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[3],
Zhiyao Xu[4], Yuhang Wang[5], Michael R. Lyu[1]

[1]The Chinese University of Hong Kong, China [2] Singapore Management University, Singapore
[3] Renmin University of China, China [4] Tsinghua University, China [5] Southwest University, China

{jyxiao, yxwan}@link.cuhk.edu.hk, ythuo@smu.edu.sg, {zixinwang, xinyixu}@link.cuhk.edu.hk, jwxwang@gmail.com,
xu-zy25@mails.tsinghua.edu.cn, wyh20030323@email.swu.edu.cn, lyu@cse.cuhk.edu.hk

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 understanding 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. Our data and code are available in https://github.com/WebPAI/Interaction2Code.

Index Terms—**Multimodal Large Language Models, Code Generation, Web Development.**

I. INTRODUCTION

As of 2025, the digital landscape consists of approximately 1.09 billion websites [1], supporting a wide range of applications in daily life. The design and development of Graphical User Interfaces (GUIs) play a critical role in shaping a website's aesthetics and functionality, improving user satisfaction through effective layout, colors, and typography [2], [3]. However, translating these designs into functional code is complex and time-consuming, often leading to discrepancies between the design and final implementation [2], [4]–[7]. To address this, automated GUI code generation methods have been proposed, categorized into learning-based and LLM-based approaches.

Learning-based methods like Pix2code [8] use CNNs and LSTMs to reverse-engineer GUI code from images, while Chen et al. [9] present a neural machine translator for encoding UI features and generating GUI skeletons. However, these methods struggle with generalizing to diverse web elements. Recent advances in Multimodal Large Language Models (MLLMs), such as GPT-4o [10], Claude-3.5 [11], and Gemini-1.5 [12], have improved visual understanding tasks [13]–[15] and shown remarkable performance in code generation [16]–[21]. These advances open new possibilities for the Design-to-Code task, where MLLMs generate code from screenshots to replicate web elements, layout, text, and colors. For instance, Design2Code [22] uses prompts to guide MLLMs in web content understanding and code generation, while DCGen [23] proposes a divide-and-conquer approach to improve the accuracy of generated webpage elements.

However, existing research only focuses on the static appearance of a webpage (e.g., color, layouts) [22], [24]–[27], 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.
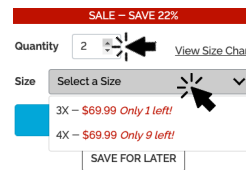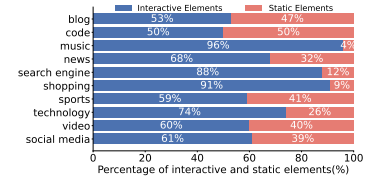
(a) Interactive elements.   (b) Interactive and static ratio.

Fig. 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 overall 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, thereby improving the performance of interaction generation. (4) **Visual and textual description combination** can makes MLLMs understand the interaction better.

II. BACKGROUND

A. Related Work

UI code generation techniques fall into three categories: Deep Learning (DL)-based, Computer Vision (CV)-based, and Multimodal Large Language Model (MLLM)-based methods. (1) DL-based methods: [2], [28]–[31] use CNNs to prototype GUIs automatically. Pix2code [8] combines CNNs and LSTM to generate a domain-specific language (DSL) from GUI images. [32] uses an encoder-decoder framework with attention mechanisms to generate DSL. (2) CV-based methods: Sketch2Code [33] feeds hand-drawn sketches into object detection models to generate code. REMAUI [34] uses OCR to identify UI elements and generate corresponding source code. (3) MLLM-based methods: Design2Code [22] leverages MLLMs for UI code generation using text-augmented and self-revision prompting. To address issues like element omission and misarrangement, DCGen [23] adopts a divide-and-conquer approach, generating

TABLE I
COMPARISONS BETWEEN INTERACTION2CODE AND EXISTING UI2CODE BENCHMARKS.

Benchmark	Real World	Failure Annotation	Interactive
WebSight [37]	✗	✗	✗
Vision2UI [25]	✓	✗	✗
Design2Code [22]	✓	✗	✗
DesignBench [26]	✓	✗	✗
Interaction2Code (Ours)	✓	✓	✓

submodules separately before assembling them. DeclarUI [35] uses element segmentation and page transition graphs to guide MLLMs in generating mobile app UI with jump logic. While these approaches show strong performance, none focus on generating interactive elements. EfficientUICoder [36] applies token compression method to accelerate UI code generation. **Although the above works achieve decent performance on the UI2Code task, none of them consider the generation of interactive webpages.**

B. Problem Definition

UI-Mockup [38] 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** [39] 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.

III. THE INTERACTION2CODE BENCHMARK

A. 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 Common-Crawl (C4 validation set [40]) and GitHub. (1) CommonCrawl: Following the Design2Code approach [22], we automatically filter out webpages that are either too long or too simple (containing only images or text) and perform deduplication. We then select 15 common web topics and randomly sample 1,000 webpages related to these topics. Four PhD students majoring in computer science, each with front-end development experience, are tasked with selecting approximately 25 webpages each, resulting in 100 webpages from C4. The selection criteria are: 1) complexity: each webpage must include at least one meaningful interaction; 2) diversity: the selection aims to cover a broad
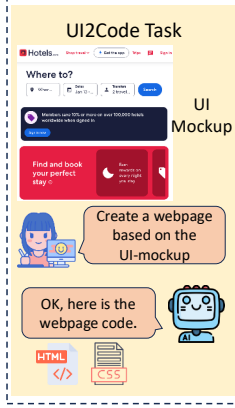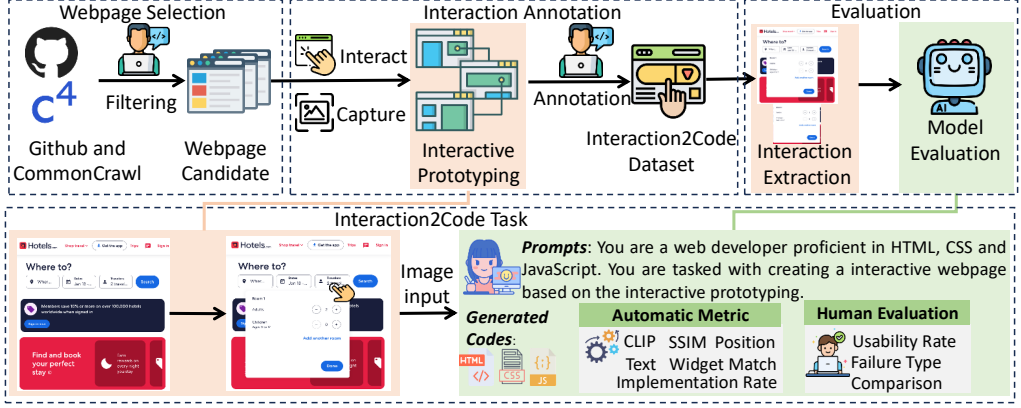
Fig. 2. UI2Code.

Fig. 3. The construction of Interaction2Code benchmark.

range of webpages with different interaction types. Since most C4 websites are traditional and do not use UI frameworks, we also collect webpages from GitHub projects that use UI frameworks. (2) GitHub projects: We search for "open-production-web-projects" and "awesome-opensource-apps" on GitHub to compile a list of web projects, from which we identify 27 popular projects with deployed links and high star counts. These projects, averaging 13k stars, represent various real-world website uses, ranging from commercial product frontends to blogs, demonstrating their quality and popularity. Ultimately, we compile a dataset of 127 webpages. Detailed selection guidelines are provided in our artifact.

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-interaction screenshots to build interactive prototyping (the guideline is shown in our artifact). 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.

B. 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 II and play a great role in human-website interaction. Visual categories
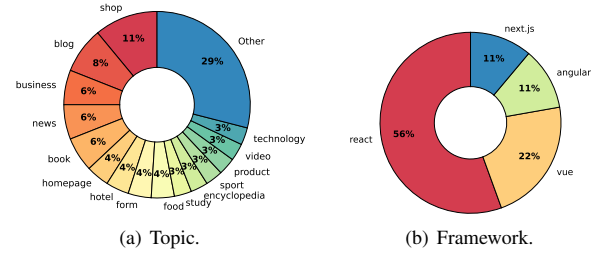

(a) Topic. (b) Framework.

Fig. 4. Topic and framework distribution.

TABLE II
TAG AND VISUAL CATEGORIES DISTRIBUTION.

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	-	-	-	-

involve changes in color, size, position, text, etc. Note that one interaction may belong to multiple tag categories and visual categories. Table II demonstrates that Interaction2Code benchmark has a rich set of interaction types, including 23 tag categories and 8 visual categories.

C. 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 [41] 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 the Git diff tool [42] to locate modified rows and columns, with their intersections marking the affected regions. The algorithm is shown in Algorithm 1.

Algorithm 1 Interaction Part Extraction Algorithm

Require:
1: Webpage screenshot A (Before interaction)
2: Webpage screenshot B (After interaction)
Ensure:
3: Coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$ of interaction region
4: **if** $dim(A) = dim(B)$ **then**
5:    $D \leftarrow |A - B|$;
6:    $C \leftarrow \{(x, y) | D(x, y) \neq 0\}$;
7:    $x_{min} \leftarrow \min\{x | (x, y) \in C\}$;
8:    $x_{max} \leftarrow \max\{x | (x, y) \in C\}$;
9:    $y_{min} \leftarrow \min\{y | (x, y) \in C\}$;
10:    $y_{max} \leftarrow \max\{y | (x, y) \in C\}$;
11: **else**
12:    $diff\_rows \leftarrow \text{DiffTool}(A, B)$;
13:    $diff\_cols \leftarrow \text{DiffTool}(A^T, B^T)$;
14:    $x_{min} \leftarrow \min(diff\_cols)$;
15:    $x_{max} \leftarrow \max(diff\_cols)$;
16:    $y_{min} \leftarrow \min(diff\_rows)$;
17:    $y_{max} \leftarrow \max(diff\_rows)$;
18: **end if**
19:
20: **return** $(x_{min}, y_{min}, x_{max}, y_{max})$

Full Page Metrics. We measure the quality of generated webpages from the following perspectives: **(1) Visual Similarity**. We use CLIP score [43] to measure the visual similarity. **(2) Structure Similarity**. SSIM [44] (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 [45] to measure the text similarity between the two webpages. **(4) Widget Match.** Widget match [46], [47] measures the widget-level consistency between the original UI and the generated UI. We calculate the Widget Similarity (WS) and Widget Match Rate (WMS) based on the method proposed by GUIPilot [46].

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 pair-wise comparison and failure type analysis in Section V-A2 and Section V-B.

IV. STUDY SETUP

A. 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 [12], GPT-4o-20240806 [10] and Claude-3.5-Sonnet-20240620 [11]. **Interaction-to-Code task takes multiple images as input, and many open source MLLMs do not support that (e.g., llava [48], llama-3.2-vision [49])**, so we select Qwen2.5-vl-instruct (3B, 7B, 72B) [50] for assessment. In configuring the MLLM models, we set the temperature to 0 and the maximum number of tokens output for Gemini-1.5-flash, GPT-4o, and Claude-3.5-Sonnect as 4096. The maximum output token for the Qwen series models is set to 2048. All other parameters were kept at their default settings as outlined in the relevant API documentation [12], [50]–[52].

B. Prompt Design

The prompts are shown in Figure 5. 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 [1] 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** [53], 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.

C. Data Contamination Check

Although the C4 dataset is widely used for training large models, it is unlikely that our test set was seen by MLLMs during training. The C4 dataset contains only text data, without HTML/CSS/JavaScript code, and current MLLM training datasets, like Qwen [50], do not include interactive prototyping data. Additionally, many existing benchmarks [22], [25] derived
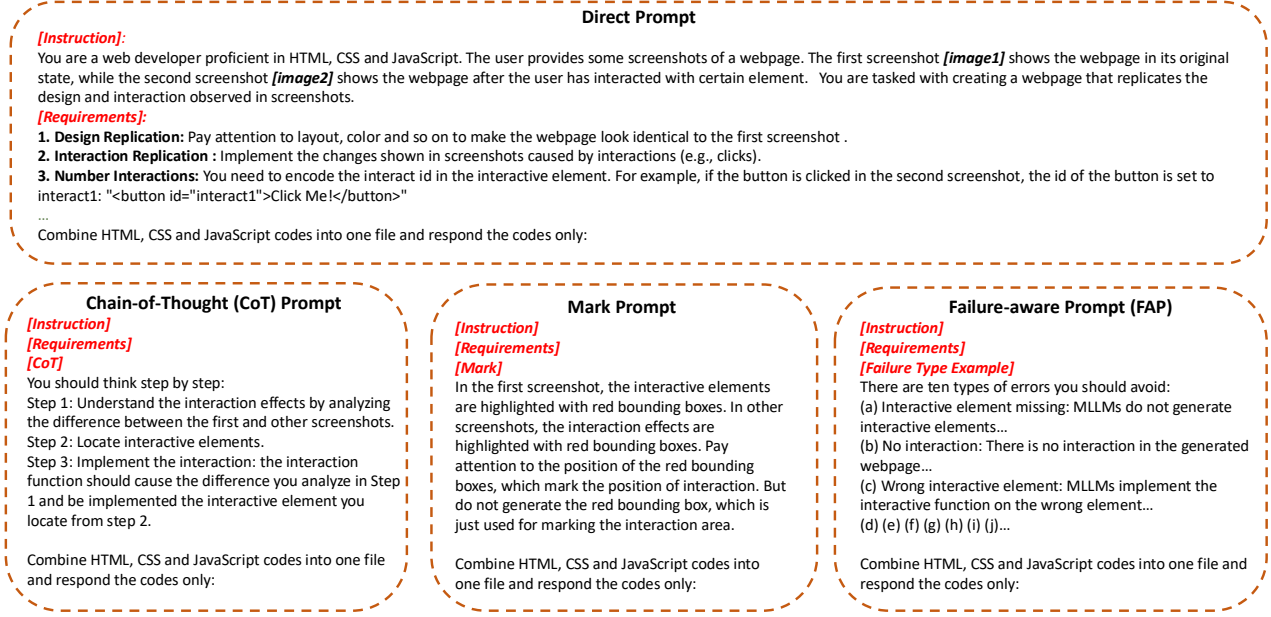
---

[1]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:

Fig. 5. The four kinds of prompts for MLLMs.

from the C4 dataset still present challenges for MLLMs. To confirm that MLLMs are not "cheating" by recognizing and reciting the original webpage code, we compute the BLEU [45] score between the generated and original webpage HTML. The very low BLEU score (Table III) indicates that the model generates the code based on the interactive prototype, not by directly copying the original webpage.

TABLE III
MODEL CODE SIMILARITY WITH ORIGINAL WEBSITE.

Model	BLEU Sim.
Qwen2.5-vl-72B	0.1274
Gemini-1.5-flash	0.1368
GPT-4o	0.1214
Claude-3.5-Sonnet	0.1294

V. EXPERIMENTS

A. Model Performance

1) Automatic Evaluation: We evaluate the performance of MLLMs on the Interaction2Code task using the metrics outlined in Section III-C. The results are presented in Table IV. Our observations are as follows: (1) GPT-4o and Claude-3.5-Sonnet outperform other models on average. (2) Among open-source models, Qwen2.5-vl-72B achieves the best performance, comparable to the commercial model Gemini-1.5-flash. Performance improves as model size increases. (3) **MLLMs perform worse on the interactive part compared to the full page (Limitation 1).** This limitation arises from the MLLMs' insufficient focus on the interactive component, motivating our proposed solution to prioritize 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 the CoT prompt [53], we design three thinking steps: analyze the interaction effects, locate the interactive elements, and implement the interaction. For the Mark prompt, we highlight the interaction area with red bounding boxes to direct MLLMs' focus on the interaction.

Both CoT and Mark prompts improve model performance compared to direct prompting, with the Mark prompt showing superior results. For Gemini-1.5-flash, the metrics (CLIP, SSIM, text, position, IR) for the interaction part improve from the direct prompt 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 the Mark prompt.

The widget similarity scores demonstrate consistently poor performance. This finding underscores fundamental limitations in the model's capacity to accurately interpret widget characteristics and hierarchical layout structures. Furthermore, analysis of GUIPilot's [46] output reveals significant omissions of web page elements. This stems from the fact that existing design consistency methodologies [46], [47] are primarily developed for mobile UI contexts, where interface images are relatively compact and uniform. In contrast, web UI images typically exhibit greater complexity and scale, thereby presenting substantial challenges to widget detection algorithms, thus designing the widget-level consistency detection algorithm

TABLE IV
Performance of different MLLMs under different prompts on Interaction-to-Code task. **Bold values** indicate the optimal performance, and <u>underlined values</u> indicate the second-best performance. The red value is the highest value among the averages. WS denotes the Widget Similarity, WMR denotes the widget match rate and the IR denote the implement rate.

| Model	Prompt	Full Page					Interaction Part				
		CLIP	SSIM	Text	WS	WMR	CLIP	SSIM	Text	Position	IR
Qwen2.5-vl-3B-instruct	Direct	**0.3220**	**0.1932**	**0.1510**	**0.0203**	**0.3462**	**0.2100**	**0.1531**	<u>0.0415</u>	**0.2090**	**0.3449**
	CoT	0.2031	0.1085	0.0800	0.0185	0.3302	0.1219	0.0894	0.0352	0.1212	0.1979
	Mark	<u>0.2752</u>	<u>0.1503</u>	<u>0.1200</u>	<u>0.0190</u>	<u>0.3418</u>	<u>0.1706</u>	<u>0.1188</u>	**0.0514**	<u>0.1706</u>	<u>0.2647</u>
	Average	0.2668	0.1507	0.1170	0.0193	0.3394	0.1675	0.1204	0.0427	0.1669	0.2692
Qwen2.5-vl-7B-instruct	Direct	<u>0.4169</u>	<u>0.2886</u>	<u>0.2519</u>	0.0222	0.3887	<u>0.3230</u>	<u>0.2177</u>	0.0952	<u>0.2529</u>	<u>0.4786</u>
	CoT	0.3895	0.2529	0.2207	**0.0286**	**0.3909**	0.2806	0.1981	0.0744	0.2259	0.4305
	Mark	**0.4586**	**0.3282**	**0.2703**	<u>0.0234</u>	0.3666	**0.3541**	**0.2468**	**0.1348**	**0.2798**	**0.5267**
	Average	0.4217	0.2899	0.2477	0.0247	0.3821	0.3192	0.2209	0.1015	0.2529	0.4786
Qwen2.5-vl-72B-instruct	Direct	<u>0.6430</u>	0.4234	0.4197	<u>0.0371</u>	0.4285	0.4624	0.3207	<u>0.2450</u>	0.3950	0.6524
	CoT	0.6335	**0.4785**	<u>0.4585</u>	0.0369	<u>0.4395</u>	**0.5090**	**0.3692**	0.2376	<u>0.4385</u>	**0.7380**
	Mark	**0.6954**	<u>0.4569</u>	**0.4586**	**0.0401**	**0.4430**	<u>0.4992</u>	<u>0.3621</u>	**0.2995**	**0.4541**	<u>0.7112</u>
	Average	0.6573	0.4529	0.4456	0.0380	0.4370	0.4902	0.3507	0.2607	0.4292	0.7005
Gemini-1.5-flash	Direct	0.5967	0.4526	0.4749	<u>0.0330</u>	**0.4964**	0.4737	0.3616	0.2809	0.4320	0.6738
	CoT	<u>0.6166</u>	<u>0.4810</u>	0.4775	**0.0332**	0.4783	<u>0.5093</u>	<u>0.3854</u>	<u>0.3217</u>	<u>0.4511</u>	<u>0.7112</u>
	Mark	**0.6321**	**0.4946**	**0.4878**	0.0322	<u>0.4826</u>	**0.5194**	**0.3898**	**0.3454**	**0.4612**	**0.7326**
	Average	0.6151	0.4761	0.4801	0.0328	0.4858	0.5008	0.3789	0.3160	0.4481	0.7059
GPT-4o	Direct	<u>0.7114</u>	<u>0.5277</u>	**0.5147**	**0.0440**	<u>0.4645</u>	<u>0.5605</u>	<u>0.4149</u>	0.3590	<u>0.4888</u>	<u>0.7754</u>
	CoT	0.6905	0.4962	0.4761	<u>0.0435</u>	**0.4674**	0.5234	0.4013	<u>0.3663</u>	0.4668	0.7273
	Mark	**0.7160**	**0.5539**	<u>0.5112</u>	0.0414	0.4418	**0.5955**	**0.4488**	**0.4474**	**0.5225**	**0.8128**
	Average	0.7059	0.5259	0.5007	0.0430	0.4579	0.5598	0.4217	0.3909	0.4927	0.7718
Claude-3.5-Sonnet	Direct	<u>0.7172</u>	**0.5318**	**0.6003**	**0.0533**	**0.5284**	<u>0.5674</u>	<u>0.4209</u>	<u>0.3833</u>	<u>0.5123</u>	<u>0.7914</u>
	CoT	0.6961	0.5110	0.5603	0.0451	0.5099	0.5606	0.4005	0.3662	0.5085	0.7727
	Mark	**0.7258**	<u>0.5299</u>	<u>0.5899</u>	<u>0.0486</u>	<u>0.5111</u>	**0.5944**	**0.4282**	**0.4319**	**0.5149**	**0.7968**
	Average	0.7130	0.5242	0.5835	0.0490	0.5165	0.5742	0.4165	0.3938	0.5119	0.7870

(a) Usability evaluation. (b) Pairwise comparision.
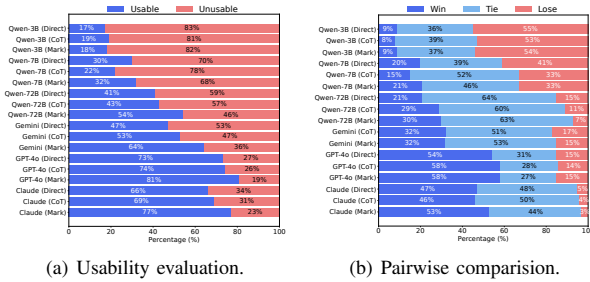
Fig. 6. Human evaluation, a higher usable rate indicates better functionality and a higher win rate indicates better quality.

suitable for webpage is necessary.

2) Human Evaluation:

a) Functionality Evaluation: In addition to automatic metrics, we conduct a functionality evaluation with four PhD students, each with three years of front-end development experience, to assess the usability of the generated interactions. An interaction is considered usable if its implementation precisely matches the behavior specified in the interactive prototype. The usability rate results are shown in Figure 6(a).

b) 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 a Tie. The results are shown in Figure 6(b), where 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 outperform 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 6(a)). (4) These human evaluation results are consistent with Section V-A, confirming the validity of our automatic evaluation metrics. Detailed instructions for human evaluations are available in our artifact.

B. Failure Type Analysis

To analyze the differences between the generated and original interactions, categorize the failure types, and assess their impact

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%
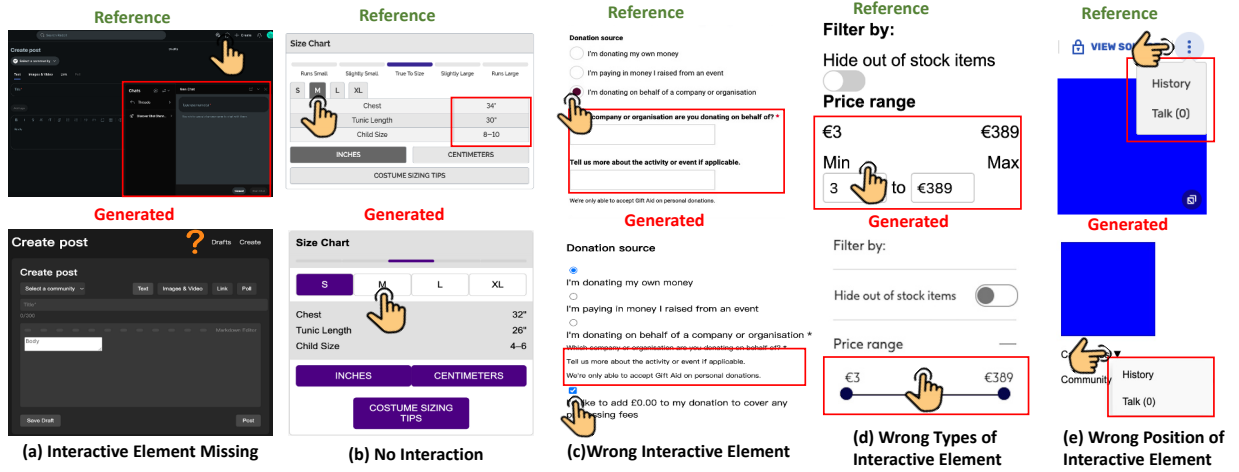
Fig. 7. Failure on interactive elements.

on content, functionality, and user experience, we employ four PhD students with three years of front-end development experience. We first randomly select 25% interactions for analysis and then discuss, revise, and refine the failure type until everyone reaches a consensus. During annotation, if the annotators encounter a new failure type, they will communicate and update the failure types in time to guide subsequent annotations (the detailed instructions are available in our artifact). The results are in Table V, which shows that **MLLMs are prone to make 10 types of failure (Limitation 2)**. Figure 7 and Figure 8 demonstrate examples of these failures, where the first row shows the reference interaction, and the second row shows the generated interaction by MLLMs. The failures are illustrated as follows:

1) Failure on Interactive Elements:

(a) Missing interactive element: MLLMs fail to generate interactive elements. For example, in Figure 7(a), the reference webpage has a chat button that opens a chat window when clicked, but the generated webpage lacks this button, preventing any interaction.

(b) No interaction: The generated webpage lacks interactive behavior. For instance, in Figure 7(b), clicking the "M" button on the reference webpage changes the displayed size, but clicking it in the generated page produces no change.

(c) Wrong interactive element: MLLMs implement the interactive function on the wrong element. For instance, in Figure 7(c), the original webpage displays input boxes after clicking "I'm donating on behalf of a company," but the generated webpage shows them only after clicking a different element.

(d) Wrong type of interactive element: MLLMs generate the wrong type of interactive element. For example, in Figure 7(d), the price adjustment element in the original page is an input field, while in the generated page, it is a progress bar.

(e) Wrong position of interactive element: MLLMs place interactive elements in the wrong position. In Figure 7(e), the button on the reference webpage is in the upper-right corner, but in the generated page, it is positioned below the image.
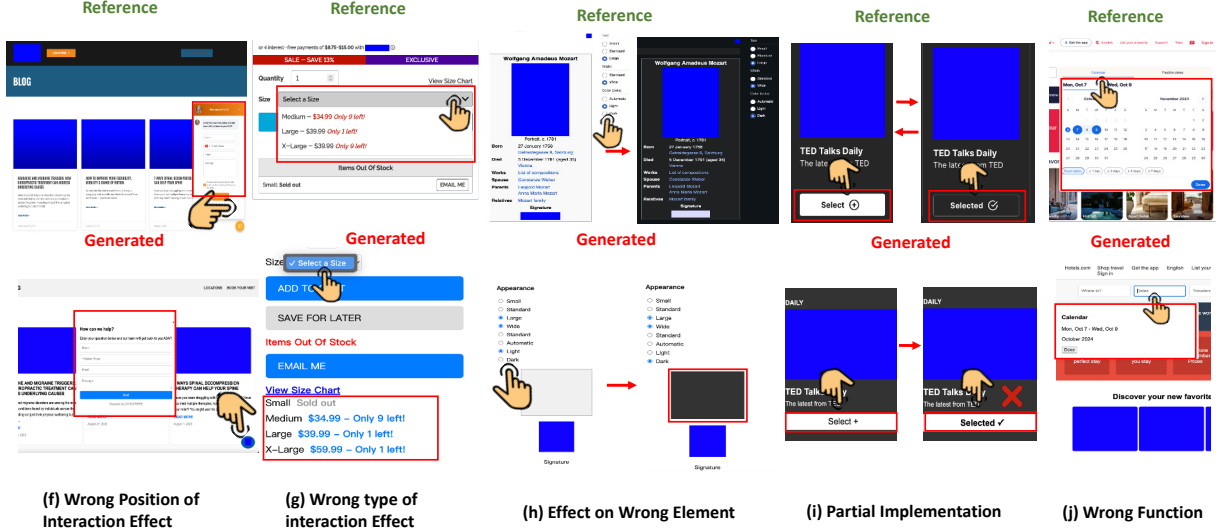
2) Failure on Interactive Effects:

247

Fig. 8. Failure of interaction effects.

(f) Wrong position after interaction: The interaction effects appear in the wrong position. For example, in Figure 8(f), the reference webpage shows a pop-up window in the lower-left corner after clicking the dialogue button, but the generated webpage displays it in the center.

(g) Wrong type of interaction effects: In Figure 8(g), the reference webpage shows an option-type element after clicking select, while the generated page shows a text-type element.

(h) Effect on wrong element: MLLMs apply the interaction effect to the wrong element. As seen in Figure 8(h), in the reference webpage, clicking the "dark" button changes the background color to black, but in the generated page, only a block turns black, leaving the background unchanged.

(i) Partial Implementation: MLLMs only implement part of the interactive functionality. For instance, in Figure 8(i), the reference webpage allows a button to be selected and unselected, but the generated webpage only allows the button to be selected, not unselected.

(j) Wrong function: MLLMs implement the wrong function. As shown in Figure 8(j), in the original webpage, clicking the button opens a date selection box, but in the generated webpage, it opens a date display box instead.

Failure reason analysis. Specifically, we cluster 10 failure types into three main symptoms that all related to the above limitation:(1) Spatial failures (Failures a, c, e, f) – the model cannot accurately locate the element linked to the interaction. (2) Type failures (Failures d, g) – it misclassifies the widget type (e.g., slider vs. input), resulting in incorrect HTML elements in the output code. (3) Behavioral failures (Failures b, h, i, j) – it fails to infer the correct state-change logic (e.g., toggle vs. open-modal), leading to missing or incorrect code implementation.

Based on the failure distribution in Figure 9, we find that, **the main failure modes are "No interaction", "Partial im-**

plementation", "Interactive element missing", and "Wrong function".**

Additionally, the most critical failures are "Interactive element missing", "Wrong function", "No interaction" and "Effect on wrong element". The severity of these failures is determined by the usability rate (UR), where a higher UR indicates lower severity, and a lower UR indicates higher severity. As illustrated in Table V, failures (a), (b), and (j) exhibit a 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 (as shown in Figure 5). We use $\frac{2}{3}$ of the dataset to annotate failure types and $\frac{1}{3}$ of the dataset to test. Table VI 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.**

C. 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 [54]. 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 indicates a
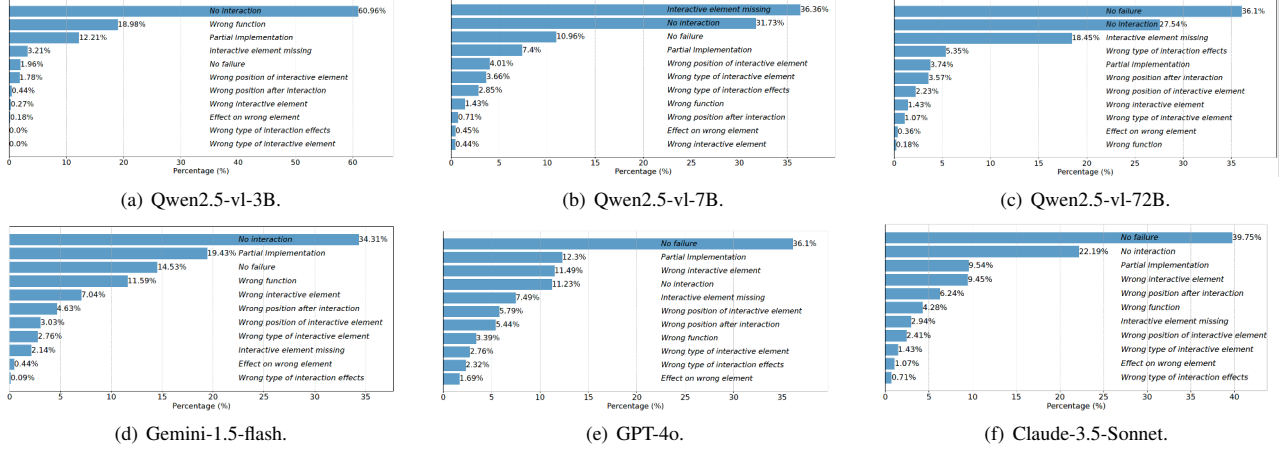
(a) Qwen2.5-vl-3B.

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

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

(d) Gemini-1.5-flash.

(e) GPT-4o.

(f) Claude-3.5-Sonnet.

Fig. 9. Failure distribution of MLLMs.

TABLE VI
COMPARISON BETWEEN DIRECT PROMPT AND FAP.

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

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 10(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 10 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 10(b), Figure 10(c) shows that the SSIM and position similarity significantly increases

as the visual saliency increases. As shown in Figure 10(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.

> **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.

We then randomly sample 10 webpages from failure cases and crop the screenshots to increase the visual saliency of
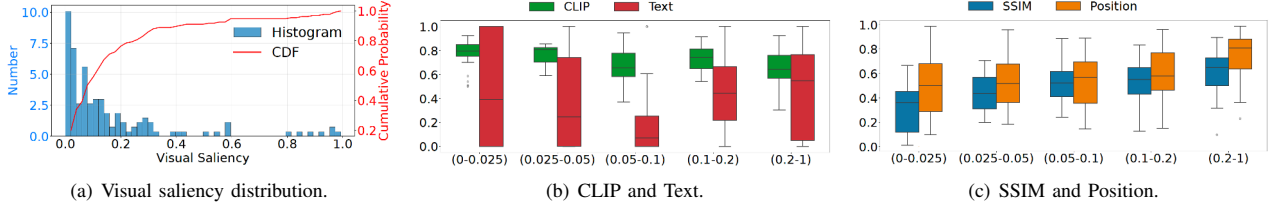
Fig. 10. Visual saliency and interaction part metrics distribution of different groups of Gemini-1.5-flash.

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 11 shows the relationship between the magnification factor and the metrics 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 overcome the original failures.
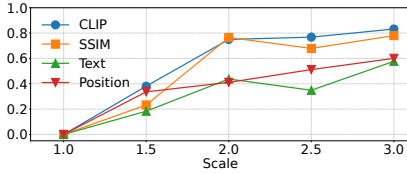
Fig. 11. Metrics under different magnification.

Although increasing magnification may result in some UI areas being cut off, this can be effectively addressed through a two-step generation approach:(1) Step 1: Generate the interactive components within the magnified, focused area. (2) Step 2: Generate the remaining cut-off UI portions separately. (3) Step 3: Combine both parts to produce the complete webpage.

Using the above method, we sampled 10 webpages with magnification factors ranging from 1x to 3x and observe the performance. Table VII shows that full-page similarity scores remain consistent. So we believe **the models' performance on the full UI page remains stable with the increased magnification factor.**

TABLE VII
THE FULL PAGE PERFORMANCE UNDER DIFFERENT MAGNIFICATION

Magnification	1	1.5	2	2.5	3
CLIP	0.826	0.842	0.838	0.843	0.836
SSIM	0.652	0.657	0.643	0.654	0.653
Text	0.657	0.659	0.649	0.665	0.659

TABLE VIII
PERFORMANCE OF GPT-4O WITH DIFFERENT MODALITY INPUTS. **BOLD VALUES** ARE THE BEST PERFORMANCE AND <u>UNDERLINED VALUES</u> ARE THE SECOND-BEST PERFORMANCE.

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

D. The Impact of Different Modalities

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 VIII 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.

E. Ablation Study

To investigate whether the improvement methods address complementary weaknesses or have overlapping benefits, we conducte the ablation experiments on GPT4o, which evaluates the combinations of all proposed enhancement strategies. The results in Table IX show that combining Interactive Element Highlighting (IEH), Failure-aware Prompt (FAP), Visual Saliency Enhancement (VSE) and Textual Description

(TD) can achieve the best performance, and removing any one of them leads to a decrease in performance (✗ denotes "without the module", ✓ indicates "with the module".)

TABLE IX
COMBINATIONS OF PROPOSED IMPROVEMENTS AND CORRESPODING PERFORMANCE.

IEH	FAP	VSE	TD	CLIP	SSIM	Text	Position
✗	✗	✗	✗	0.5700	0.4891	0.3652	0.5802
✓	✗	✗	✗	0.5968	0.5456	0.4508	0.6408
✗	✓	✗	✗	0.6072	0.5405	0.4580	0.6452
✗	✗	✓	✗	0.6328	0.5723	0.4629	0.6874
✗	✗	✗	✓	0.6547	0.5681	0.4732	0.7024
✗	✓	✓	✓	0.6584	0.5938	0.4852	0.7027
✓	✗	✓	✓	0.6627	0.5987	0.4764	0.7128
✓	✓	✗	✓	0.6737	0.6136	0.4828	0.7269
✓	✓	✓	✗	0.6532	0.5823	0.4621	0.6987
✓	✓	✓	✓	**0.6937**	**0.6251**	**0.5187**	**0.7371**

F. Human Evaluation of Interaction2Code Tool

To demonstrate the practical use of the Interaction2Code paradigm, we conduct a user study to asses the impact of our task on dynamic website developers.

Participant and preparation. We hired four PhD students with similar front-end development experience to complete four interaction tasks within 30 mins. Of these participants, two utilized our Interaction2Code tool (described in Appendix H), while the other two used LLMs but did not access the Interaction2Code tool.

Study setting. We recorded the time taken to complete tasks. We also invite other two front-end development experts evaluate their implementations on a scale of 1 to 5, with higher scores indicating better implementations.

Result. Table X presents the time costs and performance metrics. Our results demonstrate substantial improvements in both development efficiency (over 32.4%) and implementation quality (over 16.1%) when using the Interaction2Code Tool. Our user studies demonstrate that Interaction2Code task provides substantial assistance for dynamic webpage developers.

Speed-up Mechanism. Interaction2Code accelerates development by shifting the workflow from "generate-from-scratch" to "review-and-refine": (1) Ready-to-use draft: the tool emits a fully functional webpage that already contains the correct HTML structure, CSS layout, and JavaScript event handlers for the specified interaction. The developer do not need to write code from scratch. (2) Reduced lookup time: developers no longer spend minutes searching docs for the api usage like "onClick" signature, event attributes, etc. (3) Faster iteration loop: developers skip writing many lines of codes and jump straight to testing the generated interaction: either approve it or apply minor adjustments.

VI. THREATS TO VALIDITY

Limited context length. As webpages become more complex with numerous interactions, the input context expands, potentially exceeding the context window constraints of MLLMs (e.g., 128K tokens for GPT-4o). Nevertheless, this limitation can

TABLE X
TASK TIME AND SIMILARITY WITH AND WITHOUT INTERACTION2CODE.
W. DENOTES RESULTS WITH INTERACTION2CODE TOOL, W/O DENOTES RESULTS WITHOUT THE TOOL.

Task	Time (w.)	Time (w/o)	Similarity (w.)	Similarity (w/o)
Task 1	323s	487s	5	4
Task 2	182s	319s	4	4
Task 3	125s	189s	5	4
Task 4	486s	657s	4	3.5
Average	**279s**	**413s**	**4.5**	**3.875**

be mitigated by employing iterative generation, progressively producing interactions for a webpage over multiple rounds.

Model selection. This study utilizes three prominent Multimodal Large Language Models (MLLMs) to conduct experiments. There are some MLLMs such as Pixtral [55] we don't test, we will test the performance of these models on Interaction-to-Code task in the future work.

Unable to handle interactions that require back-end. Some complex functional interactions (e.g., login, search, etc.) are implemented by server-side scripting languages like Python. The benchmark we collect does not include back-end code; we cannot verify the generation effect of such interactions, but we believe our work is an important step toward generating interactive websites.

VII. 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.

VIII. ACKNOWLEDGMENT

REFERENCES

[1] "Top website statistics for 2024," *Forbes Advisor*, 2024, accessed: 2024-10-06. [Online]. Available: https://www.forbes.com/advisor/business/software/website-statistics/

[2] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "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*, 2018, pp. 665–676.

[3] K. Kuusinen and T. Mikkonen, "Designing user experience for mobile apps: Long-term product owner perspective," *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, vol. 1, pp. 535–540, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:18632493

[4] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with remaui (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 248–259.

[5] V. Lelli, A. Blouin, and B. Baudry, "Classifying and qualifying gui defects," *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 1–10, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:2288032

[6] K. Moran, B. Li, C. Bernal-Cárdenas, D. Jelf, and D. Poshyvanyk, "Automated reporting of gui design violations for mobile apps," *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 165–175, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:3634687

[7] C. Zeidler, C. Lutteroth, W. Stuerzlinger, and G. Weber, "Evaluating direct manipulation operations for constraint-based layout," in *IFIP TC13 International Conference on Human-Computer Interaction*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:8243987

[8] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in *Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems*, 2018, pp. 1–6.

[9] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "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 (ICSE)*, 2018, pp. 665–676.

[10] OpenAI, "Hello gpt-4o," 2024, accessed: 2024-10-06. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[11] Anthropic, "Introducing claude 3.5 sonnet," 2024, accessed: 2024-09-29. [Online]. Available: https://www.anthropic.com/news/claude-3-5-sonnet

[12] Google, "Gemini api," 2024, accessed: 2024-10-06. [Online]. Available: https://ai.google.dev/gemini-api

[13] Z. Yang, L. Li, K. Lin, J. Wang, C.-C. Lin, Z. Liu, and L. Wang, "The dawn of lmms: Preliminary explorations with gpt-4v(ision)," *ArXiv*, vol. abs/2309.17421, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263310951

[14] W. Dai, J. Li, D. Li, A. M. H. Tiong, J. Zhao, W. Wang, B. A. Li, P. Fung, and S. C. H. Hoi, "Instructblip: Towards general-purpose vision-language models with instruction tuning," *ArXiv*, vol. abs/2305.06500, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258615266

[15] J. Liu, J. Xiao, W. Tang, W. Wang, Z. Wang, M. Zhang, and S. Yu, "Benchmarking mllm-based web understanding: Reasoning, robustness and safety," 2025. [Online]. Available: https://arxiv.org/abs/2509.21782

[16] H. Yu, B. Shen, D. Ran, J. Zhang, Q. Zhang, Y. Ma, G. Liang, Y. Li, T. Xie, and Q. Wang, "Codereval: A benchmark of pragmatic code generation with generative pre-trained models," in *International Conference on Software Engineering (ICSE)*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:256459413

[17] J. Li, G. Li, Y. Li, and Z. Jin, "Enabling programming thinking in large language models toward code generation," *ArXiv*, vol. abs/2305.06599, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263896057

[18] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation," *ArXiv*, vol. abs/2308.01861, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260439062

[19] R. Lu, Y. Li, and Y. Huo, "Exploring autonomous agents: A closer look at why they fail when completing tasks," *arXiv preprint arXiv:2508.13143*, 2025.

[20] R. Lu, Y. Huo, M. Zhang, Y. Li, and M. R. Lyu, "Next edit prediction: Learning to predict code edits from context and interaction history," *arXiv preprint arXiv:2508.10074*, 2025.

[21] W. Tang, J. Xiao, W. Jiang, X. Xiao, Y. Wang, X. Tang, Q. Li, Y. Ma, J. Liu, S. Tang *et al.*, "Slidecoder: Layout-aware rag-enhanced hierarchical slide generation from design," *arXiv preprint arXiv:2506.07964*, 2025.

[22] C. Si, Y. Zhang, Z. Yang, R. Liu, and D. Yang, "Design2code: How far are we from automating front-end engineering?" *arXiv preprint arXiv:2403.03163*, 2024.

[23] Y. Wan, C. Wang, Y. Dong, W. Wang, S. Li, Y. Huo, and M. R. Lyu, "Automatically generating ui code from screenshot: A divide-and-conquer-based approach," *Proceedings of the ACM on Software Engineering (FSE)*, vol. 1, no. FSE, pp. 1–24, 2025.

[24] S. Yun, H. Lin, R. Thushara, M. Q. Bhat, Y. Wang, Z. Jiang, M. Deng, J. Wang, T. Tao, J. Li *et al.*, "Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms," *arXiv preprint arXiv:2406.20098*, 2024.

[25] Y. Gui, Z. Li, Y. Wan, Y. Shi, H. Zhang, B. Chen, Y. Su, D. Chen, S. Wu, X. Zhou *et al.*, "Webcode2m: A real-world dataset for code generation from webpage designs," in *Proceedings of the ACM on Web Conference (WWW)*, 2025, pp. 1834–1845.

[26] J. Xiao, M. Wang, M. H. Lam, Y. Wan, J. Liu, Y. Huo, and M. R. Lyu, "Designbench: A comprehensive benchmark for mllm-based front-end code generation," *arXiv preprint arXiv:2506.06251*, 2025.

[27] Y. Wan, Y. Dong, J. Xiao, Y. Huo, W. Wang, and M. R. Lyu, "Mrweb: An exploration of generating multi-page resource-aware web code from ui designs," *arXiv preprint arXiv:2412.15310*, 2024.

[28] B. Aşıroğlu, B. R. Mete, E. Yıldız, Y. Nalçakan, A. Sezen, M. Dağtekin, and T. Ensari, "Automatic html code generation from mock-up images using machine learning techniques," in *2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT)*. Ieee, 2019, pp. 1–4.

[29] A. A. J. Cizotto, R. C. T. de Souza, V. C. Mariani, and L. dos Santos Coelho, "Web pages from mockup design based on convolutional neural network and class activation mapping," *Multimedia Tools and Applications*, vol. 82, no. 25, pp. 38 771–38 797, 2023.

[30] K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," *IEEE Transactions on Software Engineering (TSE)*, vol. 46, no. 2, pp. 196–221, 2018.

[31] Y. Xu, L. Bo, X. Sun, B. Li, J. Jiang, and W. Zhou, "image2emmet: Automatic code generation from web user interface image," *Journal of Software: Evolution and Process*, vol. 33, no. 8, p. e2369, 2021.

[32] W.-Y. Chen, P. Podstreleny, W.-H. Cheng, Y.-Y. Chen, and K.-L. Hua, "Code generation from a graphical user interface via attention-based encoder–decoder model," *Multimedia Systems*, vol. 28, no. 1, pp. 121–130, 2022.

[33] V. Jain, P. Agrawal, S. Banga, R. Kapoor, and S. Gulyani, "Sketch2code: transformation of sketches to ui in real-time using deep neural network," *arXiv preprint arXiv:1910.08930*, 2019.

[34] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with remaui (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 248–259.

[35] T. Zhou, Y. Zhao, X. Hou, X. Sun, K. Chen, and H. Wang, "Bridging design and development with automated declarative ui code generation," *Proceedings of the ACM on Software Engineering (FSE)*, vol. 1, no. FSE, pp. 1–24, 2025.

[36] J. Xiao, Z. Zhang, Y. Wan, Y. Huo, Y. Liu, and M. R. Lyu, "Efficientuicoder: Efficient mllm-based ui code generation via input and output token compression," *arXiv preprint arXiv:2509.12159*, 2025.

[37] H. Laurençon, L. Tronchon, and V. Sanh, "Unlocking the conversion of web screenshots into html code with the websight dataset," 2024.

[38] UI-Mockup, "Ui mockups," January 2025. [Online]. Available: https://www.uxpin.com/studio/blog/what-is-a-mockup-the-final-layer-of-ui-design/

[39] Interactive, "Ui mockups," January 2025. [Online]. Available: https://www.uxpin.com/studio/blog/interactive-prototype-setting-user-interactions-without-coding/

[40] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[41] Selenium, "Selenium," January 2025. [Online]. Available: https://selenium-python.readthedocs.io/

[42] Git, "Git difference tool," January 2025. [Online]. Available: https://git-scm.com/docs/git-difftool

[43] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable

visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[44] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[45] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics (ACL)*, 2002, pp. 311–318.

[46] R. Liu, X. Teoh, Y. Lin, G. Chen, R. Ren, D. Poshyvanyk, and J. S. Dong, "Guipilot: A consistency-based mobile gui testing approach for detecting application-specific bugs," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3728909

[47] K. Moran, B. Li, C. Bernal-Cárdenas, D. Jelf, and D. Poshyvanyk, "Automated reporting of gui design violations for mobile apps," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. New York, NY, USA: Association for Computing Machinery, 2018, p. 165–175. [Online]. Available: https://doi.org/10.1145/3180155.3180246

[48] H. Liu, C. Li, Y. Li, B. Li, Y. Zhang, S. Shen, and Y. J. Lee, "Llava-next: Improved reasoning, ocr, and world knowledge," January 2024. [Online]. Available: https://llava-vl.github.io/blog/2024-01-30-llava-next/

[49] Meta, "Llama 3.2 vision," 2024, accessed: 2025-02-06. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-90B-Vision-Ins truct

[50] Qwen, "Qwen2.5-vl," January 2025. [Online]. Available: https: //qwenlm.github.io/blog/qwen2.5-vl/

[51] OpenAI, "Vision guide," 2024, accessed: 2024-10-18. [Online]. Available: https://platform.openai.com/docs/guides/vision

[52] Anthropic, "Vision documentation," 2024, accessed: 2024-10-18. [Online]. Available: https://docs.anthropic.com/en/docs/vision

[53] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems (NeurIPS)*, vol. 35, pp. 24 824–24 837, 2022.

[54] J. Zhang, J. Hu, M. Khayatkhoei, F. Ilievski, and M. Sun, "Exploring perceptual limitation of multimodal large language models," *arXiv preprint arXiv:2402.07384*, 2024.

[55] P. Agrawal, S. Antoniak, E. B. Hanna, B. Bout, D. Chaplot, J. Chud-novsky, D. Costa, B. De Monicault, S. Garg, T. Gervet *et al.*, "Pixtral 12b," *arXiv preprint arXiv:2410.07073*, 2024.