e2f1e3e0d639\">读取和写入速度分别最高可达 500MB/秒和 450MB/秒</li>
```

Question: Please edit the code according to the given html code and instructions, and only return the edited HTML code in json format.
Instruction:
Please continue to add the attribute height at the end of style"189px"
Please continue to add the attribute width at the end of style"133px";
Please continue to add the attribute background-color to the end of style as "rgb(160, 200, 231)“
HTML Code:

```
<div class="logo" id="9650be8c-45c2-479b-bcda-899b3129a634" style="display:block"></div>
```

Answer:

```
<div class="logo" id="9650be8c-45c2-479b-bcda-899b3129a634" style="display:block; height: 189px; width: 133px; background-color: rgb(160, 200, 231);"></div>
```

Figure 16: Samples of Code Function Editing.

Task 3: Visual Grounding in the Webpage To prevent interference in the testing process from elements with identical text content, we first selected web page elements with unique text content. Next, we retrieved the spatial position information of these elements based on their IDs: $[x1, y1, x2, y2]$. Using this spatial information, we constructed two types of visual grounding tasks. For the grid localization task, we divided the webpage screenshot into a 4x4 grid and automatically calculated the grid numbers occupied by the elements based on their coordinates.

Task4: OCR in the Webpage To prevent interference in the testing process from elements with identical text content, we first selected web page elements with unique text. Next, we sorted all text by length and chose short, medium, and long texts to construct question-answer pairs. Finally, using the elements' coordinate information, we drew red borders on the corresponding webpage screenshots to guide the model in performing OCR tasks.

Task 5: Code Error Correction To construct correction samples, we designed various code corruption methods based on real web element code to ensure diversity in error types, comprehensively covering common front-end code errors. The specific methods are as follows:

- **Missing Closing Tag.** *Description:* Missing a closing tag, resulting in incomplete web elements. *Method:* Delete the closing tag of real web elements to generate error samples.

- **Incorrect Character Escaping.** *Description:* Certain characters need escaping; otherwise, they interfere with HTML parsing. *Method:* Insert random special symbols such as &, <, > into element text to create escaping errors.

- **Tag Spelling Error.** *Description:* Tag name spelling error, such as writing <p> as <p1>. *Method:* Randomly modify the end tag by adding erroneous characters.

- **Attribute Syntax Error.** *Description:* Attribute values not enclosed in quotes, causing syntax errors. *Method:* Remove quotes from page element attribute values.

- **Attribute Spelling Error.** *Description:* Attribute name spelling error, such as writing class as clbss, possibly causing style loss. *Method:* Replace some attribute names of elements with misspelled versions.

- **Erroneous Addition of Tags.** *Description:* Adding a closing tag to tags that do not require one (*e.g.,* <img />, <input />). *Method:* Add erroneous "closing tags" to these tag types.

For each real web element, one of the above methods is randomly selected with equal probability to corrupt the code, producing erroneous element code, with the original web code serving as ground truth. This approach generates a large and diverse set of correction samples, aiding in a comprehensive evaluation of MLLM's ability to correct various front-end code errors.

Task 6: Code Function Editing. Similarly, based on real web elements, we construct code editing instructions and edited web element code. To cover various common code editing scenarios, we have also designed multiple editing methods. The specific editing methods are described as follows:

- **Modify Element Attributes.** *(i)* Randomly select an existing attribute of an element to modify, such as class, id, href, etc, and*(ii)* Generate random values for the selected attribute and replace it.

- **Modify Element Text.** *Uniformly modify the text content of elements (if any) to a placeholder:* Replace the first text child node of the element (if any) with "This is a text placeholder."

- **Add or Modify Element Style.** *Modify or add style attributes of elements, such as height and background-color:* If the attribute exists, modify its value; otherwise, append a new style definition in the style attribute.

- **Add or Delete Child Nodes.** *(i)* Randomly delete a child node (if any) or *(ii)* Insert a new child node into the element, randomly set the tag and attributes of the child node, and set its text content to "this is a new node."

For each real web element, one of the above editing methods is randomly selected with equal probability to edit the original code, resulting in edited code (as ground truth) and specific editing methods (as editing instructions for MLLM). This approach generates a large and diverse set of code

Table 6: Evalutaion results of WebUI-to-Code at element and layout level. Dimensions name: CCSR=Code Compile Success Rate, TS=Text Similarity, CS=Color Similarity, BCS=Background Color Similarity, CGE=Coarse-Grained Evaluation

Model	Size	CCSR	TS	CS	BCS	Layout	CGE
Closed Source Model							
GPT-4o	-	95.8	73.6	72.1	81.1	89.2	81.7
GPT-4o-mini	-	99.0	59.8	53.3	66.7	86.4	76.8
Cluad-3.5-Sonnet	-	85.2	72.9	70.4	79.6	88.7	81.1
Gemini-1.5-pro	-	96.7	73.7	67.9	80.5	87.8	78.8
Yi-Vision	-	96.1	61.4	61.1	72.8	86.5	78.4
GLM-4v	-	77.5	58.9	55.6	68.4	85.5	74.8
Step-1.5v-mini	-	62.3	51.4	59.1	73.5	85.5	75.4
Open Source Model							
Qwen2-VL	2B	11.8	51.8	54.2	79.7	83.8	68.0
InternVL2	2B	4.0	48.5	29.7	60.3	80.9	67.0
InternVL2.5	2B	18.1	47.9	39.3	55.9	84.0	62.7
Ovis1.6-Llama3.2	3B	62.4	57.5	45.7	67.5	83.2	68.6
InternVL2	4B	38.2	56.6	47.1	64.9	83.2	68.6
InternVL2.5	4B	18.7	50.8	46.6	64.5	84.7	74.2
Qwen2-VL	7B	47.3	54.1	49.2	64.3	84.6	71.8
InternVL2	8B	81.5	62.8	51.3	70.7	84.1	71.2
InternVL2.5	8B	95.3	58.8	49.7	63.5	84.3	73.4
MiniCPM-V-2.6	8B	90.9	58.0	46.6	64.7	85.0	71.9
Phi-3-vision	8B	60.5	23.7	29.8	41.8	82.1	64.5
Phi-3.5-vision	8B	53.6	21.7	27.3	38.6	81.1	62.5
Ovis1.6-Gemma2	9B	60.3	58.5	52.5	69.2	85.8	74.4
InternVL2	26B	75.0	57.7	48.3	64.0	84.3	69.4
InternVL2.5	26B	91.6	58.9	63.5	69.1	84.9	76.9
Ovsi1.6-Gemma2	27B	80.8	62.7	58.4	72.4	85.7	76.1
InternVL2.5	38B	86.6	59.2	57.9	72.7	87.3	76.5
InternVL2	40B	84.6	67.5	57.7	76.7	85.9	74.1
Qwen2-VL	72B	83.7	69.5	61.5	74.8	88.2	79.2
NVLM-D	72B	23.7	52.8	45.8	70.1	83.9	69.4
InternVL2	76B	94.9	63.3	55.7	71.7	86.6	75.3
InternVL2.5	78B	85.4	65.6	59.8	74.5	86.9	77.0

editing samples, aiding in a comprehensive evaluation of MLLM's ability to modify web element code based on diverse editing instructions.

Task 7: Webpage-HTML Matching We first matched the corresponding HTML code snippets using the ID set of all elements in the webpage slices. Next, we randomly chose, with equal probability, whether to alter the correct code snippets. For the modification, we selected the code of an element from another slice of the same webpage to replace one element's code in the correct snippet.

Task 8: Webpage-HTML Retrieval We first matched the corresponding HTML code using the ID set of all elements in the webpage slices and used it as the correct option. For the incorrect options, we randomly selected elements and their corresponding code snippets from other slices on the same website, replacing parts of the correct option's code at different levels. We chose to replace 1, 2, or 3 elements to construct three incorrect options.

A.5 More Results

We show more detailed experimental results of all models at the element level and layout level in Table 6. At the element level, we supplement the evaluation results at different fine-grained dimensions, including text content, font color and background color.

Figure 17 shows the generation effects of more models on webpage slices and full webpages. We selected the top-2 models of open source and closed source models for display. It can be seen that as the complexity of web page content increases, the generation effect of the model gradually deteriorates.

Algorithm 1: Simplify DOM Tree

Input: Original DOM tree T;similarity threshold δ
Output: Simplified DOM tree T'
begin
  Initialize T' as a copy of T;
  Group child nodes by TagName;
  foreach *group of child nodes* **do**
    foreach *pair of nodes* (n_1, n_2) *in the group* **do**
      Split ClassName of n_1 and n_2 into list $\mathcal{S}_1$ and $\mathcal{S}_2$;
      Calculate Dice similarity: Dice $\leftarrow \frac{2\times|\mathcal{S}_1 \cap \mathcal{S}_2|}{|\mathcal{S}_1|+|\mathcal{S}_2|}$
      if *Dice similarity* δ **then**
        Group n_1 and n_2 together;
      end
    end
    if *group size* ≤ 5 ***or*** *all nodes have similar heights* **then**
      Retain all nodes in the group;
    end
    else
      Sort nodes by bottom height in ascending order and retain the top 50%;
    end
  end
end
return *Simplified DOM tree* T'

Algorithm 2: Webpage Slices Generation

Input: Webpage Screenshots, slice height H_s; minimum slice height H_{min}; webpage height H_{page} ; Element coordination set $\mathcal{S} \leftarrow \{[x_1^1, y_1^1, x_2^1, y_2^1], ...[x_1^n, y_1^n, x_2^n, y_2^n]\}$
Output: List of webpage slices
begin
  Initialize slice top boundary $H_t \leftarrow 0$ and slice bottom boundary $H_b \leftarrow H_s$;
  while $H_b < H_{page}$ **do**
    Filter the elements which $y_2 > H_b$ and $y_1 < H_b$;
    $y_{max} \leftarrow$ calculate the maximum value of set y_2;
    if $H_{page} - y_{max} \leq H_{min}$ **then**
      $y_{max} \leftarrow H_{page}$;
    end
    Retrieve element IDs and save webpage slice between H_t and y_{max};
    $H_t \leftarrow y_{max}$;
    $H_b \leftarrow y_{max} + H_s$;
    if $H_b > H_{page}$ **then**
      Retrieve element IDs and save webpage slice between H_t and H_{page};
      break;
    end
  end
  return *All webpage slices, element IDs*;
end

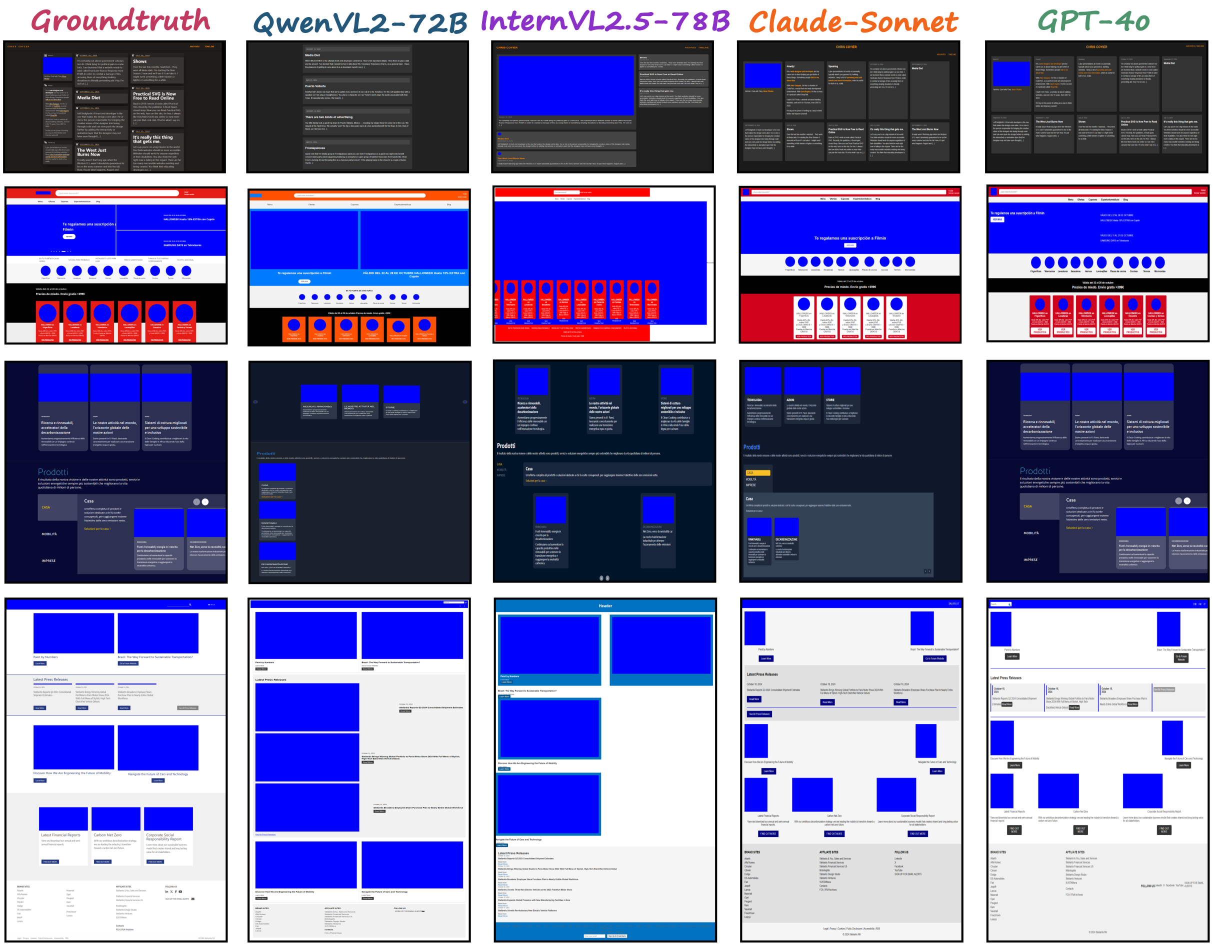


Figure 17: Comparison of webpage generation effects by different models on complex webpages and slices.