Interaction2Code: Benchmarking MLLM-based Interactive Webpage Code Generation from Interactive Prototyping

Jingyu Xiao[1], **Yuxuan Wan**[1], **Yintong Huo**[2*], **Zixin Wang**[1], **Xinyi Xu**[1],
Wenxuan Wang[1], **Zhiyao Xu**[3], **Yuhang Wang**[4], **Michael R. Lyu**[1]
[1]The Chinese University of Hong Kong, [2]Singapore Management University
[3] Tsinghua Shenzhen International Graduate School, [4] Southwest University
`whalexiao99@gmail.com` `yxwan9@cse.cuhk.edu.hk` `ythuo@smu.edu.sg`
`{zixinwang, xinyixu, wxwang}@link.cuhk.edu.hk` `lyu@cse.cuhk.edu.hk`
Project Page: https://webpai.github.io/Interaction2Code/

Abstract

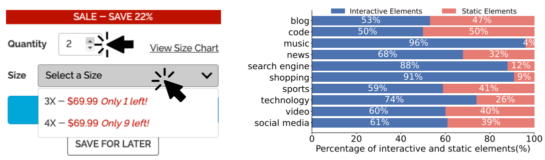
Multimodal Large Language Models (MLLMs) have demonstrated remarkable performance on the design-to-code task, i.e., generating UI code from UI mock-ups. However, existing benchmarks only contain static web pages for evaluation and ignore the dynamic interaction, limiting the practicality, usability and user engagement of the generated webpages.

To bridge these gaps, we present the first systematic investigation of MLLMs in generating interactive webpages. Specifically, we formulate the **Interaction-to-Code** task and establish the **Interaction2Code** benchmark, encompassing **127** unique webpages and **374** distinct interactions across **15** webpage types and **31** interaction categories. Through comprehensive experiments utilizing state-of-the-art (SOTA) MLLMs, evaluated via both automatic metrics and human assessments, we identify four critical limitations of MLLM on Interaction-to-Code task: (1) inadequate generation of interaction compared with full page, (2) prone to ten types of failure, (3) poor performance on visually subtle interactions, and (4) insufficient undestanding on interaction when limited to single-modality visual descriptions. To address these limitations, we propose four enhancement strategies: **Interactive Element Highlighting**, **Failure-aware Prompting (FAP)**, **Visual Saliency Enhancement**, and **Visual-Textual Descriptions Combination**, all aiming at improving MLLMs' performance on the Interaction-to-Code task. The Interaction2Code benchmark and code are available in `https://github.com/WebPAI/Interaction2Code`.

1 Introduction

Converting webpage design into functional UI code is a critical step for building websites, which can be labor-intensive and time-consuming. MLLMs have shown remarkable performance on visually rich code generation tasks (Yang et al., 2024), which create new opportunities for the Design-to-Code task, i.e., generating code from UI designs to replicate web page elements, layout, text, and colors. For example, Design2Code (Si et al., 2024) proposes three types of prompts to stimulate MLLMs' web content understanding and self-refined capabilities for GUI code generation. DCGen (Wan et al., 2024) develops a divide-and-conquer-based approach to prompt MLLMs to generate webpage elements by division and assembly stages.

However, existing research (Si et al., 2024; Yun et al., 2024; Gui et al., 2024) only focuses on the static appearance of a webpage (e.g., color, layouts), ignoring the dynamic interactive properties and functionality of such elements, like size selection list, and quantity adjustment button shown in Figure. 1(a). Additionally, we observe that such interactive elements account for a large proportion of the webpage in real-world software practices. We randomly select 10 real-world webpages with different topics to analyze the ratio of interactive elements, the results in Figure. 1(b) indicate that interactive elements take up more than 50% cases.



(a) Interactive elements. (b) Interactive and static ratio.

Figure 1: Interaction example and interactive elements ratio of different types of webpages.

Static webpages limit user interaction with web elements, hindering access to new content (such as browsing images via carousel buttons) or impeding task completion (like selecting clothing sizes from drop-down menus), thereby impairing over-

* Yintong Huo is the corresponding author.

all user experience. Therefore, we argue that **a benchmark for interactive webpages is essential to enhance the practicality, usability, and user engagement of studies on auto-generated GUI code.** To this end, we provide the first systematic analysis of MLLMs' capability in generating interactive webpages. Our contributions are summarized as follows:

- **Task formulation**. We are the *first* to formulate the **Interaction-to-Code** task and present a systematic study on the code generation capabilities of MLLMs for dynamic interaction of webpages.
- **Benchmark**. We build the *first* **real-world** webpage interaction datasets **Interaction2Code** containing **127** webpages and **374** interactions, spanning **15** webpage topics and **31** interaction types. We also provide **failure annotations** for the MLLM-generated webpages.
- **Key Findings.** Our in-depth analysis reveals four main limitations: (1) MLLMs struggle to generate interactive part compared with full static webpage generation; (2) MLLMs are prone to make 10 types of failures; (3) MLLMs perform poorly on interactions that are not visually obvious; (4) Single visual modality description is not enough for MLLMs to understand the interaction.
- **Improvements.** We propose four methods to improve the performance of MLLMs on the Interaction-to-Code task. (1) **Interactive element highlighting**, i.e., applying visual markers for interactive elements can improve MLLMs' performance by forcing MLLMs to focus on the Interaction. (2) **Failure-aware prompting (FAP)** can make MLLMs avoid potential failures by incorporating the failure example into prompts. (3) **Visual saliency enhancement (VSE)** allows the model to better perceive the interaction area by image cropping, thereby improving the performance of interaction generation. (4) **Visual and textual description combination** can makes MLLMs understand the interaction better.

2 Background

2.1 Related Work

Some benchmarks and methods are proposed to evaluate and improve the ability of MLLM's UI code generation. Websight (Laurençon et al., 2024) synthesize a dataset consisting of 2 million pairs of

Benchmark	Real World	Failure Annotation	Interactive
WebSight (Laurençon et al., 2024)	✗	✗	✗
Vision2UI (Gui et al., 2024)	✓	✗	✗
Design2Code (Si et al., 2024)	✓	✗	✗
Interaction2Code (Ours)	✓	✓	✓

Table 1: Comparisons between Interaction2Code and existing UI2Code benchmarks.

HTML codes and their corresponding screenshots for fine-tuning MLLMs on UI2Code tasks. Vision2UI (Gui et al., 2024) extracts from real-world scenarios, augmented with comprehensive layout information, tailored for finetuning MLLMs in UI code generation. Design2Code (Si et al., 2024) generates UI code through text-augmented and self-revision prompting. DCGen (Wan et al., 2024) proposes a divide-and-conquer-based approach to generate the UI code. DeclarUI (Zhou et al., 2024) uses the element segmentation method to accurately generate elements and page transition graphs to prompt MLLMs to generate app UI with jump logic. **Although the above works achieve decent performance on the UI2Code task, none of them consider the generation of interactive webpages.**

2.2 Problem Definition

UI-Mockup (UI-Mockup, 2025) is a visual representation of a user interface, essentially a static image showing the look and feel of a webpage. Figure 2 shows that the UI2Code task takes the static UI-Mockup S as input and generates a static webpage. **Interactive Prototyping** (Interactive, 2025) is a functional model of that design, allowing users to simulate interactions and navigate through the interface to test usability and functionality before full development. An interactive behavior is represented as an interactive prototype $IP = \{S_o, S_I\}$, where S_o is the UI-Mockup of original webpage and S_I is the UI-Mockup after the interaction I.

Interaction2Code task takes the interactive prototyping IP as input and generates an interactive webpage as shown in Figure 3.

3 The Interaction2Code Benchmark

3.1 Dataset Collection

We follow these steps for constructing benchmark that represent a variety of real-world use cases (i.e., diverse webpages and interactions).

Webpage Selection. We collect webpages from CommonCrawl (C4 validation set (Raffel et al.,

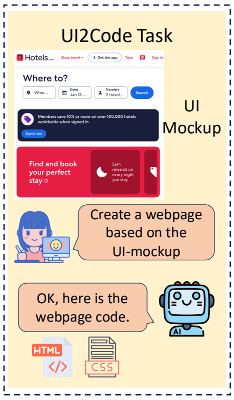


Figure 2: UI2Code.

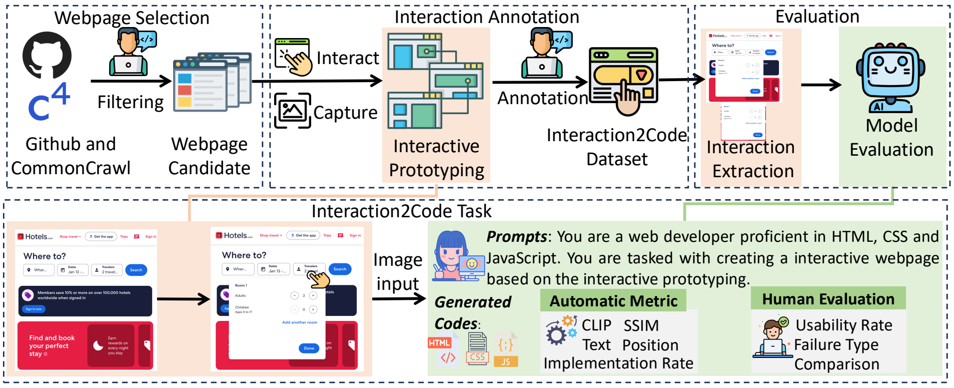
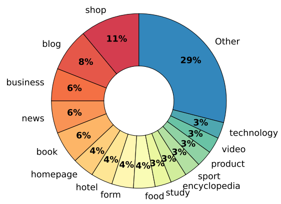


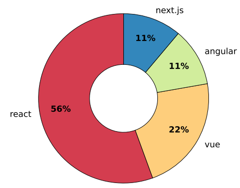
Figure 3: The construction of Interaction2Code benchmark.

2020)) and github. (1) CommonCrawl. Following the Design2Code (Si et al., 2024), we automatically filter out webpages that are too long or too simple (only contain images or texts) and run deduplication. We then choose 15 common web topics and randomly sample 1k web pages related with these topics. Finnaly, we employ four PhD students majoring in computer science, each with experience in front-end development. Each student is assigned to select approximately 25 webpages, thus obtaining 100 webpages from C4. The selection guideline is shown in Appendix E.4.1. The selection criteria are as follows: 1) complexity: each web page must contain at least one meaningful interactions; 2) diversity: the selection process aims to include a wide range of webpages with different interaction types. Most of the websites in C4 are traditional and do not use UI frameworks, so we also collect webpaes from github website projects built with UI frameworks. (2) Github projects. We search for "open-production-web-projects" and "awesome-opensource-apps" on GitHub to get a summary list of web projects, then we identify 27 popular projects with deployed links and higher star counts. These projects represent various real-world website uses, ranging from commercial product frontend websites to blogs, with 13k average star counts. Their popularity have proven their usefulness and quality. Ultimately, we compile a dataset consisting of 127 webpages.

Interaction Annotation. (1) Interactive Prototyping Construction. In real-world webpages, there are many trivial interactions, like underlining texts when hovering. To preserve meaningful interactions and ensure the complexity and diversity of interactions, the four PhD students are employed to interact with webpages and select complex and meaningful interactions to capture the pre- and post-



(a) Topic.



(b) Framework.

Figure 4: Topic and framework distribution.

interaction screenshots to build interactive prototyping (the guideline is shown in Appendix E.4.2). Finally, 1-10 important and functional interactions are retained on one webpage and we get 374 interactions. (2) Annotation. The four PhD students manually annotate the topics of the web pages, the development framework, and the types of interactions for benchmark diversity analysis.

3.2 Data Statistics and Diversity

Topic and Framework Distribution. Figure 4(a) shows that our benchmark covers a diverse range of web topics with more than 15 types, including business, shop, technology, entertainment, and so on. Figure 4(b) shows that the benchmark includes mainstream front-end open source frameworks such as react, next.js, vue, and angular.

Interaction Type Distribution. We manually annotate the type of interactions based on their element tag and the visual effect perspective. Tag categories come from HTML tags such as button, image, and link. Buttons, input boxes, and links are the most frequent types as shown in Table 2 and play a great role in human-website interaction. Visual categories involve changes in color, size, position, text, etc. Note that one interaction may belong to multiple tag categories and visual cate-

Tag Categories				Visual Categories	
Element	Number	Element	Number	Type	Number
button	235	summary	15	text	162
input	52	form	13	new component	161
span	37	detail	12	color	85
link	36	video	11	position	45
select	35	area	9	switch	41
textarea	35	output	9	new page	37
option	31	datalist	8	new window	34
iframe	28	dialog	6	size	20
text	24	audio	5	-	-
progress	22	template	3	-	-
image	21	table	1	-	-
label	16	-	-	-	-

Table 2: Tag and visual categories distribution.

gories. Table 2 demonstrates that Interaction2Code benchmark has a rich set of interaction types, including 23 tag categories and 8 visual categories.

3.3 Evaluation

Automated Interaction When generating web pages, we prompt MLLMs to encode the id of the interactive elements (for example, id="interact1"). During evaluation, we apply selenium webdriver (Selenium, 2025) to locate the interactive elements by id and automatically interact with the generated webpage and take screenshots to construct the interactive prototyping.

Interaction Extraction. After obtaining the interactive prototyping (i.e., screenshots before and after the interaction), we automatically extract the interactive part for evaluation. For interactions that preserve webpage dimensions, we identify affected areas through pixel-wise subtraction. For interactions that alter webpage dimensions (e.g., showing details), we employ Git diff tool (Git, 2025) to locate modified rows and columns, with their intersections marking the affected regions. Detailed extraction algorithm is provided in Appendix C.

Full Page Metrics. We measure the quality of generated webpages from the following perspectives: **(1) Visual Similarity**. We use CLIP score (Radford et al., 2021) to measure the visual similarity. **(2) Structure Similarity**. SSIM (Wang et al., 2004) (Structural Similarity Index Measure) score is applied to calculate the structure similarity. **(3) Text Similarity**. We apply OCR tools to recognize the text in the webpages, and then use the BLEU score (Papineni et al., 2002) to measure the text similarity between the two webpages.

Interaction Part Metrics. We also evaluate the interactive parts of webpages from the perspective of the position and function of the interaction.**(1) Position Similarity**. The position similarity between original interaction I_o and generated interaction I_g is defined as $P(I_o, I_g) = 1 - max(|x_o - x_g|, |y_o - y_g|)$, where (x_o, y_o) and (x_g, y_g) are normalized coordinates (in $[0, 1]$) of the interactive area center. **(2) Implement Rate (IR)** measures the ratio of interactions successfully implemented by MLLM. An interaction is considered implemented if detectable by webdriver, and unimplemented otherwise. Let $N(\cdot)$ denote the quantity, we can calculate the **IR** as $IR = \frac{N(implemented)}{N(implemented)+N(unimplemented)}$. **(3) Usability Rate (UR)**. Human annotators are asked to interact with the generated webpage and judge the usability. We can calculate as $UR = \frac{N(usable)}{N(usable)+N(unusable)}$. We also employ human annotators to conduct pairwise comparison and failure type analysis in Section 5.2 and Section 5.3.

4 Study Setup

4.1 Evaluation Models

To understand the MLLMs' performance on Interaction-to-Code task and identify the gap between open-source and closed-source models, we conduct experiments on three popular commercial models: Gemini-1.5-flash (Google, 2024), GPT-4o-20240806 (OpenAI, 2024a) and Claude-3.5-Sonnet-20240620 (Anthropic, 2024). **Interaction-to-Code task takes multiple images as input, and many open source MLLMs do not support that (e.g., llava (Liu et al., 2024), llama-3.2-vision (Meta, 2024))**, so we select Qwen2.5-vl-instruct (3B, 7B, 72B) (Qwen, 2025) for assessment. The detailed parameters are in Appendix E.2.

4.2 Prompt Design

We provide the reference webpage interactive prototyping consisting of two screenshots, along with the instruction to generate the HTML, CSS and JavaScript code (full prompt in Appendix E.1)

5 Experiments

5.1 Model Performance

The model performance is shown in Table 3. We can make the following observations MLLMs under *direct prompting*: (1) GPT-4o and Claude-3.5-Sonnet have higher performance than other models according to the average value. (2) Among the open source models, Qwen2.5-vl-72B has the

Model	Prompt	Full Page			Interaction Part				
		CLIP	SSIM	Text	CLIP	SSIM	Text	Position	IR
Qwen2.5-vl-3B-instruct	Direct	**0.3220**	**0.1932**	**0.1510**	**0.2100**	**0.1531**	0.0415	**0.2090**	**0.3449**
	CoT	0.2031	0.1085	0.0800	0.1219	0.0894	0.0352	0.1212	0.1979
	Mark	0.2752	0.1503	0.1200	0.1706	0.1188	**0.0514**	0.1706	0.2647
	Average	0.2668	0.1507	0.1170	0.1675	0.1204	0.0427	0.1669	0.2692
Qwen2.5-vl-7B-instruct	Direct	0.4169	0.2886	0.2519	0.3230	0.2177	0.0952	0.2529	0.4786
	CoT	0.3895	0.2529	0.2207	0.2806	0.1981	0.0744	0.2259	0.4305
	Mark	**0.4586**	**0.3282**	**0.2703**	**0.3541**	**0.2468**	**0.1348**	**0.2798**	**0.5267**
	Average	0.4217	0.2899	0.2477	0.3192	0.2209	0.1015	0.2529	0.4786
Qwen2.5-vl-72B-instruct	Direct	0.6430	0.4234	0.4197	0.4624	0.3207	0.2450	0.3950	0.6524
	CoT	0.6335	**0.4785**	0.4585	**0.5090**	**0.3692**	0.2376	0.4385	**0.7380**
	Mark	**0.6954**	0.4569	**0.4586**	0.4992	0.3621	**0.2995**	**0.4541**	0.7112
	Average	0.6573	0.4529	0.4456	0.4902	0.3507	0.2607	0.4292	0.7005
Gemini-1.5-flash	Direct	0.5967	0.4526	0.4749	0.4737	0.3616	0.2809	0.4320	0.6738
	CoT	0.6166	0.4810	0.4775	0.5093	0.3854	0.3217	0.4511	0.7112
	Mark	**0.6321**	**0.4946**	**0.4878**	**0.5194**	**0.3898**	**0.3454**	**0.4612**	**0.7326**
	Average	0.6151	0.4761	0.4801	0.5008	0.3789	0.3160	0.4481	0.7059
GPT-4o	Direct	0.7114	0.5277	**0.5147**	0.5605	0.4149	0.3590	0.4888	0.7754
	CoT	0.6905	0.4962	0.4761	0.5234	0.4013	0.3663	0.4668	0.7273
	Mark	**0.7160**	**0.5539**	0.5112	**0.5955**	**0.4488**	**0.4474**	**0.5225**	**0.8128**
	Average	0.7059	0.5259	0.5007	0.5598	0.4217	0.3909	0.4927	0.7718
Claude-3.5-Sonnet	Direct	0.7172	**0.5318**	**0.6003**	0.5674	0.4209	0.3833	0.5123	0.7914
	CoT	0.6961	0.5110	0.5603	0.5606	0.4005	0.3662	0.5085	0.7727
	Mark	**0.7258**	0.5299	0.5899	**0.5944**	**0.4282**	**0.4319**	**0.5149**	**0.7968**
	Average	0.7130	0.5242	0.5835	0.5742	0.4165	0.3938	0.5119	0.7870

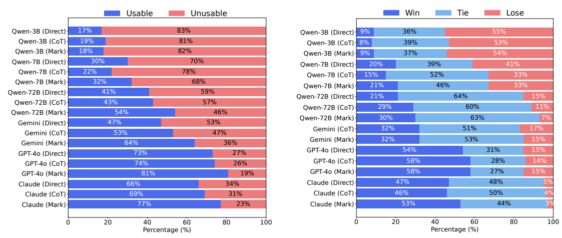
Table 3: Performance of different MLLMs under different prompts on Interaction-to-Code task. **Bold values** indicate the optimal performance, and underlined values indicate the second-best performance. The red value is the highest value among the averages.

best performance and is comparable to the commercial model Gemini-1.5-flash. As the model size increases, the performance gradually improves. (3) **The performance of MLLMs in the interactive part is lower than that of the full page (Limitation 1).** Limitation 1 is caused by the fact that the MLLMs do not pay attention to the interaction part enough, which motivates us to propose a solution to emphasize the interaction part.

Improvement 1: Interactive element highlighting. To improve the performance of generated interaction, we further propose *Chain-of-Thought (CoT) and Mark prompts* to force models to focus on the interaction.

For CoT prompt (Wei et al., 2022), we design three thinking steps: analyze the interaction effects, locate the interactive elements, and implement the interaction. For Mark prompt, we use red bounding boxes to highlight the interaction area, prompting MLLMs to focus on the interaction.

Both CoT and Mark prompts enhance model performance compared to direct prompt, the



(a) Usability evaluation. (b) Pairwise comparision.

Figure 5: A higher usable rate indicates better functionality and a higher win rate indicates better quality.

Mark prompt demonstrates superior performance compared to the CoT prompt. Gemini-1.5-flash's metrics (CLIP, SSIM, text, position, IR) of the interaction part improve from direct prompting scores (0.4737, 0.3616, 0.2809, 0.4302, 0.6738) to (0.5093, 0.3854, 0.3217, 0.4511, 0.7112) with CoT, and further to (0.5194, 0.3898, 0.3454, 0.4612, 0.7326) with Mark prompting.

5.2 Human Evaluation

Functionality Evaluation. Four PhD students with three years of front-end development experience are employed to evaluate the functionality (i.e., usability) of generated interaction. If the interactive function is consistent with ground truth, it is regarded as usable, otherwise unusable (the guideline details are shown in Appendix E.4.3). The usability rate results are shown in Figure 5(a).

Pairwise Model Comparison. We ask five human annotators to rank a pair of generated interactions (one from the baseline, the other from the tested methods) to decide which one implements the reference interaction function better. We use Gemini-1.5-flash with direct prompt as the baseline and collect the other 17 methods' Win/Tie/Lose rates against this baseline. Each pair will count as Win (Lose) only when Win (Lose) receives the majority vote (≥ 3). All other cases are considered Tie. The guideline is shown in Appendix E.4.4. The results are shown in Figure 5(b); a higher win rate and lower loss rate suggest better quality as judged by human annotators.

Results. (1) Our human evaluation reveals that GPT-4o and Claude-3.5-Sonnet consistently demonstrates superior performance compared to other baseline models. (2) Both CoT and Mark prompting strategies can enhance model performance beyond direct prompting, showing higher win rates and usability rates across most models (except Qwen-vl-7B-instruct's CoT prompt). (3) Mark prompting yields the most significant improvements in usability, with Claude-3.5-Sonnet showing 11% and 8% increases compared to Direct and CoT prompts, respectively (Figure. 5(a)). (4) These human evaluation results align with Section 5.1, validating that our automatic evaluation metrics are reasonable.

5.3 Failure Type Analysis

Four PhD students with three years of front-end development experience are employed to analyze the difference between the generated and the original interactions, then summarize the failure types and evaluate their influence from content, function and user experience. We first randomly select 25% interactions for analysis and then discuss, revise, and refine the failure type until everyone reaches a consensus. During annotating new data, if encountering a new failure type, annotators will communicate and update failure type in time to guide subsequent annotations (the guideline is shown in Appendix E.4.5). Table 4 shows that **MLLMs are prone to make 10 types of failure (Limitation 2)**. the failure definition is in the Appendix F. Ten representative failure examples are shown in Figure. 11 and Figure. 12, where the first row shows the reference interaction, and the second row shows the generated interaction by MLLMs.

Failure reason analysis. Failures (a), (c), (e), and (f) stem from MLLMs' limitations in element localization. Failures (d) and (g) are caused by MLLMs' misidentification of element types. Failures (b), (h), (i), and (j) arise from MLLMs' misunderstanding of interaction.

Base on the failure distribution in Figure 6, we find that, **the main failure modes include "No interaction", "Partial implementation", "Interactive element missing", and "Wrong function".**

Besides, the most serious failures are "Interactive element missing", "Wrong function", "No interaction" and "Effect on wrong element". The severity of the failures depends on the usability rate (UR), with higher UR meaning lower severity and lower UR meaning higher severity. As illustrated in Table 4, failure (a), (b) and (j) exhibit UR lower than 10%, rendering the generated interactions completely ineffective.

Improvement 2: Failure-aware Prompt (FAP). Based on failure types, we propose FAP to stimulate the self-criticism ability of MLLM, thereby avoiding problems that may occur in the Interaction-to-Code task.

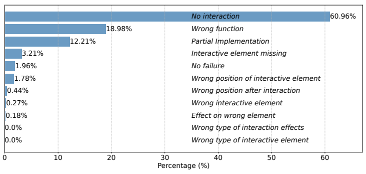
FAP incorporates the failure example into the prompt and tell MLLMs to avoid these types of failures (full prompt in Appendix E.1). We use $\frac{2}{3}$ of the dataset to annotate failure types and $\frac{1}{3}$ of the dataset to test. Table 5 shows the results of the FAP methods, we can find that **Failure-aware Prompt can improve the performance of the Interaction-to-Code task on all models.** The full results are shown in Appendix G.2.

5.4 The Impact of Interaction Visual Saliency

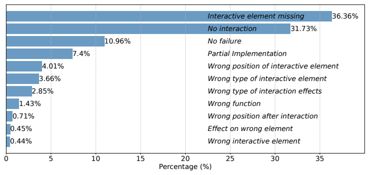
The visual perception limitations of MLLMs affect their performance on visual understanding tasks, especially when facing small low-resolution objects (Zhang et al., 2024). We examine the impact of interaction area ratio (i.e., visual saliency) on generation outcomes. Let I denote interaction, S_I denote the screenshot of the webpage after interaction I, we define the visual saliency $VS(I) = \frac{area(I)}{area(S_I)}$, where $area()$ calculates the size (in pixels) of a component. A higher VS score

Failure Object	Failure Type	Content	Function	User Experience	Usability Rate
Interactive element	(a) Interactive element missing	🔴	🔴	🔴	0%
	(b) No interaction	🟡	🔴	🔴	6.93%
	(c) Wrong interactive element	🟡	🟡	🔴	92.31%
	(d) Wrong type of interactive element	🟡	🟡	🔴	96.82%
	(e) Wrong position of interactive element	🟡	🟡	🔴	98.41%
Interaction effects	(f) Wrong position after interaction	🟡	🟡	🔴	96.17%
	(g) Wrong type of interaction effects	🟡	🟡	🔴	57.14%
	(h) Effect on wrong element	🟡	🟡	🔴	44.44%
	(i) Partial Implementation	🟡	🟡	🔴	89.20%
	(j) Wrong function	🔴	🔴	🔴	0%

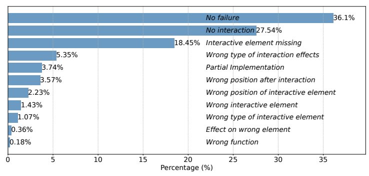
Table 4: Failure types and their influences, where 🔴 represents full impact and 🟡 represents partial impact.



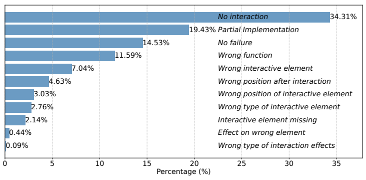
(a) Qwen2.5-vl-3B.



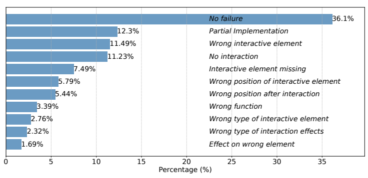
(b) Qwen2.5-vl-7B.



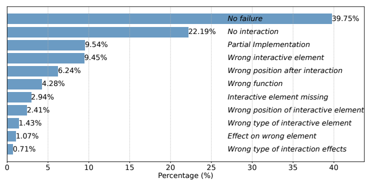
(c) Qwen2.5-vl-72B.



(d) Gemini-1.5-flash.



(e) GPT-4o.



(f) Claude-3.5-Sonnet.

Figure 6: Failure distribution of MLLMs.

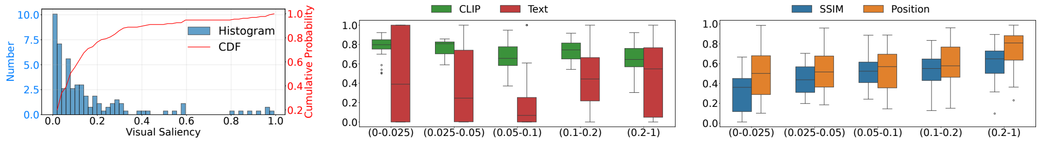
Model	Method	CLIP	SSIM	Text	Position
Gemini 1.5-flash	Direct	0.5403	0.4494	0.3602	0.5802
	FAP	0.5886	0.4584	0.4394	0.6032
	Δ	↑ 0.0483	↑ 0.0090	↑ 0.0792	↑ 0.0230
GPT 4o	Direct	0.5700	0.4891	0.3652	0.5803
	FAP	0.6072	0.5405	0.4580	0.6452
	Δ	↑ 0.0372	↑ 0.0514	↑ 0.0928	↑ 0.0649
Claude 3.5 Sonnet	Direct	0.4582	0.3771	0.3086	0.4927
	FAP	0.4921	0.4035	0.3822	0.5154
	Δ	↑ 0.0339	↑ 0.0264	↑ 0.0736	↑ 0.0227
Qwen2.5 vl-72B instruct	Direct	0.4741	0.3612	0.3275	0.5022
	FAP	0.5144	0.3750	0.3286	0.5376
	Δ	↑ 0.0403	↑ 0.0138	↑ 0.0011	↑ 0.0354

Table 5: Comparison between direct prompt and FAP.

indicates a larger area influenced by the interaction and, consequently, a higher visual saliency.

We first calculate the visual saliency for all interactions and plot the distribution, as shown in Figure 7(a). We then divide the samples into five groups based on the distribution results, keeping the number of samples in each group roughly balanced. The VS ranges for the five groups are as follows: [0, 0.025), [0.025, 0.05), [0.05, 0.1], [0.1, 0.2), [0.2, 1). Figure 7 shows the box plot distribution of metrics for Gemini-1.5 across these five groups, we can find that **the group with lower visual saliency has lower SSIM and position similarity (Limitation 3).** Although the clip and text similarity fluctuates among different groups, as shown in Figure 7(b), Figure 7(c) shows that the SSIM and position similarity significantly increases as the visual saliency increases. As shown in Figure 7(c), the group [0.2, 1) shows the highest metrics, while the group [0, 0.025) shows the lowest metrics. This demonstrates that MLLMs are more likely to capture structural and positional features for samples with high visual saliency.

We then randomly sample 10 webpages from failure cases and crop the screenshots to increase the visual saliency of the interactions in the webpages (for example, if the webpage is cropped to $\frac{1}{2}$ of the original, the visual saliency of the interaction will be doubled). Figure 8 shows the relationship between the magnification factor and the metrics



(a) Visual saliency distribution. (b) CLIP and Text. (c) SSIM and Position.

Figure 7: Visual saliency and interaction part metrics distribution of different groups of Gemini-1.5-flash.

of generation results. We observe that: when the magnification factor is set to 1, all evaluation metrics yield values of 0, indicating the unsuccessful interaction generation. Upon increasing VS by 1.2 times, the model is able to reproduce interactions, but with relatively low metric scores. As the magnification factor increases from 1.2 to 3, we observe substantial improvements in performance metrics: the CLIP and SSIM similarities approach 0.8, while text and position similarities reach approximately 0.6. This suggests that models effectively overcome the original failure cases.

Improvement 3: Visual Saliency Enhancement (VSE). By cropping the image to increase the proportion of the interactive part, VSE makes the model to better perceive the interaction area.

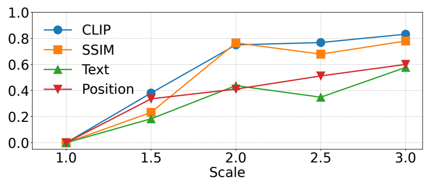


Figure 8: Metrics under different magnification.

5.5 The Impact of Different Modalities

Prompt	Modality	CLIP	SSIM	Text	Position
Direct	V	0.3737	0.1793	0.2539	0.3951
	T	0.4174	0.4067	0.2316	0.4293
	V+T	**0.6735**	**0.5612**	**0.3919**	**0.7157**
CoT	V	0.3871	0.3101	0.2433	0.4461
	T	0.5579	0.1828	0.3045	0.5465
	V+T	**0.6440**	**0.4800**	**0.4287**	**0.7080**
Mark	V	0.5015	**0.4520**	0.3389	0.5025
	T	0.4613	0.4454	0.2805	0.4810
	V+T	**0.6923**	0.4336	**0.4248**	**0.7469**

Table 6: Performance of GPT-4o with different modality inputs. **Bold values** are the best performance and underlined values are the second-best performance.

MLLMs' UI code generation effectiveness hinges on interaction comprehension, with complex or visually subtle interactions being particularly challenging when using images alone. Natural language descriptions can complement visual inputs. To investigate the impact of different input signals, we conduct experiments on GPT-4o using 10 randomly selected webpages from failure cases. Human annotators provide textual descriptions for each interaction (e.g., "clicking the login button triggers a new window with two input boxes"). We evaluate three settings: visual input only (V), textual description only (T), and combined visual-textual input (V+T). Table 6 shows that **visual-only (V) and text-only (T) inputs exhibits unsatisfactory performance (Limitation 4)**, the combined approach (V+T) consistently outperforms single-modality inputs across all prompt types, indicating complementary benefits.

Improvement 4: Visual and Textual Description Combination. Combined visual and textual inputs can optimize MLLMs' Interaction-to-Code performance.

6 Conclusion

We present the first systematic study of MLLMs' capabilities in generating interactive webpages. We formulate the **Interaction-to-Code** task and establish the **Interaction2Code** benchmark. Through comprehensive experiments, we identify four critical limitations: (1) inadequate generation of interaction compared with full page, (2) susceptibility to ten types of failures, (3) poor performance on visually subtle interactions, and (4) insufficient comprehension when limited to single-modality visual descriptions. To address these limitations, we propose four enhancement strategies: interactive element highlighting, failure-aware prompting (FAP), visual saliency enhancement, and the integration of visual-textual descriptions.

Limitations

While Interaction2Code establishes a foundation for evaluating web interaction generation, several opportunities exist for future enhancement and research directions:

- Extending from interactive webpages to full-stack website development. Some complex functional interactions (e.g., login, search, etc.) are implemented by server-side scripting languages like Python. This also requires the evaluation to consider back-end functions beyond just front-end functions.
- Expand a single-page webpage to multiple pages. In real scenarios, a website usually has multiple interfaces, as well as external links and pictures. Therefore, a benchmark can be established to evaluate the ability of MLLM to generate multi-page and multi-resource websites.

References

Anthropic. 2024. Introducing claude 3.5 sonnet. Accessed: 2024-09-29.

Anthropic. 2024. Vision documentation. Accessed: 2024-10-18.

Batuhan Aşıroğlu, Büşta Rümeysa Mete, Eyyüp Yıldız, Yağız Nalçakan, Alper Sezen, Mustafa Dağtekin, and Tolga Ensari. 2019. Automatic html code generation from mock-up images using machine learning techniques. In *2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT)*, pages 1–4. Ieee.

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.

C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu. 2018. From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation. In *Proceedings of the 40th International Conference on Software Engineering*, pages 665–676.

Wen-Yin Chen, Pavol Podstreleny, Wen-Huang Cheng, Yung-Yao Chen, and Kai-Lung Hua. 2022. Code generation from a graphical user interface via attention-based encoder–decoder model. *Multimedia Systems*, 28(1):121–130.

André Armstrong Janino Cizotto, Rodrigo Clemente Thom de Souza, Viviana Cocco Mariani, and Leandro dos Santos Coelho. 2023. Web pages from mockup design based on convolutional neural network and class activation mapping. *Multimedia Tools and Applications*, 82(25):38771–38797.

Git. 2025. Git difference tool.

Google. 2024. Gemini api. Accessed: 2024-10-06.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Shaoling Dong, Xing Zhou, and Wenbin Jiang. 2024. Vision2ui: A real-world dataset with layout for code generation from ui designs. *arXiv preprint arXiv:2404.06369*.

Interactive. 2025. Ui mockups.

Vanita Jain, Piyush Agrawal, Subham Banga, Rishabh Kapoor, and Shashwat Gulyani. 2019. Sketch2code: transformation of sketches to ui in real-time using deep neural network. *arXiv preprint arXiv:1910.08930*.

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

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. 2024. Llava-next: Improved reasoning, ocr, and world knowledge.

Meta. 2024. Llama 3.2 vision. Accessed: 2025-02-06.

Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. 2018. Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46(2):196–221.

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.

OpenAI. 2024a. Hello gpt-4o. Accessed: 2024-10-06.

OpenAI. 2024b. Vision guide. Accessed: 2024-10-18.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Qwen. 2025. Qwen2.5-vl.

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Selenium. 2025. Selenium.

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*.

UI-Mockup. 2025. Ui mockups.

Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R Lyu. 2024. Automatically generating ui code from screenshot: A divide-and-conquer-based approach. *arXiv preprint arXiv:2406.16386*.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612.

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.

Y. Xu, L. Bo, X. Sun, B. Li, J. Jiang, and W. Zhou. 2021. image2emmet: Automatic code generation from web user interface image. *Journal of Software: Evolution and Process*, 33(8):e2369.

John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, et al. 2024. Swe-bench multimodal: Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*.

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*.

Jiarui Zhang, Jinyi Hu, Mahyar Khayatkhoei, Filip Ilievski, and Maosong Sun. 2024. Exploring perceptual limitation of multimodal large language models. *arXiv preprint arXiv:2402.07384*.

Ting Zhou, Yanjie Zhao, Xinyi Hou, Xiaoyu Sun, Kai Chen, and Haoyu Wang. 2024. Bridging design and development with automated declarative ui code generation. *arXiv preprint arXiv:2409.11667*.

A Basic Knowledge about Website Development

A.1 Website Development Process

A typical industrial application development lifecycle includes the following stages:

1. **Design stage**: designers create high-fidelity mock-ups and interactive prototyping using prototyping tools such as Sketch [1] and Axure [2] during design stage. UI Mock-ups represent the visual design of the interface without functionality. They are static representations that show the layout, color scheme, typography, and overall look and feel of the web application. Interactive prototyping takes the static mock-ups a step further by adding functionality and interactivity. These prototypes simulate the actual user experience by allowing users to interact with webpage.

2. **Development stage**: this phase involves transforming the design concepts into a functional application through coding. The development stage typically consists of GUI and underlying functionalities implementation.

A.2 Basic Knowledge about Front-end Development

Front-end development focuses on what users see and interact with in their web browsers. Visual design and interactive implementation are two key parts of creating visually appealing and user-friendly interfaces. The primary technologies used in front-end development are Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript.

A.2.1 HTML

HTML (HyperText Markup Language) is a markup language used to create web page content. It defines the structure and content of a web page through tags, such as titles, paragraphs, and buttons, as shown in Fig 9; each HTML element includes an opening tag, content, and a closing tag, forming the basic block of a webpage. HTML does not support complex interactions, but some specific elements (e.g., form, button) and attributes can be used to implement basic interactive functions. For example, the HTML code in Figure. 9 set the “draggable”

[1]https://www.sketch.com/
[2]https://www.axure.com/

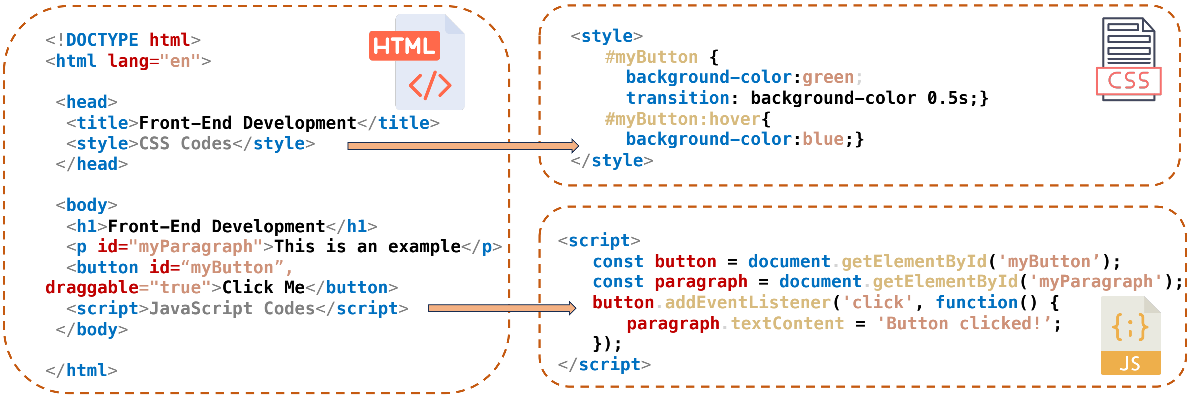


Figure 9: Example code of HTML, CSS and JavaScript.

attribute as true in the button flag to allow user to drag the button.

A.2.2 CSS

CSS (Cascading Style Sheets) is a style sheet language used to describe the style of HTML documents. It allows web developers to control the layout, fonts, colors, spacing, and other visual effects of the page. CSS can achieve interactive effects through pseudo-classes, pseudo-elements, transitions and animations. For example, the CSS program between the style tag in Figure. 9 leverages the ":hover" pseudo-class to add an interaction on the button. The button's color will change from green to blue once the mouse hovers. The transition ("transition: background-color 0.5s") can smoothly change the color of the button over 0.5 second to create an animation effect.

A.2.3 JavaScript

JavaScript is a high-level, dynamic, and versatile programming language that is primarily used for adding interactivity and dynamic behavior to websites. JavaScript enables developers to create rich, interactive user experiences, manipulate the Document Object Model (DOM), and handle events. For example, Figure. 9 shows that the JavaScript program between the script tag adds an event listener on the button, once clicked, the text content of the paragraph will be changed to "Button clicked!".

In summary, the interaction of the front end of the web page comes from HTML tags and attributes, style changes implemented by CSS, and custom events implemented by JavaScript.

A.3 Related Work

UI code generation techniques can be divided into three categories: Deep Learning (DL) based methods, Computer Vision (CV) based methods, and Multimodal Large Language Model (MLLM) based methods. (1) DL-based methods: (Aşıroğlu et al., 2019; Cizotto et al., 2023; Moran et al., 2018; Xu et al., 2021; Chen et al., 2018) leverages CNNs to automatically prototype software GUIs. Pix2code (Beltramelli, 2018) utilizes CNNs and LSTM to extract features from GUI images to generate a domain-specific language (DSL). (Chen et al., 2022) implements an encoder-decoder framework with an attention mechanism to generate the DSL. (2) CV-based methods: Sketch2Code (Jain et al., 2019) inputs hand-drawn sketches into object detection models to learn the object representation, which is read by the UI parser to generate code for targeted platforms. REMAUI (Nguyen and Csallner, 2015) identifies user interface elements via optical character recognition (OCR) techniques and then infers a suitable user interface hierarchy and exports the results as source code. (3) MLLM-based methods: with the help of MLLMs' powerful understanding of images, Design2Code (Si et al., 2024) generates UI code through text-augmented and self-revision prompting. To solve the element omission distortion and misarrangement problems during UI code generation, DCGen (Wan et al., 2024) proposes a divide-and-conquer-based approach to generate the code of the submodules separately and then assemble them to construct the full webpage. DeclarUI (Zhou et al., 2024) uses the element segmentation method to accurately generate elements and page transition graphs to prompt MLLMs to generate mobile app UI with jump logic. **Although the above works achieve decent performance on the UI code generation task, none of them consider the generation of interactive elements.**

B Quantitative Metrics of Interaction2Code benchmark

Quantitative Metrics. To measure the diversity and complexity of our dataset, we adopt the same statistical metrics as those in Design2Code (Si et al., 2024), with the results presented in Table 7. The Length indicates the token length obtained through the GPT-2 tokenizer (Radford et al., 2019), tag count refers to the number of tags in the HTML code, DOM depth signifies the maximum depth of the HTML's DOM Tree, and unique tags denote the number of unique tags in the HTML code. Table 7 shows that the data is rich in HTML tags (1,291 in a page on average).

Table 7: Quantitative metrics.

	Min	Max	Average	Std
Length (tokens)	82	769,466	127,604	165,046
Tag Count	2	5,739	983	1,038
DOM Depth	2	38	16	6
Unique Tags	2	52	28	11
Total size				127

C Interaction Part Extraction Method

After obtaining the screenshots before and after the interaction, we extract the interactive part from them to evaluate the generation effect of the interaction part. If the interaction does not change the size of the webpage, we can directly subtract the pixels of the two screenshots to obtain different areas (the area where the pixel value is not 0 after subtraction is the interaction area). However, some interactions will change the size of the web page (e.g., generating new components). In this case, we use the Git difference tool [3] to calculate the different row and column numbers of the two screenshots. The areas where these rows and columns intersect are the areas affected by the interaction. The algorithm is shown in Algorithm 1.

D Visual Categories of Interaction

Visual categories explanations are as follows:

- New component: it represents new elements are generated after an interaction. For example, as shown in Fig 11(c), two new input elements will be generated after selecting the third choice.
- Text: text change after interaction, As shown in Figure. 12(i), after clicking the "Select" button, the text on it will change to "Selected".

[3] https://git-scm.com/docs/git-difftool

Algorithm 1 Interaction Part Extraction Algorithm

Require:
Webpage screenshot A (Before interaction)
Webpage screenshot B (After interaction)
Ensure:
Coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$ of interaction region
if $dim(A) = dim(B)$ **then**
  $D \leftarrow |A - B|$;
  $C \leftarrow \{(x, y)|D(x, y) \neq 0\}$;
  $x_{min} \leftarrow \min\{x|(x, y) \in C\}$;
  $x_{max} \leftarrow \max\{x|(x, y) \in C\}$;
  $y_{min} \leftarrow \min\{y|(x, y) \in C\}$;
  $y_{max} \leftarrow \max\{y|(x, y) \in C\}$;
else
  $diff\_rows \leftarrow$ DiffTool(A, B);
  $diff\_cols \leftarrow$ DiffTool(A^T, B^T);
  $x_{min} \leftarrow \min(diff\_cols)$;
  $x_{max} \leftarrow \max(diff\_cols)$;
  $y_{min} \leftarrow \min(diff\_rows)$;
  $y_{max} \leftarrow \max(diff\_rows)$;
end if

return $(x_{min}, y_{min}, x_{max}, y_{max})$

- Color: it denotes the color change after interaction. For example, the background color change from while to dark after clicking the dark label as illustrated in Figure. 12(c).
- New window: it represents that a new window is generated after the interaction, such as a form popping up after clicking the contact button, as shown in Figure. 12(f).
- New page: it represents the webpage jumps to another page after interaction, such as clicking the login button to jump to login page.
- Position: it indicates that the position of the element changes after the interaction. For example, on a text editing website, clicking the right button can move the text from the left to the right.
- Size: it indicates that the size of the element changes after the interaction. For example, the text size will increase after clicking the large label as shown in Figure. 12(h).
- Switch: it indicates the switching of content. For example, in Figure. 11(b), after clicking the "M" button, the clothes parameter will be switched from "S" to "M".

E Experiment Detail

E.1 Prompt Design Details

The prompts are shown in Figure 10. In the **Direct prompt**, the first screenshot represents the original webpage state, while subsequent screenshots depict states after specific interactions. Requirements are applied to guide MLLMs in replicating the webpage design and interaction. Requirement 3 allows MLLM to number the interactions when generating code, so that in the automated testing phase, webdriver [4] can locate the interactive elements through the interaction ID (e.g., interact1) and perform the interaction automatically.

To achieve Interactive element highlighting, we design CoT and Mark prompt to let MLLM focus on the interactive part. For the **CoT prompt** (Wei et al., 2022), we use the instruction "let's think step by step" and design three intermediate steps: analyze the interaction effects, locate the interactive elements, and implement the interaction. For the **Mark prompt**, we use red bounding boxes to highlight the interaction area, prompting MLLMs to focus on the interactive parts.

To enable MLLM to avoid potential errors as much as possible when generating interactions, we design **Failure-aware prompt** to put the failure types in the prompt to guide MLLM to avoid corresponding failures.

E.2 Model Details

In configuring the MLLM models, we set the temperature to 1 and the maximum number of tokens output for Gemini-1.5-flash, GPT-4o, Claude-3.5-Sonnect as 4096. For the Qwen series models, the maximum output token are set to 2048. All other parameters were kept at their default settings as outlined in the relevant API documentation (Google, 2024; OpenAI, 2024b; Anthropic, 2024; Qwen, 2025).

E.3 Metrics Details

We employ both full webpage metric and interactive part metric to judge the capability of MLLMs in the Interaction-to-Code task. We measure the quality of webpages generated by MLLMs from the perspectives of visual, structure, and text:

- **Visual Similarity**. We use CLIP score (Radford et al., 2021) to measure the visual similarity. This metric measures the semantic similarity between the generated and original webpages, serving as an indicator of how effectively the generated GUI captures the intended visual elements and overall design concept.
- **Structure Similarity**. SSIM (Wang et al., 2004) (Structural Similarity Index Measure) score is applied to calculate the structure similarity. It evaluates the layout and compositional accuracy, emphasizing the spatial arrangement and structural similarities between the generated and original webpages.
- **Text Similarity**. We first use python OCR tools to recognize the text in the original and the generated webpages, and then use the Bilingual Evaluation Understudy (BLEU) score (Papineni et al., 2002) to measure the text similarity between the two web pages.

For the interactive parts of webpages, in addition to the above visual, structure and text similarity, we also evaluate them from the perspective of the position and function of the interaction.

- **Position Similarity**. The position similarity between original interaction I_o and generated interaction I_g is defined as follows:

$$P(I_o, I_g) = 1 - max(|x_o - x_g|, |y_o - y_g|), \quad (1)$$

where (x_o, y_o) and (x_g, y_g) are normalized coordinates (in $[0, 1]$) of the center of the interactive area.

- **Implement Rate (IR)** measures the percentage of interactions successfully implemented by MLLM. An interaction is considered implemented if detectable by webdriver, and unimplemented otherwise. Let $N(\cdot)$ denote the quantity, we can calculate the **IR** as:

$$IR = \frac{N(implemented)}{N(implemented) + N(unimplemented)} \quad (2)$$

- **Function Usability**. This metric is used to measure whether the interactive function is usable, human annotators are asked to interact with the generated webpage and judge the usability. Let $N(\cdot)$ denote the quantity, we can calculate the **Usability Rate (UR)**:

$$UR = \frac{N(usable)}{N(usable) + N(unusable)}. \quad (3)$$

[4] https://selenium-python.readthedocs.io/

Direct Prompt

[Instruction]:

You are a web developer proficient in HTML, CSS and JavaScript. The user provides some screenshots of a webpage. The first screenshot ***[image1]*** shows the webpage in its original state, while the second screenshot ***[image2]*** shows the webpage after the user has interacted with certain element. You are tasked with creating a webpage that replicates the design and interaction observed in screenshots.

[Requirements]:

1. Design Replication: Pay attention to layout, color and so on to make the webpage look identical to the first screenshot .

2. Interaction Replication : Implement the changes shown in screenshots caused by interactions (e.g., clicks).

3. Number Interactions: You need to encode the interact id in the interactive element. For example, if the button is clicked in the second screenshot, the id of the button is set to interact1: "<button id="interact1">Click Me!</button>"

...

Combine HTML, CSS and JavaScript codes into one file and respond the codes only:

Chain-of-Thought (CoT) Prompt

[Instruction]

[Requirements]

[CoT]

You should think step by step:

Step 1: Understand the interaction effects by analyzing the difference between the first and other screenshots.

Step 2: Locate interactive elements.

Step 3: Implement the interaction: the interaction function should cause the difference you analyze in Step 1 and be implemented the interactive element you locate from step 2.

Combine HTML, CSS and JavaScript codes into one file and respond the codes only:

Mark Prompt

[Instruction]

[Requirements]

[Mark]

In the first screenshot, the interactive elements are highlighted with red bounding boxes. In other screenshots, the interaction effects are highlighted with red bounding boxes. Pay attention to the position of the red bounding boxes, which mark the position of interaction. But do not generate the red bounding box, which is just used for marking the interaction area.

Combine HTML, CSS and JavaScript codes into one file and respond the codes only:

Failure-aware Prompt (FAP)

[Instruction]

[Requirements]

[Failure Type Example]

There are ten types of errors you should avoid:

(a) Interactive element missing: MLLMs do not generate interactive elements...

(b) No interaction: There is no interaction in the generated webpage...

(c) Wrong interactive element: MLLMs implement the interactive function on the wrong element...

(d) (e) (f) (g) (h) (i) (j)...

Combine HTML, CSS and JavaScript codes into one file and respond the codes only:

Figure 10: The four kinds of prompts for MLLMs.

E.4 Human Annotation Guidelines

E.4.1 Webpage Selection Guidelines

Task Overview

You will be given some web links with diverse topics. Your task is to select some webpages from different topics.

Guidelines

1. Complexity: Each webpage must contain at least one meaningful interactive element.

2. Diversity: The selection process should include multiple different types of interactions to ensure that the selected pages are diverse.

3. User Experience: When selecting webpages, ensure that the layout and design of the pages are user-friendly. Avoid selecting pages that are too cluttered.

4. Accessibility: The selected pages should meet basic accessibility standards to ensure that all users, including those with special needs, can interact effectively.

5. Representativeness: Strive to select webpages that represent a wide range of specific topics to ensure that the sample is representative.

E.4.2 Interaction Annotation Guidelines

Task Overview

You will be given a webpage link. Your task is interacting with the webpage and choose 1-10 meaningful interactions for annotation. You should take screenshots before interaction and after interaction for interactive prototyping construction. The selection guidelines are:

Guidelines

1. Functional dimension

- *1.1 Interactions to achieve user goals: (1) Complete form submission; (2) Perform information retrieval; (3) Implement data screening; (4) Complete purchase process.*
- *1.2 Interactions to change status/data: (1) Update user settings (2) Modify content status (3) Save/delete data (4)Switch display mode.*

2. User experience dimension

- *2.1 Interactions to provide feedback: (1) Operation confirmation prompt; (2) Status update display; (3)Error message prompt; (4)Loading progress indicator*

3. Business value dimension

- *3.1 Interactions to promote business processes: (1) User registration/login; (2) Order processing; (3) Payment process; (4) Information collection.*
- *3.2 Interactions to improve conversion: (1) Product purchase;(2) Sharing function; (3) Collect/follow; (4)Rating and evaluation.*

E.4.3 Usability Annotation Guidelines

Task Overview
You will be given a reference interactive prototyping IP consisting of two screenshots, as well as one webpage that try to implement the interaction of the reference interactive prototyping. Your task is to judge whether the interaction indicated in the IP is usable in the webpage.
Guidelines
1. Evaluation should focus exclusively on the interactive functionality, disregarding overall visual appearance.
2. An interaction is considered usable if its implementation precisely matches the behavior specified in the interactive prototype.
3. In cases where the implemented interaction differs from the prototype, evaluate whether it effectively achieves the intended goals. The interaction is considered usable if it accomplishes the desired goal, despite implementation variations; otherwise, it is deemed unusable.

E.4.4 Pair Wise Comparison Guidelines

Task Overview
You will be given a reference interactive prototyping consisting of two screenshots, as well as two candidate webpages that try to implement the interaction of the reference interactive prototyping. Your task is to judge which of the two candidates implements the interaction better.
Guidelines
*1. **Function Check**: (1) **Interactive Elements:** Check whether the interactive elements are correct and pay attention to the types of interactive elements. (2) **Interactive Effect:** Check whether the effect after interaction is correct. Please pay attention to the changes after interaction.*
*2. **Appearance Check** (1) **Content Check:** Check whether there are any missing elements. (2)**Layout Check:** Check if their organization, order, and hierarchy match the reference.*
*3. **Comparison***
Based on the criteria in the order of priority (Function > Appearance), make an overall judgment on which webpage is more similar to the reference interactive prototyping.
Judgment Options
(1)Select "Example 1 better" if Example 1 is closer to the reference. (2) Select "Example 2 better" if Example 2 is closer to the reference. (3) Select "Tie" only if both examples are similarly or equally distant from the reference.

E.4.5 Failure Annotation Guidelines

Task Overview
You will be given a reference interactive prototyping IP consisting of two screenshots, as well as one webpage that try to implement the interaction of the reference interactive prototyping. Your task is to determine whether the given webpage has failures shown in below.
Failure Type
Here are x types of error examples:
a) Interactive element missing: MLLMs do not generate interactive elements. [Example]
b) No interaction: There is no interaction in the generated webpage. [Example]
c) Wrong interactive element: MLLMs implement the interactive function on the wrong element. [Example]
.....
Guidelines
If there are failures, but they do not belong to the failure types above, you need to mark them as unknown failures, and then further discuss to determine the type of failure. If there are no errors, mark them as "no failure".

F Failure Type and Explanation

F.1 Failure on Interactive Elements

(a) Interactive element missing: MLLMs do not generate interactive elements. As shown in Figure 11(a), there is a chat button in the upper right corner of the reference web page. When clicked, a chat window pops up. However, there is no such button in the generated web page, and users cannot perform any operation.

(b) No interaction: There is no interaction in the generated webpage. As shown in Figure 11(b), clicking button M in the original webpage will switch to the information of size "M". However, clicking "M" button in the generated, there is no change of the size information. It should be noted here that sometimes the lack of interaction does not result in the unavailability of functions. For example, suppose a web page contains a menu bar that can display detailed information after clicking. If MLLM

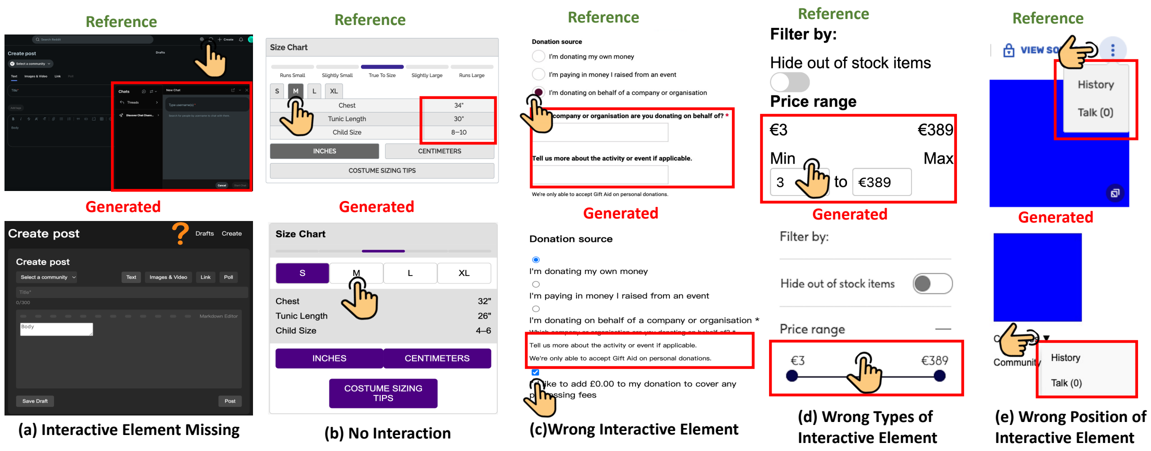


Figure 11: Failure on interactive elements.

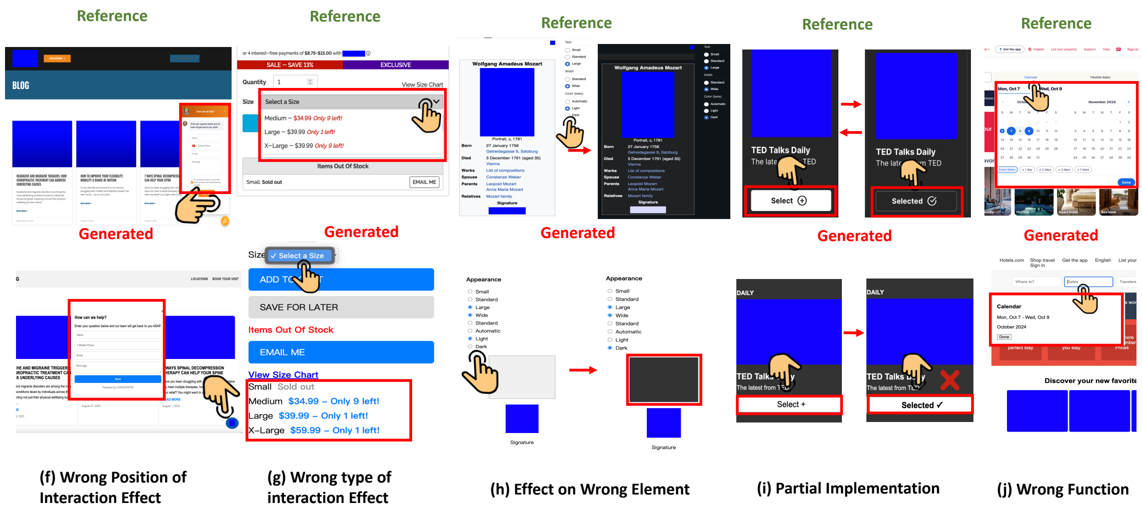


Figure 12: Failure of interaction effects.

does not achieve the click effect, but has displayed the detailed menu information, it does not affect the functionality of the web page.

(c) Wrong interactive element: MLLMs implement the interactive function on the wrong element. As shown in Figure 11(c), in the original webpage, after clicking "I'm donating on behalf of a company or organisation", two input boxes will appear. However, in the generated webpage, the input box will only appear after clicking "I' like to add 0.00 to my donation to cover any fees."

(d) Wrong type of interactive element: The types of interactive elements generated by MLLM are wrong. As shown in Figure 11(d), the element for adjusting the price in the original web page is of input type, while the element for adjusting the price in the generated web page is of progress type.

(e) Wrong position of interactive element: The interactive elements generated by MLLM are positioned incorrectly. As shown in Figure 11(e), the button in the original webpage is in the upper right corner of the image, while the generated button is below the image.

F.2 Failure on Interactive Effects

(f) Wrong position after interaction: The interactive effects generated by MLLM are in the wrong position. As shown in Figure 12(f), after clicking the dialogue button, the pop-up window is displayed in the lower left corner of the reference webpage, but appears in the middle of the generated webpage.

(g) Wrong type of interaction effects: As shown in Figure 12(g), in the reference webpage, the element that appears after clicking select is of option type, but in the generated web page, the element that appears is of text type.

(h) Effect on wrong element: MLLMs achieve the effect of interaction on the wrong elements. As shown in Figure 12(h), in the reference webpage, after clicking the "dark" button, the background color of the web page turns black. However, in the generated web page, after clicking the "dark" button, the block turns black and the background does not change.

(i) Partial Implementation: MLLMs only implement a part of the interactive functionality. As shown in Figure 12(i), in the reference webpage, after clicking the select button, the button will become selected, and will return to its original state when clicked again. However, in the generated web page, the button can only be selected but not unselected.

(j) Wrong function: MLLM implements the wrong function. As shown in Figure 12(j), in the original webpage, clicking the button will cause a date selection box to appear, but in the generated webpage, clicking the button will generate a date display box.

G Experimental Results Details

G.1 MLLMs' Performance under Different Interaction Scenarios

We study the performance of MLLMs on the Interaction-to-Code task under different interaction types. The results of varying tag categories with high frequency and visual categories are shown in Table 8 and Table 9, respectively.

For tag categories, FORM, SELECT, and OPTION are the easiest interaction types to generate, achieving a usability rate higher than 80%. This is because these interactions scenarios always contain fixed patterns, for example, SELECTION and OPTION only appear in drop-down lists, and FORM often merely contains input boxes. In contrast, IFRAME and PROGRESS elements show lower usability rates (<60%), attributed to their complexity: IFRAMES involve embedding external content, while PROGRESS bars require intricate component coordination for functions like audio control or price range adjustment, raising difficulties for MLLM to understand.

For visual categories, MLLMs excel at generating interactions that result in prominent visual changes, such as creating new windows, and components. However, they struggle with subtle visual modifications, such as color shifts and positional adjustments, indicating their limitations in handling fine-grained interaction effects.

Finding: Performance varies by interaction type: MLLMs are good at handling interactions with fixed pattern (e.g., selection list) and obvious changes (e.g., new window creation), while struggling with interactions involving complex changes (e.g., iframe, progress) and subtle visual modifications (e.g., position change).

G.2 Full results of Failure-aware Prompting

Table 10 shows the full results of failure-aware prompting results. We can find that for commercial models and the open source 72B model Qwen2.5-vl-72B-instruct, failure-aware prompt can guide the model to use self-critic ability to avoid potential errors. However, for 3B and 7B models, due to their own limitations in understanding failure samples, the performance will decrease after using FAP.

G.3 Full Results of Modality influence

Table 11 shows the influence of different modalities, we can find that combine visual and textual modality can optimize the models' performance.

H Tool

The Interaction2Code tool is shown in Figure 13. The tool comprises several key components: a model selector, a prompt method chooser, and three main functional modules for code download, webpage preview, and code generation. Users can upload webpage screenshots both before and after their intended interactions, allowing the system to analyze the interaction and generate corresponding HTML code.

I Visual Comparison of UI2Code Benchmark and Interaction2Code Benchmark

Figure 14 shows the comparison between UI2Code benchmark and our Interaction2Code benchmark. UI2Code benchmark only contains the static webpage, whereas Interaction2Code contains interactive webpage, which is represented by interactive prototyping.

Table 8: Usability rate of different tag categories.

Model	Prompt	button	input	span	link	select	textarea	option	iframe	text	progress
Qwen2.5-vl-3B-instruct	Direct	0.1149	0.4038	0.0541	0.0556	0.3143	0.2000	0.2258	0.2500	0.0000	0.0909
	CoT	0.1660	0.4231	0.1081	0.0833	0.4571	0.2571	0.3710	0.2500	0.0625	0.2500
	Mark	0.2199	0.4679	0.1802	0.1204	0.5143	0.4000	0.4516	0.3214	0.1111	0.3333
Qwen2.5-vl-7B-instruct	Direct	0.2830	0.4856	0.2095	0.2222	0.5500	0.4071	0.5081	0.3482	0.2188	0.2955
	CoT	0.3779	0.5462	0.3459	0.3444	0.6229	0.4857	0.5806	0.4143	0.3000	0.3545
	Mark	0.4206	0.5833	0.3829	0.3519	0.6619	0.5286	0.6237	0.4405	0.3125	0.3939
Qwen2.5-vl-72B-instruct	Direct	0.3812	0.5659	0.3398	0.3095	0.6163	0.4816	0.5714	0.4184	0.2679	0.3636
	CoT	0.3516	0.5385	0.3176	0.2882	0.5964	0.4607	0.5565	0.3839	0.2396	0.3409
	Mark	0.3537	0.5470	0.3333	0.2809	0.5937	0.4698	0.5735	0.3889	0.2500	0.3434
Gemini-1.5-flash	Direct	0.3745	0.5462	0.3486	0.3111	0.6000	0.4771	0.5968	0.4071	0.2667	0.3545
	CoT	0.4085	0.5664	0.3980	0.3510	0.6182	0.5013	0.6188	0.4188	0.3106	0.3678
	Mark	0.4305	0.5849	0.4167	0.3588	0.6429	0.5238	0.6452	0.4226	0.3264	0.3712
GPT-4o	Direct	0.4062	0.5725	0.3867	0.3355	0.6198	0.4989	0.6154	0.4093	0.3013	0.3531
	CoT	0.3960	0.5604	0.3707	0.3353	0.6082	0.4857	0.6106	0.3954	0.2917	0.3442
	Mark	0.4037	0.5667	0.3838	0.3259	0.6076	0.5086	0.6237	0.4000	0.3000	0.3455
Claude-3.5-Sonnet	Direct	0.4186	0.5709	0.3986	0.3507	0.6089	0.5089	0.6250	0.4018	0.3281	0.3665
	CoT	0.4431	0.5837	0.4277	0.3758	0.6235	0.5294	0.6414	0.4202	0.3529	0.3877
	Mark	0.4612	0.6004	0.4399	0.3827	0.6397	0.5524	0.6577	0.4345	0.3657	0.3965
Average		0.3561	0.5396	0.3245	0.2879	0.5831	0.4598	0.5609	0.3847	0.2558	0.3362

J Summarization

Limitation 1:The performance of the MLLMs in the interactive part is lower than that of the full page.
Limitation 2:The MLLMs are prone to make 10 types of failure.
Limitation 3: MLLMs perform poorly on interactions that are not visually obvious.
Limitation 4: Single visual modality description is not enough for MLLMs to understand the interaction.

Improvement 1: Interactive element highlighting. To improve the performance of generated interaction, we further propose *Chain-of-Thought (CoT) and Mark prompts* to force models to focus on the interaction.
Improvement 2: Failure-aware Prompt (FAP). Based on failure types, we propose FAP to stimulate the self-criticism ability of MLLM, thereby avoiding problems that may occur in the Interaction-to-Code task.
Improvement 3: Visual Saliency Enhancement (VSE). By cropping the image to increase the proportion of the interactive part, VSE makes the model to better perceive the interaction area.

Improvement 4: Visual and Textual Description Combination. Combined visual and textual inputs can optimize MLLMs' Interaction-to-Code performance.

Table 9: Usability rate of different visual categories.

Model	Prompt	text	new component	color	position	switch	new page	new window	size
Qwen2.5-vl-3B-instruct	Direct	0.1667	0.1366	0.1765	0.0889	0.0976	0.0556	0.1176	0.2500
	CoT	0.2438	0.2112	0.2824	0.1333	0.1341	0.0694	0.1765	0.2750
	Mark	0.3086	0.2961	0.3294	0.2000	0.1870	0.1019	0.2549	0.3500
Qwen2.5-vl-7B-instruct	Direct	0.3534	0.3509	0.3382	0.2556	0.2317	0.2500	0.3088	0.4000
	CoT	0.4358	0.4335	0.3953	0.3600	0.3220	0.3889	0.4235	0.4400
	Mark	0.4825	0.4803	0.4294	0.4037	0.3821	0.4213	0.4755	0.5000
Qwen2.5-vl-72B-instruct	Direct	0.4383	0.4348	0.3950	0.3619	0.3449	0.3730	0.4202	0.4714
	CoT	0.4105	0.4022	0.3838	0.3250	0.3201	0.3403	0.3860	0.4437
	Mark	0.4198	0.4106	0.3922	0.3309	0.3171	0.3148	0.3954	0.4500
Gemini-1.5-flash	Direct	0.4309	0.4286	0.3882	0.3511	0.3439	0.3444	0.4176	0.4550
	CoT	0.4641	0.4585	0.4139	0.3818	0.3792	0.3939	0.4572	0.4864
	Mark	0.4887	0.4840	0.4284	0.3981	0.4065	0.4120	0.4804	0.5083
GPT-4o	Direct	0.4649	0.4577	0.4090	0.3726	0.3827	0.3846	0.4525	0.4885
	CoT	0.4524	0.4494	0.4017	0.3603	0.3711	0.3810	0.4370	0.4714
	Mark	0.4609	0.4576	0.4118	0.3585	0.3707	0.3796	0.4529	0.4800
Claude-3.5-Sonnet	Direct	0.4734	0.4674	0.4206	0.3764	0.3857	0.4080	0.4614	0.4938
	CoT	0.4956	0.4881	0.4401	0.3987	0.4118	0.4379	0.4827	0.5118
	Mark	0.5117	0.5072	0.4556	0.4160	0.4295	0.4475	0.5065	0.5250
Average		0.4167	0.4085	0.3828	0.3262	0.3232	0.3280	0.3948	0.4444

Table 10: Performance comparison between Direct and FAP methods (Full results of RQ6).

Model	Method	Full Page			Interaction Part				
		CLIP	SSIM	Text	CLIP	SSIM	Text	Position	IR
Gemini-1.5-flash	Direct	0.6276	0.4984	0.5231	0.5403	0.4494	0.3602	0.5802	0.7636
	FAP	0.6580	0.5337	0.5311	0.5886	0.4584	0.4394	0.6032	0.8182
	Δ	↑ 0.0304	↑ 0.0353	↑ 0.0080	↑ 0.0483	↑ 0.0090	↑ 0.0792	↑ 0.0230	↑ 0.0546
GPT-4o	Direct	0.6660	0.5480	0.4995	0.5700	0.4891	0.3652	0.5803	0.7636
	FAP	0.7047	0.5976	0.6045	0.6072	0.5405	0.4580	0.6452	0.8364
	Δ	↑ 0.0387	↑ 0.0496	↑ 0.1050	↑ 0.0372	↑ 0.0514	↑ 0.0928	↑ 0.0649	↑ 0.0728
Claude-3.5-Sonnet	Direct	0.5747	0.3950	0.4611	0.4582	0.3771	0.3086	0.4927	0.6364
	FAP	0.6080	0.4500	0.4810	0.4921	0.4035	0.3822	0.5154	0.6545
	Δ	↑ 0.0333	↑ 0.0550	↑ 0.0199	↑ 0.0339	↑ 0.0264	↑ 0.0736	↑ 0.0227	↑ 0.0181
Qwen2.5-vl-3B-instruct	Direct	0.4284	0.2466	0.1674	0.2777	0.2180	0.0285	0.3020	0.4727
	FAP	0.3647	0.2076	0.1146	0.2375	0.1867	0.0328	0.2213	0.3818
	Δ	↓ 0.0637	↓ 0.0390	↓ 0.0528	↓ 0.0402	↓ 0.0313	↑ 0.0043	↓ 0.0807	↓ 0.0909
Qwen2.5-vl-7B-instruct	Direct	0.3596	0.1981	0.1758	0.2802	0.1894	0.0854	0.2580	0.4000
	FAP	0.3828	0.1642	0.1948	0.2603	0.1747	0.0746	0.2419	0.4182
	Δ	↑ 0.0232	↓ 0.0339	↑ 0.0190	↓ 0.0199	↓ 0.0147	↓ 0.0108	↓ 0.0161	↑ 0.0182
Qwen2.5-vl-72B-instruct	Direct	0.6169	0.3967	0.4060	0.4741	0.3612	0.3275	0.5022	0.6545
	FAP	0.6194	0.4208	0.4426	0.5144	0.3750	0.3286	0.5376	0.7636
	Δ	↑ 0.0025	↑ 0.0241	↑ 0.0366	↑ 0.0403	↑ 0.0138	↑ 0.0011	↑ 0.0354	↑ 0.1091

Table 11: Performance of MLLMs with different modality inputs. Bold values are the best performance and underlined values are the second-best performance.

Prompt	Modality	Gemini-1.5-flash				GPT-4o			
		CLIP	SSIM	Text	Position	CLIP	SSIM	Text	Position
Direct	V	0.3338	0.1587	**0.2777**	0.3342	0.3737	0.1793	0.2539	0.3951
	T	0.3116	0.1550	0.1687	0.3999	0.4174	0.4067	0.2316	0.4293
	V+T	**0.5679**	**0.3010**	0.2732	**0.5964**	**0.6735**	**0.5612**	**0.3919**	**0.7157**
CoT	V	0.4357	0.1975	0.3072	0.4303	0.3871	0.3101	0.2433	0.4461
	T	0.3677	0.0897	0.2290	0.4403	0.5579	0.1828	0.3045	0.5465
	V+T	**0.5503**	**0.4027**	**0.3558**	**0.5656**	**0.6440**	**0.4800**	**0.4287**	**0.7080**
Mark	V	0.4502	0.3256	0.2197	0.4302	0.5015	**0.4520**	0.3389	0.5025
	T	0.5019	0.2478	0.2921	**0.5301**	0.4613	0.4454	0.2805	0.4810
	V+T	**0.5946**	**0.4327**	**0.3416**	0.4791	**0.6923**	0.4336	**0.4248**	**0.7469**

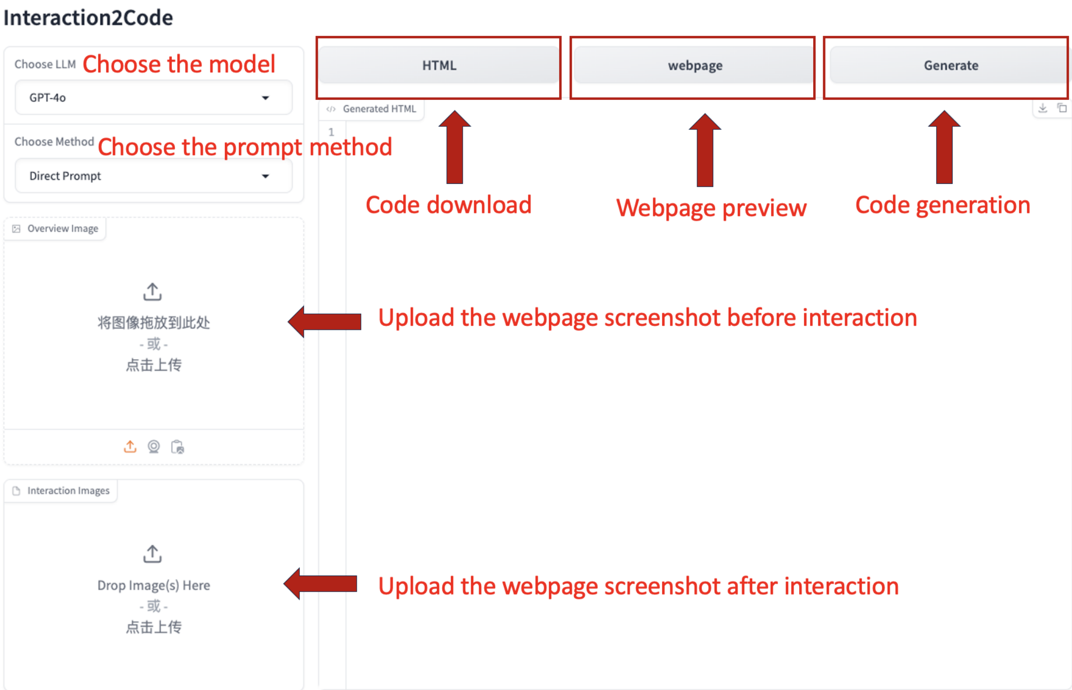


Figure 13: The interactive webpage generation tool.

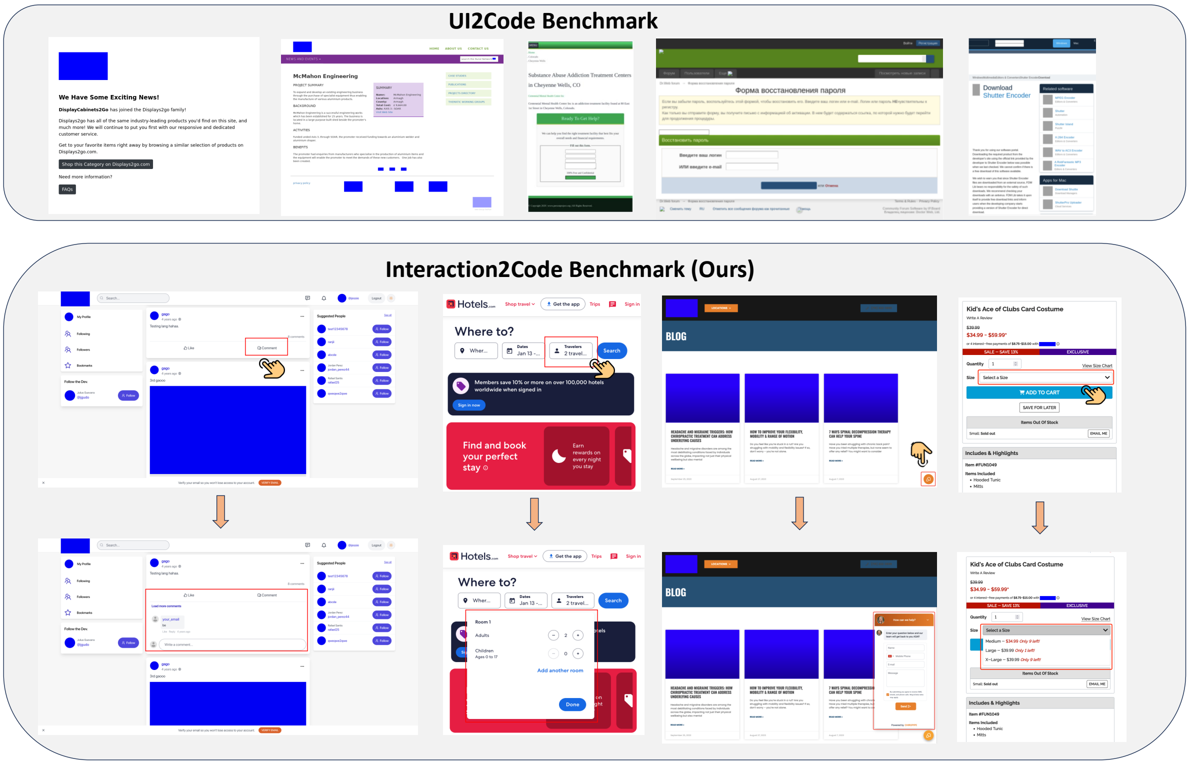


Figure 14: Comparison between UI2Code benchmark and our Interaction2Code benchmark. UI2Code benchmark only contains the static webpage, whereas Interaction2Code contains interactive webpage, which is represented by interactive prototyping.