WebGen-Bench: Evaluating LLMs on Generating Interactive and Functional Websites from Scratch

Zimu Lu[1*], **Yunqiao Yang**[1*], **Houxing Ren**[1*†], **Haotian Hou**[2], **Han Xiao**[1], **Ke Wang**[1]
Weikang Shi[1], **Aojun Zhou**[1], **Mingjie Zhan**[2‡], **Hongsheng Li**[1,3,4‡]
[1]Multimedia Laboratory (MMLab), The Chinese University of Hong Kong,
[2]SenseTime, [3]CPII under InnoHK, [4]Shanghai AI Laboratory
luzimu@link.cuhk.edu.hk zhanmingjie@sensetime.com hsli@ee.cuhk.edu.hk

Abstract

LLM-based agents have demonstrated great potential in generating and managing code within complex codebases. In this paper, we introduce WebGen-Bench, a novel benchmark designed to measure an LLM-based agent's ability to create multi-file website codebases from scratch. It contains diverse instructions for website generation, created through the combined efforts of human annotators and GPT-4o. These instructions span three major categories and thirteen minor categories, encompassing nearly all important types of web applications. To assess the quality of the generated websites, we generate test cases targeting each functionality described in the instructions. These test cases are then manually filtered, refined, and organized to ensure accuracy, resulting in a total of 647 test cases. Each test case specifies an operation to be performed on the website and the expected outcome of the operation. To automate testing and improve reproducibility, we employ a powerful web-navigation agent to execute test cases on the generated websites and determine whether the observed responses align with the expected results. We evaluate three high-performance code-agent frameworks—Bolt.diy, OpenHands, and Aider—using multiple proprietary and open-source LLMs as engines. The best-performing combination, Bolt.diy powered by DeepSeek-R1, achieves only 27.8% accuracy on the test cases, highlighting the challenging nature of our benchmark. Additionally, we construct WebGen-Instruct, a training set consisting of 6,667 website-generation instructions. Training Qwen2.5-Coder-32B-Instruct on Bolt.diy trajectories generated from a subset of the training set achieves an accuracy of 38.2%, surpassing the performance of the best proprietary model. We release our data-generation, training, and testing code, along with both the datasets and model weights at `https://github.com/mnluzimu/WebGen-Bench`.

1 Introduction

Recent developments in large language models (LLMs) have demonstrated increasingly strong performance. When paired with agent frameworks, they have become much more competent at solving challenging tasks such as fixing bugs in complex codebases and competing in coding competitions. Prior works have sought to quantify the software engineering abilities of these LLM-powered agents by testing them on curated GitHub issues [26, 56] and feature-patching requests [35]. These tasks mainly involve modifications to existing codebases and primarily target expert engineers.

[*]Equal contribution.
[†]Project lead.
[‡]Corresponding author.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Track on Datasets and Benchmarks.

Table 1: Comparison of WebGen-Bench with other repository-level software engineering benchmarks. The "Number of Files" and "Lines of Code" represent the average values per sample (*except for SWE-Bench Multimodal, where the values are medians). The values for our benchmarks are derived from the test results of Bolt.diy, OpenHands, and Aider using DeepSeek-V3. The values for the other benchmarks are obtained from [26],[35], and[56], respectively.

Benchmark	From Scratch	Training Set	Number of Files	Lines of Code
WebGen-Bench (ours)	✓	✓	8.1	315.3
SWE-Bench	×	✓	1.7	32.8
SWE-Bench Multimodal*	×	×	2	27
SWE-Lancer	×	×	2	55

On the other hand, there is a growing need for code agents to assist non-experts with little or no programming background in building applications tailored to their needs and expectations. For example, Bolt.new[4] and Lovable.dev[5] are two projects that generate complete websites based on user requests and have become popular among customers. This task poses significant challenges for LLM-based agents, as building a fully functional and customized web application from scratch tests a wide range of capabilities—including high-level planning, organizing complex multi-file codebases, and implementing nuanced user requirements. However, there is currently a lack of systematic and reliable evaluation methods for this task. The high demand for such applications, coupled with the value of assessing agent capabilities, highlights the need for a novel benchmark to evaluate the ability to generate websites from scratch based on natural language instructions.

To this end, we introduce **WebGen-Bench**, the first benchmark to systematically evaluate LLM-based agents' ability to construct websites that satisfy the functional and appearance requirements specified in user instructions. As shown in Tab. 1, unlike prior software-engineering benchmarks [26, 35, 56], which focus on fixing bugs or supplying patches to existing codebases, our benchmark requires models to build a complex codebase from scratch, assessing agents' ability to plan, develop, and manage projects with multi-file structures. There are two critical challenges to address when creating the benchmark: (1) how to curate diverse instructions covering major web-application categories and (2) how to accurately evaluate the websites generated from scratch.

To tackle these problems, we introduce a systematic data curation and evaluation pipeline for assessing website-generation agents. Starting from 20 common categories identified across popular development platforms, we first manually construct diverse website descriptions and then use GPT-4o to generate comprehensive instructions and test cases that cover both functionality and appearance. For evaluation, we leverage WebVoyager for automated functional testing and prompt GPT-4o to rate design aesthetics on a scale of 1 to 5. Using this framework, we benchmark Bolt.diy, OpenHands, and Aider, and find that Bolt.diy performs best. Further evaluation across models shows that DeepSeek-R1 achieves the highest functional success rate (27.8%), whereas Claude-3.5-Sonnet leads in appearance with an average score of 3.0 out of 5.0, indicating substantial room for improvement.

We also construct a training dataset named **WebGen-Instruct**, which contains 6,667 website-generation instructions. To avoid data contamination, we removed instructions that are semantically similar to those in WebGen-Bench by applying Jaccard-similarity filtering and Sentence-Transformers–based deduplication [40], as detailed in Appendix D. Fine-tuning Qwen2.5-Coder-32B-Instruct on Bolt.diy trajectories—generated from a subset of WebGen-Instruct by DeepSeek-V3 with rejection sampling raises its accuracy to 38.2%, a substantial improvement over its original 9.5% and even higher than the performance of DeepSeek-R1. We also fine-tune Qwen2.5-Coder-7B-Instruct and Qwen2.5-Coder-14B-Instruct on the same training data, and name the resulting family of website-generation models **WebGen-LM**.

Our contributions are as follows:

- We introduce WebGen-Bench, the first benchmark designed to test the ability of an LLM-based agent to generate websites from scratch. It includes diverse instructions for website generation and corresponding test cases to evaluate website functionalities.

[4]https://bolt.new
[5]https://lovable.dev

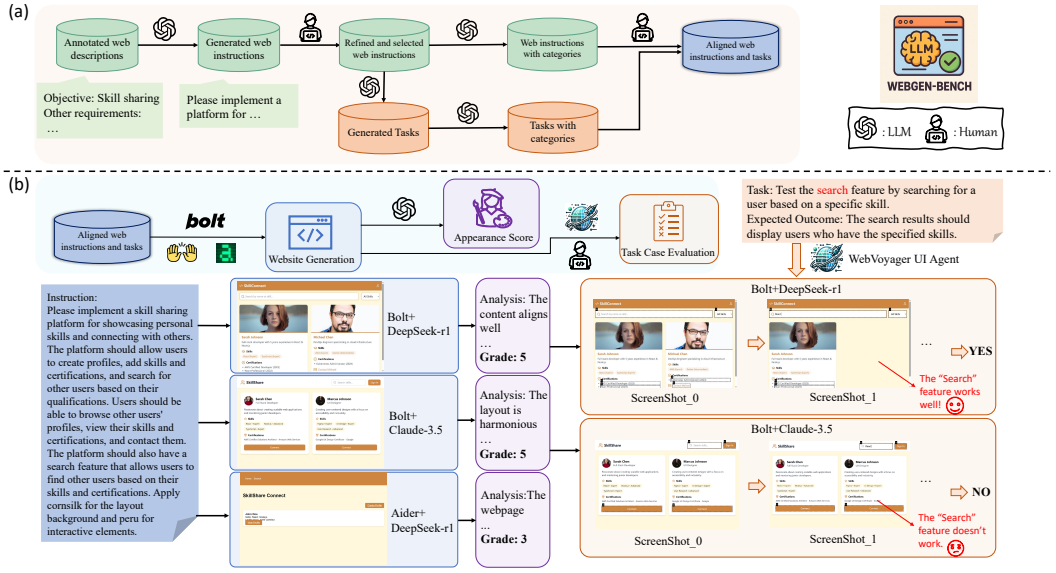


Figure 1: The data-curation and automatic-testing pipeline. (a) shows the process for curating website-generation instructions together with their corresponding test cases; (b) demonstrates the testing pipeline for verifying whether the generated websites satisfy functionality and design requirements with the WebVoyager UI agent. Their aesthetic quality is evaluated using GPT-4o.

- We conduct comprehensive evaluations of three high-performance code-agent frameworks — Bolt.diy, OpenHands, and Aider — using different proprietary LLMs as engines, demonstrating the challenging nature of our benchmark.
- We construct WebGen-Instruct, a training set consisting of 6,667 website-generation instructions. We use this training set to fine-tune Qwen2.5-Coder-Instruct models of sizes 7B, 14B, and 32B, resulting in a family of LLMs specialized in website generation, named WebGen-LM. WebGen-LM-32B achieves an accuracy of 38.2% on WebGen-Bench, surpassing DeepSeek-R1.

2 Related Work

Software Engineering Benchmarks. Code generation has long been used as a means to evaluate the abilities of LLMs [16, 6, 4]. Previous works have collected coding problems from various sources, such as user queries [63], coding contests [20], model synthesis [65], and expert design [36, 6], to evaluate LLMs' performance on single-file, function-level coding tasks. Recently, as stronger models have reached a plateau on these simpler benchmarks, new benchmarks such as SWE-bench [26, 56] and SWE-Lancer [35] have been constructed by collecting real-world code repositories and corresponding issue requests to test models' ability to solve bugs and implement new functionalities. These benchmarks require models to identify and fix issues [26, 56, 2], perform code completions [29, 60], or provide functionality patches [35] within an existing multi-file codebase. Different from previous works, our benchmark focuses on creating web applications from scratch based on natural language instructions, requiring models to generate a complex, multi-file codebase, implement multiple functionality and appearance requirements, and make independent technical design decisions.

LLM-based Code Agents and Pipelines. Various agent-based [50, 55, 1, 9, 8, 51] and pipeline-based [52, 44, 64] methods have been proposed to address software engineering problems such as code completion and GitHub issue resolution. While pipeline-based methods sometimes demonstrate strong performance on specific tasks with fixed pipelines [26], agent-based methods are generally more flexible. Code agent frameworks such as OpenHands [50] and SWE-agent [55] interact with executable environments to obtain feedback from the execution of generated code. To evaluate our benchmark, we selected three open-source code agents. Among them, OpenHands [50] and Aider [1] are general-purpose code agent frameworks that we adapted for our benchmark, while Bolt.diy [47] is a specialized framework for generating web applications. Prior works [38, 34, 32, 53, 33, 21] have

employed various post-training methods to improve the performance of open-source models. In this work, we also fine-tune open-source models with generated trajectories.

Automatic Software User-testing. User-testing is a common method in software engineering to assess the functionality of software with high user-interaction requirements. However, human testing can be costly and introduce significant management complexities. Various works have employed agents to test websites [30], graphical user interfaces (GUIs) [10], and games [48, 11]. Similar to our work, [19, 5, 27] use unit tests to enable automated testing of the outputs generated by LLMs, though their tests are assertion-based while ours are based on GUI Agent operation results on the generated LLMs. Among them, UXAgent [30] uses UI agents with pre-defined personas to simulate user experiences on websites. Our work also utilizes a web navigation UI agent to evaluate generated websites. Different from prior works, we define atomic test cases targeting functionality and appearance requirements, enabling the agent to perform operations and observe whether the website behaves as intended.

3 WebGen-Bench

In this section, we introduce WebGen-Bench, the first benchmark designed to test the ability of LLM-based agents to generate websites from scratch based on natural language instructions. The benchmark consists of diverse website-generation instructions and comprehensive test cases that have been carefully constructed and repeatedly validated. A reliable and cost-effective testing pipeline, built around a strong web navigation agent, has been developed to ensure efficient evaluation of the generated websites. The data curation process and testing pipeline are shown in Fig. 1 (a) and (b) respectively.

3.1 Instruction Curation

Web Development Project Descriptions Collection. To ensure the diversity and practicality of the instructions, we first carefully browsed several platforms containing website development project listings, including Upwork[6], Freelancer[7], and Proginn[8]. We identified twenty prevalent web application categories, as outlined in Appendix E. To simulate numerous customized web applications, we employ a panel of forty computer science student volunteers to conduct brainstorming sessions to determine various specific web applications belonging to these categories, as well as a brief and clear list of corresponding functionality and appearance requirements for each application. A customized application and its corresponding requirements are combined into a project description. We manually created 10152 project descriptions in total.

Website-Generation Instruction Curation. From the collected project descriptions, we use one-shot prompting with GPT-4o to generate the corresponding instructions. The prompt template is shown in Appendix C. Because the total number of generated instructions exceeds the practical limits of benchmarking code agents—which require substantial computational resources and long inference trajectories—we sample 2 to 8 representative examples from each category to preserve both coverage and diversity. This procedure produces a curated test set containing 101 instructions.

Next, we decontaminate the remaining instructions by first filtering those with a 5-gram Jaccard similarity score exceeding 0.6 relative to any testing instruction. We then perform semantic deduplication by computing cosine similarity between sentence embeddings [40] of the remaining instructions and the testing set. This process produces a training set of 6,667 website-generation instructions, which we name **WebGen-Instruct**. Details of the decontamination process are provided in Appendix D.

Test Set Adjustment and Validation. We refine and validate the selected test instructions to ensure they exclude unreasonable designs and overly specific technical details. We intentionally omit technical design specifications because our dataset aims to evaluate code agents in scenarios where they receive instructions from non-expert users. The agents should autonomously determine the optimal technical approach. Including tool-specific hints in the instructions would compromise this objective.

[6] https://www.upwork.com

[7] https://www.freelancer.com

[8] https://www.proginn.com

Table 2: The number of website-generation instructions in each technical category in WebGen-Bench. Each main category contains multiple subcategories. A sample may belong to one main category and multiple subcategories.

Main Categories	Sample Number	Sub Category	Sample Number
Content Presentation	28	Static Page Generation	20
		Dynamic Content Rendering	18
		Data Visualization	36
		Media Display	6
User Interaction	49	Form Systems	40
		Authentication	18
		Real-time Features	20
		E-commerce	22
		AI Integration	19
Data Management	24	CRUD Operations	29
		API Integration	20
		Big Data	12
		File Handling	5
Total	101		

Technical Classification of the Testing Set. Given the limited number of testing instructions per application category, analyzing categorical statistics based on the original 20 application categories would be confusing. To enable higher-level analysis, we reorganize the 101 testing instructions into three broader technical categories (see Tab. 2): (1) Content Presentation: Static page generation (e.g., corporate/portfolio sites), dynamic rendering (e.g., blogs/news feeds), data visualization (e.g., dashboards), and immersive media displays (e.g., 360° product views). (2) User Interaction: Form systems, authentication flows, real-time collaboration tools, e-commerce transactions, and AI-enhanced features (e.g., chatbots). (3) Data Management: CRUD operations for content administration, third-party API integrations (e.g., payment/social platforms), analytical processing of user behavior data, and file operations (e.g., cloud synchronization, bulk exports).

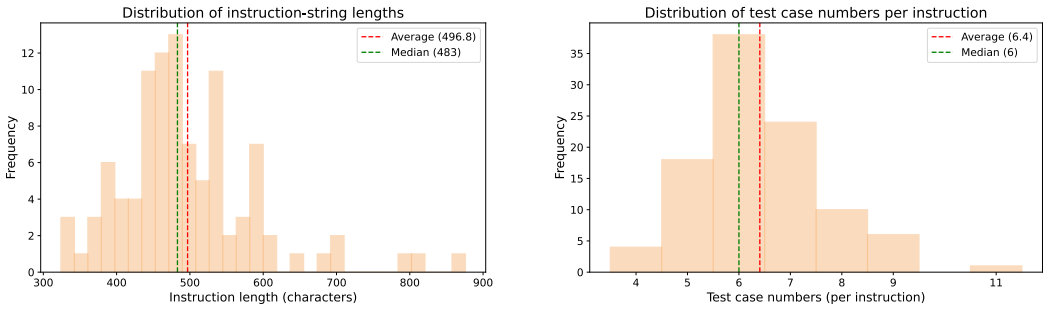
3.2 Test Case Construction and Evaluation

Since the websites are generated from scratch based on the instructions, the tested agents have significant freedom in their implementation choices. To accurately evaluate how well the agents satisfy the instruction requirements while accommodating diverse implementation approaches, we construct test cases targeting each and every requirement in the instructions.

Test Case Construction. Each test case consists of an operation verifying a specific functionality or appearance requirement, paired with its expected outcome. We first generate draft test cases using GPT-4o with the prompt shown in Appendix F. Two computer science Ph.D. students then independently review and refine these test cases. After comparing their adjustments, we resolved discrepancies through discussion, yielding a final set of 647 test cases (4–11 per instruction). This manual validation process guarantees strict alignment between test cases and instructions, ensuring: (1) all instruction requirements are covered by test cases, and (2) each test case corresponds to an instruction requirement. This approach ensures comprehensive evaluation while preserving implementation flexibility for the tested agents.

UI Agent-based Evaluation. With instructions and test cases prepared, we must determine how to effectively evaluate the generated websites. Manual testing by human evaluators is costly and time-consuming, as completing a test case takes at least 60 seconds, and finishing all 647 test cases would require approximately 10.8 hours at an estimated cost of $377.8 [49]. This slow, labor-intensive process would hinder rapid iteration during framework development, preventing researchers from obtaining timely feedback when refining website-generation systems.

To improve testing efficiency, we automate test case evaluation. Inspired by [30], which employs persona-based agents for web usability testing, we utilize WebVoyager [15], a robust web navigation UI agent, to execute test operations and verify outcomes. We structure each test case's operation and



(a) Instruction-string lengths

(b) Test case numbers

Figure 2: Distributions of instruction lengths (left) and test case numbers per instruction (right).

Table 3: Statistics for the instruction string lengths and the test case numbers in our dataset.

Statistic	**Minimum**	**Maximum**	**Median**	**Average**
Instruction Length (chars)	324	876	483	496.8
Test Case Number	4	11	6	6.4

expected outcome into a standardized prompt, which directs the agent to simulate user interactions, analyze action trajectories and screenshots, and return YES, NO, or PARTIAL assessments based on requirement fulfillment. The process is shown on the right side of Fig. 1 (b). When the agent reaches its interaction limit, we trigger a decision prompt, inducing the agent to make a final decision. The prompts are presented in Appendix G. Considering the cost induced by multiple interactions with the website in evaluating each test case, we employ Qwen2.5-VL-32B-Instruct, an efficient open-source vision-language model that balances performance and cost-effectiveness, as the agent's engine.

3.3 Evaluation of Website Appearance

Apart from the fulfillment of the functionality and appearance constraints in the instructions, another important aspect of website generation is the level of relevance, harmony, and aesthetics of the webpage. To conduct a quantitative analysis of this aspect, we designed a set of detailed metrics, ranging from the success of rendering and the relevance of the content to the harmony of the layout and the modernness of the design. We then place the metrics in a prompt, asking GPT-4o to grade the appearance of the website with a score ranging from 1 to 5 (the higher the better), as demonstrated in the middle part of Fig. 1 (b). The prompt is shown in Appendix. H. Examples are presented in Appendix. M.

3.4 Analysis of Dataset Attributes

We analyze the distribution of instruction lengths and the number of test cases per instruction. The corresponding plots are shown in Fig.2, and the minimum, maximum, median, and average values are summarized in Tab.3.

As depicted in Fig. 2 (a), most website-generation instructions contain between 400 and 600 characters, with a median length of 483 and an average length of 496.8. These relatively long prompts add considerable complexity, posing a meaningful challenge to the agents under evaluation. Fig. 2 (b) indicates that most instructions are associated with five to seven test cases. The median and average numbers of test cases are 6 and 6.4, respectively. Because each test case corresponds to a distinct requirement in the instruction, these statistics confirm that every instruction encompasses a sufficient set of functional and appearance requirements.

4 Experiments

4.1 Experimental Setup

Frameworks. We evaluate three popular code-agent frameworks: Bolt.diy [47], OpenHands [50], and Aider [1]. Bolt.diy is the open-source version of Bolt.new[9], a browser-based framework for generating and previewing web applications. It provides a user interface and a Linux-like WebContainer environment that can execute code. It first prompts the model to decide which frontend and backend frameworks to use (such as Vite, React, Remix, etc.), then imports the basic template and builds upon it. OpenHands is a platform for AI-powered software development agents. For OpenHands, we pair it with CodeActAgent to evaluate it on our benchmark. The adapted instruction is presented in Appendix I. Aider is a terminal-based AI programming framework that natively supports many popular programming languages, including Python, JavaScript, PHP, HTML, CSS, and more. Aider constructs a map of the entire codebase, which helps it function well in larger projects. We use the adapted instruction in Appendix J to generate websites with Aider.

Models. We first evaluate the three frameworks on DeepSeek-V3 [28], a model that is both performant and cost-effective. We then evaluate several strong general-purpose proprietary and open-source LLMs—including Claude-3.5-Sonnet [3], DeepSeek-R1 [12], GPT-4o [18], o3-mini [37], Qwen2.5-Coder-32B [17], and Qwen2.5-72B-Instruct [54]—on the best-performing framework, Bolt.diy. We do not test general-purpose models smaller than Qwen2.5-Coder-32B, as we observe that such models often fail to follow the specified output format and therefore cannot generate valid websites.

Training Details. To validate the effectiveness of our training set, we selectively generated Bolt.diy trajectories for a subset of 2K instructions from WebGen-Instruct using DeepSeek-V3. Using rejection sampling [59], we retained only the trajectories whose corresponding websites achieved an appearance score greater than or equal to 3, resulting in 600 trajectories. This filtering ensures that the remaining generated websites are relevant to the instructions and do not exhibit major rendering issues. We then fine-tuned Qwen2.5-Coder-Instruct models of sizes 7B, 14B, and 32B for 2 epochs, with a learning rate of 4e-5 and a batch size of 32. The 7B, 14B, and 32B models were trained on 8, 16, and 32 A800 GPUs, respectively. This fine-tuning process yields a family of models specialized in website generation, which we name **WebGen-LM**.

4.2 Experimental Results

We present the results on the entire WebGen-Bench dataset in Tab.4, and the accuracy for each category of instructions and test cases in Tab. 5. Accuracy is computed using the formula $\text{Accuracy} = \frac{N_{\text{Yes}} + 0.5 \times N_{\text{Partial}}}{N_{\text{Total}}} \times 100\%$, where N_{Yes} and N_{Partial} denote the number of test cases assessed as YES and PARTIAL, respectively, and N_{Total} is the total number of test cases.

Main Results. Based on the experimental results, we make the following observations: (1) As shown in Tab. 5, WebGen-LM-32B achieves the highest accuracy of 38.2%, surpassing the best proprietary model, DeepSeek-R1, by 10.4%, demonstrating the effectiveness of our training set and the rejection-sampling process. (2) Bolt.diy with DeepSeek-R1 as the engine achieves the highest accuracy among general LLMs at 27.8%, closely followed by Claude-3.5-Sonnet with an accuracy of 26.4%. This indicates that the best-performing models are still far from saturating WebGen-Bench, highlighting that our benchmark remains challenging for current LLMs and agent frameworks. (3) Smaller general open-source models, such as Qwen2.5-Coder-32B and Qwen2.5-72B-Instruct, show a significant performance gap compared to proprietary models. (4) In terms of appearance scores, Bolt.diy with Claude-3.5-Sonnet achieves the best performance of 3.0. The appearance score exhibits a loose correlation with accuracy, as functional webpages typically do not suffer from major rendering issues. To better understand the statistical characteristics of the generated websites, we analyzed the file count and line count in the generated codebases, as detailed in Appendix K.

Categorical Results. Apart from the three main instruction categories (shown in Tab. 2), we also classify the test cases into three primary categories based on what they are intended to evaluate: Functional Testing, Data Display Testing, and Design Validation Testing. Detailed definitions and statistics for these categories are provided in Appendix L. As shown in Tab. 5, among the different categories of test cases, Design Validation Testing achieves the highest accuracy in most cases,

[9] https://bolt.new

Table 4: Evaluation of three powerful code-agent frameworks using different proprietary and open-source models. Accuracy is computed using a weighted score, where YES samples are weighted by 1 and PARTIAL samples are weighted by 0.5; the total score is then divided by the number of test cases. The highest accuracy and appearance scores are marked in **bold**.

Test Name	Yes Rate	Partial Rate	No Rate	Start Failed	Accuracy	Appearance Score
Bolt.diy						
Claude-3.5-Sonnet	22.6	7.6	64.1	5.7	26.4	**3.0**
DeepSeek-R1	24.7	6.2	64.3	4.8	27.8	2.5
DeepSeek-V3	18.5	4.5	73.9	3.1	20.8	2.0
GPT-4o	10.4	4.8	64.5	20.4	12.8	1.5
o3-mini	17.9	3.4	40.0	38.6	19.6	1.6
Qwen2.5-Coder-32B	8.2	2.6	81.8	7.4	9.5	1.1
Qwen2.5-72B-Instruct	12.1	3.6	80.7	3.7	13.8	1.4
WebGen-LM-7B	24.9	7.1	68.0	0.0	28.4	2.5
WebGen-LM-14B	25.0	8.7	66.3	0.0	29.4	2.5
WebGen-LM-32B	34.2	8.0	57.8	0.0	**38.2**	2.8
OpenHands						
Claude-3.5-Sonnet	18.1	8.3	58.6	15.0	22.3	2.6
DeepSeek-R1	8.5	3.4	60.4	27.7	10.2	1.4
Deepseek-V3	7.4	3.2	73.9	15.5	9.0	1.5
Aider						
Claude-3.5-Sonnet	19.9	5.9	42.0	32.1	22.9	1.9
DeepSeek-R1	23.3	8.7	44.5	23.5	27.7	2.7
Deepseek-V3	12.5	3.1	54.3	30.1	14.1	1.2

Table 7: Comparison of yes rate and accuracy at different sample sizes. The base model is Qwen2.5-Coder-32B-Instruct.

Sample Number	Yes Rate	Accuracy
150	21.8	25.1
300	28.6	31.9
600	**34.2**	**38.2**

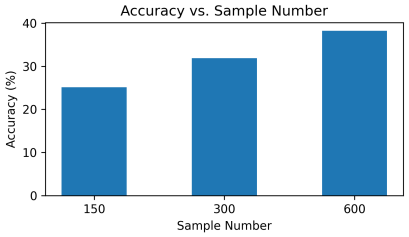


Figure 3: Accuracy vs. sample number.

while Functional Testing generally yields lower accuracy. Among instruction categories, Content Presentation consistently demonstrates the highest accuracies. This indicates that superficial aspects, such as color themes, are easier to implement than deeper internal functionalities.

4.3 Ablation Studies

Analysis of the Accuracy of UI Agent Testing Results. To analyze the accuracy of the UI agent testing process, we manually examined three sets of testing results on Bolt.diy. We select the results of Claude-3.5-Sonnet, DeepSeek-R1, and DeepSeek-V3 as the accuracies of these three models are high and are close to each other. The manual testing results serve as the ground truth and require precision; therefore, three human testers independently annotated the results and we assessed the consistency of their annotations. If the annotations of a test case are inconsistent, a fourth human tester is tasked with re-examining the test case and the inconsistent annotations to decide on a final annotation. We present the results of manual testing in Tab. 6. The Alignment Rate is computed with Alignment Rate $= \frac{N_{\text{Manual=Agent}}}{N_{\text{total}}} \times 100\%$, where $N_{\text{Manual=Agent}}$ denotes the number of test cases where the agent-generated result aligns with the manually-annotated result.

Analysis of Reliability of Appearance Scores. To strengthen the appearance assessment, we evaluate the bolt.diy results produced by Claude-3.5-Sonnet, DeepSeek-R1, and DeepSeek-V3 with

Table 5: Category-wise evaluation results. The first three columns represent categories of website-generation instructions, while the last three represent categories of test cases. The highest score in each category is marked in bold.

Test Name	Instruction Categories			Test Case Categories		
	Content Presentation	User Interaction	Data Management	Functional Testing	Data Display Testing	Design Validation Testing
Bolt.diy						
Claude-3.5-Sonnet	35.6	21.2	26.2	17.1	26.3	52.0
DeepSeek-R1	43.7	20.6	24.7	21.1	29.3	44.3
DeepSeek-V3	37.1	16.6	11.2	10.5	28.2	38.1
GPT-4o	26.4	5.9	11.2	4.7	19.6	24.6
o3-mini	28.7	17.7	13.4	11.4	25.5	33.6
Qwen2.5-Coder-32B	17.5	6.9	5.9	1.9	14.5	23.0
Qwen2.5-72B-Instruct	28.2	10.1	5.6	5.8	21.0	25.4
WebGen-LM-7B	27.9	23.8	38.1	22.0	27.7	47.5
WebGen-LM-14B	30.2	27.8	31.6	23.6	26.9	49.2
WebGen-LM-32B	**46.6**	**33.2**	**38.8**	**29.1**	**43.0**	**56.1**
OpenHands						
Claude-3.5-Sonnet	32.8	18.4	18.4	12.4	33.9	32.0
DeepSeek-R1	16.4	8.9	5.9	5.0	9.9	25.0
Deepseek-V3	12.6	7.3	8.4	3.8	8.1	25.0
Aider						
Claude-3.5-Sonnet	31.9	21.1	16.6	14.9	30.1	34.0
DeepSeek-R1	39.1	28.6	13.4	17.6	35.2	44.3
Deepseek-V3	17.8	12.8	12.5	9.7	19.1	18.4

Table 6: Alignment between the UI agent testing results and human testing results. The alignment rate denotes the proportion of test cases in which the UI agent's results match those of human testers.

Model	Testing Method	Yes Rate	Partial Rate	No Rate	Accuracy	Alignment Rate
Claude-3.5-Sonnet	UI Agent	22.6	7.6	64.1	26.4	90.3
	Manual	22.4	7.1	59.0	26.0	–
Deepseek-R1	UI Agent	24.7	6.2	64.3	27.8	86.1
	Manual	28.0	4.3	58.1	30.1	–
Deepseek-V3	UI Agent	18.5	4.5	73.9	20.8	94.4
	Manual	19.0	4.5	70.3	21.3	–

two additional strong multimodal graders—o3 and Claude-3.5-Sonnet. We also manually grade the website screenshots to capture human preference. The results are presented in Tab. 8. Although o3 and Claude-3.5-Sonnet assign slightly lower scores than GPT-4o, the relative ordering of the three results is unchanged (Claude-3.5-Sonnet > DeepSeek-R1 > DeepSeek-V3). Also, for every set of screenshots, the scores descend in the same order: GPT-4o > o3 > Claude-3.5-Sonnet, showing a consistency in the models' grading pattern. The ensemble results of the three grading models and the human scores both show the same ranking as GPT-4o. These findings indicate that GPT-4o reliably reflects relative appearance quality and aligns with human preference, making it a suitable choice under a limited budget.

Analysis of the Number of Training Samples. We analyze the effect of the number of training samples on the accuracy of the fine-tuned models. Specifically, we fine-tune Qwen2.5-Coder-32B-Instruct using 150, 300, and 600 samples, respectively. As shown in Fig. 3 and Tab. 7, accuracy consistently increases with the number of training samples, highlighting the potential of our training set. We did not sample additional trajectories due to API budget constraints. Nevertheless, the current sample size already demonstrates the effectiveness of WebGen-Instruct for training website

Table 8: Average appearance scores obtained when the output of each agent–engine LLM is graded by different grading methods.

Agent-engine LLM \ Grading method	GPT-4o	o3	Claude-3.5-Sonnet	Average (ensemble)	Human
Claude-3.5-Sonnet	**3.0**	2.7	2.4	2.7	2.8
DeepSeek-R1	**2.5**	2.3	2.2	2.3	2.3
DeepSeek-V3	**2.0**	1.9	1.7	1.9	1.9

generation LLMs. Further accuracy improvements through additional data or techniques such as data augmentation are left for future work.

Analysis of Errors in WebGen-Bench Tasks. We analyze the errors and flaws that occur in web generation pipelines using mainstream LLMs such as DeepSeek-V3. A detailed explanation of the error types, along with examples of erroneous cases, is provided in Appendix N. In addition, we present statistics on the distribution of error types for each task in the test set, as shown in Fig. 4. Currently, more than half of the task errors are due to failures in launching a web page or in modifying a template. These results highlight significant potential for future research focused on improving the success rate of web page initialization and template adaptation in generated websites.

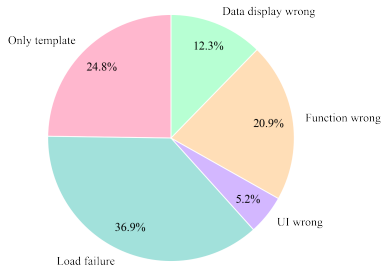


Figure 4: The distribution of the task errors.

5 Conclusion

In this paper, we introduce WebGen-Bench, a novel benchmark for evaluating the ability of LLM-based agents to generate websites from scratch. The benchmark requires agents to construct and organize multi-file codebases while satisfying various functional and visual constraints. We evaluate three code-agent frameworks using both proprietary and open-source LLMs. The best-performing combination, Bolt.diy with DeepSeek-R1, achieves an accuracy of only 27.8%, highlighting the challenging nature of our benchmark. Additionally, we construct a training set of 6,667 website-generation instructions and fine-tune Qwen2.5-Coder-32B on 600 Bolt.diy trajectories generated by DeepSeek-V3, resulting in an accuracy of 38.2%—surpassing even the best proprietary model.

6 Acknowledgements

This study was supported in part by National Key R&D Program of China Project 2022ZD0161100, in part by the Centre for Perceptual and Interactive Intelligence, a CUHK-led InnoCentre under the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government, in part by NSFC-RGC Project N_CUHK498/24, and in part by Guangdong Basic and Applied Basic Research Foundation (No. 2023B1515130008, XW).

References

[1] Aider-AI. Ai pair programming in your terminal, 2024. Accessed: 2025-04-22.

[2] Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. Swe-bench+: Enhanced coding benchmark for llms. *arXiv preprint arXiv:2410.06992*, 2024.

[3] Anthropic. Introducing claude 3.5 sonnet, 2024. Accessed: 2025-04-22.

[4] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[5] Shubham Chandel, Colin B. Clement, Guillermo Serrato, and Neel Sundaresan. Training and evaluating a jupyter notebook data science assistant, 2022.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[7] Xinyan Chen, Renrui Zhang, Dongzhi Jiang, Aojun Zhou, Shilin Yan, Weifeng Lin, and Hongsheng Li. Mint-cot: Enabling interleaved visual tokens in mathematical chain-of-thought reasoning. *arXiv preprint arXiv:2506.05331*, 2025.

[8] GitHub Copilot. Github copilot, 2024. Accessed: 2025-04-22.

[9] Cursor. Cursor: The ai code editor, 2024. Accessed: 2025-04-22.

[10] Juha Eskonen, Julen Kahles, and Joel Reijonen. Automating gui testing with image-based deep reinforcement learning. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 160–167, 2020.

[11] Pedro M. Fernandes, Manuel Lopes, and Rui Prada. Agents for automated user experience testing. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 247–253, 2021.

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[13] Zilu Guo, Hongbin Lin, Zhihao Yuan, Chaoda Zheng, Pengshuo Qiu, Dongzhi Jiang, Renrui Zhang, Chun-Mei Feng, and Zhen Li. Pisa: A self-augmented data engine and training strategy for 3d understanding with large models. *arXiv preprint arXiv:2503.10529*, 2025.

[14] Ziyu Guo, Ray Zhang, Hao Chen, Jialin Gao, Dongzhi Jiang, Jiaze Wang, and Pheng-Ann Heng. Sciverse: Unveiling the knowledge comprehension and visual reasoning of lmms on multi-modal scientific problems. *arXiv preprint arXiv:2503.10627*, 2025.

[15] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.

[16] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

[17] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

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

[19] Nizar Islah, Justine Gehring, Diganta Misra, Eilif Muller, Irina Rish, Terry Yue Zhuo, and Massimo Caccia. Gitchameleon: Unmasking the version-switching capabilities of code generation models, 2024.

[20] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

[21] Dongzhi Jiang, Ziyu Guo, Renrui Zhang, Zhuofan Zong, Hao Li, Le Zhuo, Shilin Yan, Pheng-Ann Heng, and Hongsheng Li. T2i-r1: Reinforcing image generation with collaborative semantic-level and token-level cot. *arXiv preprint arXiv:2505.00703*, 2025.

[22] Dongzhi Jiang, Guanglu Song, Xiaoshi Wu, Renrui Zhang, Dazhong Shen, Zhuofan Zong, Yu Liu, and Hongsheng Li. Comat: Aligning text-to-image diffusion model with image-to-text concept matching. *arXiv preprint arXiv:2404.03653*, 2024.

[23] Dongzhi Jiang, Renrui Zhang, Ziyu Guo, Yanwei Li, Yu Qi, Xinyan Chen, Liuhui Wang, Jianhan Jin, Claire Guo, Shen Yan, et al. Mme-cot: Benchmarking chain-of-thought in large multimodal models for reasoning quality, robustness, and efficiency. *arXiv preprint arXiv:2502.09621*, 2025.

[24] Dongzhi Jiang, Renrui Zhang, Ziyu Guo, Yanmin Wu, Jiayi Lei, Pengshuo Qiu, Pan Lu, Zehui Chen, Guanglu Song, Peng Gao, et al. Mmsearch: Benchmarking the potential of large models as multi-modal search engines. *arXiv preprint arXiv:2409.12959*, 2024.

[25] Dongzhi Jiang, Renrui Zhang, Yankai Shu, Yuyang Peng, Zhuofan Zong, Yuchen Duan, Zihao Wang, Jiaming Liu, Hao Chen, Ziyu Guo, Junyan Ye, Rui Liu, Pheng Ann Heng, Shanghang Zhang, and Hongsheng Li. Ulmevalkit: An open-source toolkit for evaluating unified large multi-modal models and generative models, October 2025.

[26] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[27] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation, 2022.

[28] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[29] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

[30] Yuxuan Lu, Bingsheng Yao, Hansu Gu, Jing Huang, Jessie Wang, Laurence Li, Jiri Gesi, Qi He, Toby Jia-Jun Li, and Dakuo Wang. Uxagent: An llm agent-based usability testing framework for web design. *arXiv preprint arXiv:2502.12561*, 2025.

[31] Bingqi Ma, Zhuofan Zong, Guanglu Song, Hongsheng Li, and Yu Liu. Exploring the role of large language models in prompt encoding for diffusion models. *arXiv preprint arXiv:2406.11831*, 2024.

[32] Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua Li, Fei Huang, and Yongbin Li. Lingma swe-gpt: An open development-process-centric language model for automated software improvement. *arXiv preprint arXiv:2411.00622*, 2024.

[33] Zexiong Ma, Shengnan An, Zeqi Lin, Yanzhen Zou, and Bing Xie. Repository structure-aware training makes slms better issue resolver. *arXiv preprint arXiv:2412.19031*, 2024.

[34] Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzhen Zou, and Bing Xie. Sorft: Issue resolving with subtask-oriented reinforced fine-tuning. *arXiv preprint arXiv:2502.20127*, 2025.

[35] Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. Swe-lancer: Can frontier llms earn $1 million from real-world freelance software engineering? *arXiv preprint arXiv:2502.12115*, 2025.

[36] Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro Von Werra, and Shayne Longpre. Octopack: Instruction tuning code large language models. In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

[37] OpenAI. Openai o3-mini, 2025. Accessed: 2025-04-22.

[38] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

[39] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

[40] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[41] Houxing Ren, Zimu Lu, Weikang Shi, Haotian Hou, Yunqiao Yang, Ke Wang, Aojun Zhou, Junting Pan, Mingjie Zhan, and Hongsheng Li. Alignment with fill-in-the-middle for enhancing code generation. *arXiv preprint arXiv:2508.19532*, 2025.

[42] Houxing Ren, Mingjie Zhan, Zhongyuan Wu, and Hongsheng Li. Empowering character-level text infilling by eliminating sub-tokens. *arXiv preprint arXiv:2405.17103*, 2024.

[43] Houxing Ren, Mingjie Zhan, Zhongyuan Wu, Aojun Zhou, Junting Pan, and Hongsheng Li. Reflectioncoder: Learning from reflection sequence for enhanced one-off code generation. *arXiv preprint arXiv:2405.17057*, 2024.

[44] Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. Specrover: Code intent extraction via llms. *arXiv preprint arXiv:2408.02232*, 2024.

[45] Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. Visual cot: Unleashing chain-of-thought reasoning in multi-modal language models. *arXiv e-prints*, pages arXiv–2403, 2024.

[46] Dazhong Shen, Guanglu Song, Yi Zhang, Bingqi Ma, Lujundong Li, Dongzhi Jiang, Zhuofan Zong, and Yu Liu. Adt: Tuning diffusion models with adversarial supervision. *arXiv preprint arXiv:2504.11423*, 2025.

[47] stackblitz labs. bolt.diy, 2024. Accessed: 2025-04-22.

[48] Samantha . Stahlke, Atiya Nova, and Pejman Mirza-Babaei. Artificial playfulness: A tool for automated agent-based playtesting. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI EA '19, page 1–6, New York, NY, USA, 2019. Association for Computing Machinery.

[49] US Bureau of Labor Statistics. Table b-3. average hourly and weekly earnings of all employees on private nonfarm payrolls by industry sector, seasonally adjusted., 2024.

[50] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2024.

[51] Scott Wu. Introducing devin, the first ai software engineer, 2024. Accessed: 2025-04-22.

[52] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

[53] Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. Swe-fixer: Training open-source llms for effective and efficient github issue resolution. *arXiv preprint arXiv:2501.05040*, 2025.

[54] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[55] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[56] 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. Swe-bench multimodal: Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*, 2024.

[57] Junyan Ye, Dongzhi Jiang, Jun He, Baichuan Zhou, Zilong Huang, Zhiyuan Yan, Hongsheng Li, Conghui He, and Weijia Li. Blink-twice: You see, but do you observe? a reasoning benchmark on visual perception. *arXiv preprint arXiv:2510.09361*, 2025.

[58] Junyan Ye, Dongzhi Jiang, Zihao Wang, Leqi Zhu, Zhenghao Hu, Zilong Huang, Jun He, Zhiyuan Yan, Jinghua Yu, Hongsheng Li, et al. Echo-4o: Harnessing the power of gpt-4o synthetic images for improved image generation. *arXiv preprint arXiv:2508.09987*, 2025.

[59] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

[60] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*, 2023.

[61] Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Peng Gao, et al. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? *ECCV 2024*, 2024.

[62] Renrui Zhang, Xinyu Wei, Dongzhi Jiang, Ziyu Guo, Shicheng Li, Yichi Zhang, Chengzhuo Tong, Jiaming Liu, Aojun Zhou, Bin Wei, et al. Mavis: Mathematical visual instruction tuning with an automatic data engine. *arXiv preprint arXiv:2407.08739*, 2024.

[63] Shudan Zhang, Hanlin Zhao, Xiao Liu, Qinkai Zheng, Zehan Qi, Xiaotao Gu, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Naturalcodebench: Examining coding performance mismatch on humaneval and natural user prompts. *arXiv preprint arXiv:2405.04520*, 2024.

[64] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1592–1604, 2024.

[65] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

[66] Zhuofan Zong, Dongzhi Jiang, Bingqi Ma, Guanglu Song, Hao Shao, Dazhong Shen, Yu Liu, and Hongsheng Li. Easyref: Omni-generalized group image reference for diffusion models via multimodal llm. *arXiv preprint arXiv:2412.09618*, 2024.

[67] Zhuofan Zong, Bingqi Ma, Dazhong Shen, Guanglu Song, Hao Shao, Dongzhi Jiang, Hongsheng Li, and Yu Liu. Mova: Adapting mixture of vision experts to multimodal context. *arXiv preprint arXiv:2404.13046*, 2024.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and precede the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: As shown in Abstract and Introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: As shown in Limitations and Future Work.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: The paper does not include theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: As detailed in Experiments and the open-source data and code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We open-source all our code and data.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: As shown in Experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Error bars are not reported because it would be too computationally expensive.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: As detailed in Experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: As explained in Ethics Statement in Appendix.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: As explained in Ethics Statement in Appendix.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: As explained in Ethics Statement in Appendix.

    Guidelines:

    - The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: As explained in Ethics Statement in Appendix.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: As shown in the released code and data.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: We did not use crowdsourcing in our experiments. We used authors and student volunteers instead.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our study does not pose any potential risks to the participants.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: As detailed in Method.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Ethics Statement

The WebGen-Bench dataset is entirely composed of synthetically generated instructions and test cases, curated manually and synthesized using artificial intelligence. While this resource is non-commercial, we emphasize that its construction process maintains a clear distance from potential ethical or legal concerns, particularly regarding intellectual property.

Legal compliance. We take great care in our methodology to uphold copyright integrity, utilizing three protective approaches to safeguard against infringement: (1) all base project descriptions originate from the creative efforts of the authors and student volunteers; (2) the 20 fundamental categories are sufficiently abstracted through systematic analysis; and (3) our framework does not copy content from existing websites or platforms, thereby avoiding copyright infringement risks associated with specific commercial implementations.

Dataset Intended Usage and License. We document the WebGen-Bench dataset in this paper and note that both the dataset and the code used for reproducing results are publicly available. We intend for researchers to use this dataset to better evaluate the ability of LLM-based agents to generate websites from scratch. We take full responsibility in the event of any rights violations. The WebGen-Bench dataset and our open-source code are released under the MIT license.

B Limitations and Future Work

Website generation in this work is primarily conducted using TypeScript, JavaScript, CSS, and HTML. Other languages such as Python, Java, and Go are not used, due to the complexity of integrating them into the agent framework. Expanding the range of supported languages and tools for automatic website generation with code agents is a promising direction for future research. Additionally, we only employed supervised fine-tuning to enhance the performance of open-source LLMs on website generation, without utilizing other post-training strategies such as reinforcement learning or direct preference optimization [39]. These methods present valuable opportunities for future exploration.

C Prompt for Deriving Instructions from Website Development Project Descriptions

Fig. 5 presents the prompt used to derive website-generation instructions from web development project descriptions created by human annotators. Notably, the model is instructed to exclude any requirements related to technical implementation details, as the goal is to evaluate the code agents' ability to make such decisions independently.

D Details of the Decontamination Process

In this section, we introduce the methods we used to decontaminate the training set from the testing set. We first employ 5-gram Jaccard similarity, removing the instructions in the training set with a similarity score higher than 0.6 with one of the instructions in the testing set. Then, to remove the instructions that are semantically similar to those in the testing set, we compute the sentence embeddings of the instructions using the all-MiniLM-L6-v2 model of Sentence-Transformer [40], and compute the cosine similarity of the embeddings. We experimented with various threshold settings, and finally settled on removing the training instructions with a cosine similarity of larger than 0.55.

We then inspect whether the remaining training samples contain instructions that are semantic duplicates of the instructions in the testing. For each testing instruction, we retrieve the top-3 training instructions with the highest cosine similarity, and manually inspect them for semantic duplication. We found that the retrieved training samples are all completely different from the testing samples, proving that the final training set is not contaminated. The first three samples in WebGen-Bench and their top matches in the training set are shown in Fig. 7, Fig. 8, and Fig. 9, respectively. The matches are completely different from the test samples. Fig. 6 shows the distribution of the cosine similarity between the test set and the training set. The cosine similarity is gathered around 0.2 to 0.3, which is relatively low.

Prompt:
<task>
You will be given a piece of text containing the basic information of a web development project. The information involves a main objective and a list of functional and appearance requirements. You are requested to convert the information into instructions to build a web application. You should output a detailed multi-sentence instruction in English explaining in detail the different functions the applications should have.
</task>

<important>
1. Your output should align with the main objective of the website and expand upon the requirements.
2. You should not specify any technical details in the instructions.
3. You should not refer to any outside applications in your instructions.
4. You should not output any additional comments.
</important>

The following is an example:

<example>

Objective:
A hotel and travel ticket distribution website.
Other requirements:
1. User login
2. Order tickets and hotels
3. Cancel orders
4. Verify orders
5. Browse tickets and hotels
6. Light blue background and dark olive green component

Converted Instruction:

Please implement a distribution website for travel and ticketing that sells products such as tickets and hotels. The website should have functionalities for placing, canceling, and verifying orders. Users should be able to log in, browse products like tickets and hotels, place orders for selected products, cancel selected orders, and verify consumption records. Use light blue in the background layer and dark olive green for the component layer.
</example>

Objective:
{Objective}
Other requirements:
{Other requirements}

Converted Instruction:

Figure 5: The prompt for deriving instructions from human annotated descriptions.

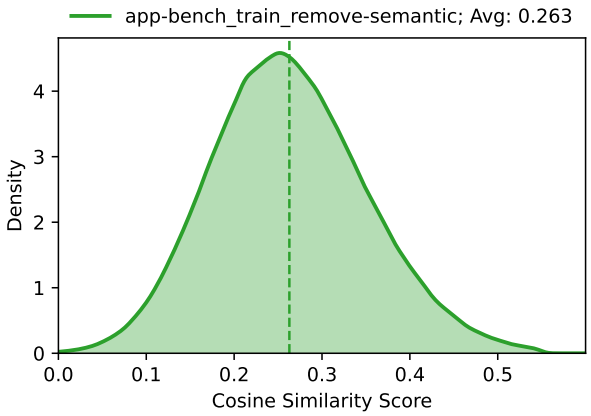


Figure 6: The distribution of the cosine similarity between the test set and the training set.

Test Instruction 1: Please implement a website for generating stock reports to provide stock information and analysis. The website should have the functionality to search and summarize stock information, and generate customized stock reports based on user requirements. Users should be able to input stock codes or names, select report formats and content, and the website will automatically generate the corresponding reports. The reports should include basic stock information, market trends, financial data, and more. Set the background color to white and the component color to navy.

Match 1: Please implement a website for generating PDF reports that creates PDF files containing directories, word clouds, logos, and chart displays. The website should have functionalities for uploading data, selecting templates, customizing content, previewing, and downloading PDFs. Users should be able to upload relevant data, choose from different templates, customize the report content, preview the generated PDF file, and download the final PDF report. Specify bisque as the base color and dark salmon for all components.
Similarity: 0.549

Match 2: Please implement an accounting factory website for enterprise financial management and statistics. The website should have functionalities for creating service enterprises, setting declaration types, and extracting statistics by quarter and year. Users should be able to log in, create and manage service enterprises, set declaration types, view and analyze financial data, and perform WeChat payment and other operations. Set page background to light beige; color all components with sienna.
Similarity: 0.542

Match 3: Please implement a report frontend website to display vehicle inspection report data. The website should have functionalities for displaying report templates, inspection report information, and audit status. Users should be able to log in, browse, and view inspection reports, including report details, inspection results, and audit status. Use powder blue for container backgrounds and royal blue for component visuals.
Similarity: 0.538

Figure 7: Top semantic matches for the first test instruction in WebGen-Bench with similarity scores.

E Application Categories of WebGen-Instruct and WebGen-Bench.

Tab. 9 lists the 20 application categories manually summarized by the authors through browsing web development projects on popular platforms that connect programmers with clients seeking custom website solutions, such as Upwork[10], Freelancer[11], and Proginn[12]. These application categories serve as seed ideas for our human annotators during the brainstorming of new application scenarios.

Detailed definition of each category is as follows:

[10]https://www.upwork.com
[11]https://www.freelancer.com
[12]https://www.proginn.com

Test Instruction 2: Please implement a web-based neighborhood mapping application for comparing data across different areas. The application should allow users to compare demographic, economic, and crime data across different areas. The application should also include data dashboards with interactive charts and customizable layouts. Use ivory for the background and forest green for components.

Match 1: Please implement a geographic spatial data processing website for handling and analyzing geographic spatial data. The website should have functionalities for data conversion, file interpolation, data operation, and data extraction. Users should be able to upload geographic spatial data files, choose different data formats for conversion, perform data interpolation and operation, and extract the required data. The website should also provide data visualization functionality, allowing users to view and analyze geographic spatial data. Assign mint frost as the background color and apply seagreen to all elements.
Similarity: 0.546

Match 2: Please implement a geographic information system website for displaying maps and managing the backend. The website should have map visualization capabilities to display different types of geographic information. The backend management platform should be able to manage users, permissions, roles, menus, and support specific business management, such as setting up construction orders, inspecting and evaluating drainage facilities, and managing facilities. Users should be able to log in, browse maps, manage backend data, and perform related operations. Set all pages to have a cream background and dark orange components.
Similarity: 0.542

Match 3: Please develop a Boundary Hunter app to provide nearby data research services. The app should have functionalities for data research, report generation, and user management. Users should be able to log in, browse nearby data research projects, submit research requests, and view reports. The app should also have automated testing and stress testing capabilities to ensure its stability and performance. Use ghost white for the outer layout and cadet blue for UI blocks.
Similarity: 0.542

Figure 8: Top semantic matches for the second test instruction in WebGen-Bench with similarity scores.

- Personal Portfolio Sites: Showcase individual professional projects, achievements, and skills.
- Company Brochure Sites: Static or minimally interactive websites providing company information, products, services, and contact details.
- Personal Blog Sites: Regularly updated content sites focusing on personal writing, opinions, experiences, and sharing of knowledge.
- Social Media Platforms: Applications enabling users to interact socially, share content, and build networks.
- Discussion Forums: Platforms facilitating conversations, topic-based discussions, threads, and community interactions.
- E-commerce Web Applications: Online platforms designed for buying and selling goods and services, handling transactions, inventory, and payments.
- Email Clients: Applications for managing emails, sending, receiving, organizing, and scheduling email communication.
- Project Management Tools: Platforms aiding in task organization, scheduling, collaboration, and resource management for projects.
- Streaming and Interactive Platforms: Media-centric platforms for video, audio streaming, or interactive media consumption.
- CRM Systems: Customer Relationship Management tools designed to manage interactions, sales, customer data, and marketing.
- ERP Platforms: Enterprise Resource Planning systems integrating core business processes such as finance, HR, supply chain, and operations.
- Internal Tools: Applications focused on internal company operations, communication, and collaboration.
- News and Information Sites: Platforms primarily dedicated to delivering news content, articles, and timely updates.

Test Instruction 3: Please implement a multi-company dashboard for managing and displaying financial data from multiple companies. The dashboard should be able to collect and display financial information from each company, provide consolidated reports, and support cross-company comparisons and reporting. Users should be able to browse financial data from each company, view consolidated reports, and perform financial management and reporting. Apply mint cream as the background; style all components with teal.

Match 1: Please implement a multi-lingual accounting website for managing financial accounts. The website should have functionalities for logging in, registering, recording, querying, and statistical analysis. Users should be able to log in, create, edit, and delete financial accounts, query historical accounts, and analyze financial status. The website should support multiple languages to facilitate use by users of different languages. Configure the background color to spring green, with components using lime green.
Similarity: 0.549

Match 2: Please implement an enterprise resource planning backend management system for managing internal company data. The system should have user management, permission management, module lists, add, edit, delete, and display functions. Users should be able to log in to the system, browse and manage data in different modules, including adding new data, editing existing data, deleting unnecessary data, and displaying all data. The system should also support Excel import and export functions for convenient batch data operations. Use alabaster as the screen background and dark cyan for component highlights.
Similarity: 0.542

Match 3: Please implement a data visualization website for a telecommunications company to display company data. The website should have multiple pages, each with different dynamic effects. The website should include various charts and maps, with charts having dynamic refresh effects and maps implementing three-level drill-down functionality. Users should be able to browse different pages and view the company's data and statistical information. Use almond as the screen background and sienna for component highlights.
Similarity: 0.540

Figure 9: Top semantic matches for the third test instruction in WebGen-Bench with similarity scores.

Table 9: 20 application categories manually summarized from popular web-development websites.

Application Category	**Application Category**
Productivity Applications	Project Management Tools
Internal Tools	Company Brochure Sites
E-commerce Web Applications	Streaming and Interactive Platforms
Analytics Platforms/Dashboards	News and Information Sites
Publishing/Blogging Platforms	ERP Platforms
Travel Booking Portals	Learning Platforms
CRM Systems	Social Media Platforms
Discussion Forums	Personal Blog Sites
Email Clients	Browser-Based Games
Job Search Platforms	Personal Portfolio Sites

- Publishing/Blogging Platforms: Platforms enabling users to publish, edit, and manage content on a large scale.
- Analytics Platforms/Dashboards: Applications providing insights through data visualization, including Business Intelligence and Financial Dashboards.
- Browser-Based Games: Interactive, entertainment-focused applications running directly in web browsers.
- Learning Platforms: Educational platforms providing courses, training materials, quizzes, and learning management systems.
- Travel Booking Portals: Platforms allowing users to search, compare, and book travel services like flights, hotels, and car rentals.
- Job Search Platforms: Websites connecting job seekers with employers, allowing job postings, applications, and resume management.
- Productivity Applications: Tools for productivity tasks like document editing, spreadsheets, presentations, and collaborative work.

Prompt:
Act as a testing specialist. Based on the provided prompt below, which was used to generate a website, create a list of 5-10 actionable instructions to test the website's functionality, content accuracy, and user experience. Each instruction must:

1. Direct a UI agent to perform a single, atomic task.
2. Include validation criteria.
3. Align with the goals and features described in the original prompt.
4. Ensure each task is atomic (tests one function at a time) and avoids combining multiple sub-tasks.

Structure each instruction as:

Task: Clear, singular task for the UI agent.
Expected Result: Specific outcome to confirm success.

Original prompt:
{orig prompt}

Focus on testing:
- Core functionalities (e.g., forms, navigation).
- Content relevance to the prompt's intent.
- Accessibility and responsiveness.
- Appearance requirements.

IMPORTANT: The tasks must directly reflect ALL of the prompt's requirements and ensure each instruction is independent and minimal. You must not include tasks that test functions that are not explicitly required by the original prompt!

Figure 10: The prompt for deriving test cases that covers all the functional and appearance requirements in the instruction. The {orig prompt} is replaced with the corresponding website-generation instruction.

F Prompt for Creating Website Test Cases

Fig. 10 presents the prompt used to construct test cases that evaluate whether the generated website fulfills the requirements specified in the corresponding instruction. The prompt emphasizes the importance of ensuring that all functionality and appearance requirements are covered by the generated test cases. Conversely, every test case should directly reflect an aspect of the instruction. This ensures that the website is thoroughly evaluated and that all test cases are valid.

G Prompt for Automatic Evaluation of Test Cases Using an UI Agent

Fig. 11 presents the prompt used to instruct the UI agent to perform the operation described in the test case and respond with YES, NO, or PARTIAL, depending on whether the expected outcome is achieved. Fig. 12 shows the prompt used to induce the agent to make a final decision when the maximum number of allowed website interactions has been reached.

H Prompt for Grading Website Appearance

Fig. 13 shows the prompt used to grade the aesthetics of webpage appearances. The grading vision-language model (GPT-4o in this case) is instructed to consider metrics such as successful rendering, content relevance, layout harmony, and the modernity and visual appeal of the design, and then output a grade ranging from 1 to 5 (the higher, the better).

I Prompt for Adapting OpenHands Paired with CodeActAgent for WebGen-Bench Evaluation

Figure 14 presents the prompt used to evaluate OpenHands in combination with CodeActAgent on the WebGen-Bench benchmark.

Start-Testing Prompt:
Task: {task}
Expected Result: {expected result}

Instructions:
- Attempt the task as a user would, using the UI elements available.
- Make multiple attempts if needed to try and achieve the expected result.
- Observe whether the expected result is fully, partially, or not at all achieved.
- IMPORTANT: You can at most interact with the website 15 times. If the limit is reached, directly output your answer.

At the end of your testing, answer only with one of the following:
- YES: if the expected result was fully achieved.
- NO: if the expected result could not be achieved at all.
- PARTIAL: if only some aspects of the expected result were achieved.

Figure 11: The prompt for starting the operation of a test case, where {task} is replaced with the operation to be performed, {expected result} is replaced with the expected state of the website after the operation is performed.

Limit-reached Prompt:
You have reached the maximum number of allowed interactions with the website.

Please evaluate the outcome of your attempts based on the expected result:

Expected Result: {expected result}

Now, answer with one of the following:

- YES: if the expected result was fully achieved during your interactions.
- NO: if the expected result was not achieved at all.
- PARTIAL: if the expected result was only partially achieved.

Provide your final answer based on your testing experience.

Figure 12: The prompt for inducing an answer when the limit of the number of website interactions is reached, where {task} is replaced with the operation to be performed, {expected result} is replaced with the expected state of the website after the operation is performed.

J Prompt for Aider to Generate Websites for WebGen-Bench Evaluation

Fig. 15 shows the prompt used by Aider to generate websites for the WebGen-Bench evaluation.

K Analysis of Average File Count and Average Line Count

Fig. 16 reports the average file and line counts produced by each model, while Fig. 17 (a) and Fig. 17 (b) show the distributions of file and line counts generated specifically by WebGen-LM-32B. As shown in Fig. 16, GPT-4o, o3-mini, Qwen2.5-72B-Instruct, and Qwen2.5-Coder-32B-Instruct exhibit high average file counts relative to their average line counts, yet their overall performance remains relatively low. One plausible explanation is that, although these models create many files, the files are poorly organized and each contains too little code to support a complete website.

By contrast, the WebGen-LM models generate more lines of code without disproportionately increasing the number of files. Their average line counts all exceed those of DeepSeek-V3, the teacher model used during distillation—an effect that can partly be attributed to the use of rejection sampling. For every WebGen-LM variant, both the file count and the line count rise consistently with model size, indicating that the generated websites become increasingly comprehensive and complex as model scale grows.

Appearance-Grading Prompt:
Instruction:
You are tasked with evaluating the functional design of a webpage that had been constructed based on the following instruction:

{instruction}

Grade the webpage's appearance on a scale of 1 to 5 (5 being highest), considering the following criteria:

- Successful Rendering: Does the webpage render correctly without visual errors? Are colors, fonts, and components displayed as specified?
- Content Relevance: Does the design align with the website's purpose and user requirements? Are elements (e.g., search bars, report formats) logically placed and functional?
- Layout Harmony: Is the arrangement of components (text, images, buttons) balanced, intuitive, and clutter-free?
- Modernness & Beauty: Does the design follow contemporary trends (e.g., minimalism, responsive layouts)? Are colors, typography, and visual hierarchy aesthetically pleasing?

Grading Scale:

- 1 (Poor): Major rendering issues (e.g., broken layouts, incorrect colors). Content is irrelevant or missing. Layout is chaotic. Design is outdated or visually unappealing.
- 2 (Below Average): Partial rendering with noticeable errors. Content is partially relevant but poorly organized. Layout lacks consistency. Design is basic or uninspired.
- 3 (Average): Mostly rendered correctly with minor flaws. Content is relevant but lacks polish. Layout is functional but unremarkable. Design is clean but lacks modern flair.
- 4 (Good): Rendered well with no major errors. Content is relevant and logically organized. Layout is harmonious and user-friendly. Design is modern and visually appealing.
- 5 (Excellent): Flawless rendering. Content is highly relevant, intuitive, and tailored to user needs. Layout is polished, responsive, and innovative. Design is cutting-edge, beautiful, and memorable.

Task:
Review the provided screenshot(s) of the webpage. Provide a detailed analysis and then assign a grade (1–5) based on your analysis. Highlight strengths, weaknesses, and how well the design adheres to the specifications.

Your Response Format:

Analysis: [2–4 paragraphs addressing all criteria, referencing the instruction]

Grade: [1–5]

Your Response:

Figure 13: The prompt for grading the appearance of the webpage.

OpenHands Prompt:
Create a website app using typescript, html, and css. Your codebase should be able to be setup using 'npm install', and the service should be able to be started using 'npm run dev'.

{instruction}

Figure 14: The prompt for testing OpenHands paired with CodeActAgent on WebGen-Bench.

Aider Prompt:
You are Aider, an expert AI assistant and exceptional senior software developer with vast knowledge across multiple programming languages, frameworks, and best practices.
<system_constraints>
- You MUST generate the code and files Directly without telling me the implementation plan, just generate the codes and files.
- No C/C++ compiler, native binaries, or Git
- Prefer Node.js scripts over shell scripts
- Use Vite for web servers and Node.js for backend
- Databases: prefer libsql, sqlite, or non-native solutions
- When for react dont forget to write vite config and index.html to the project
- You MUST generate a complete package.json file with valid package release version.
</system_constraints>

{instruction}

Make sure all the files imported are correctly generated, and a complete package.json file with valid package release version exists. Generate the remaining files if needed.

Figure 15: The prompt for aider websites generation.

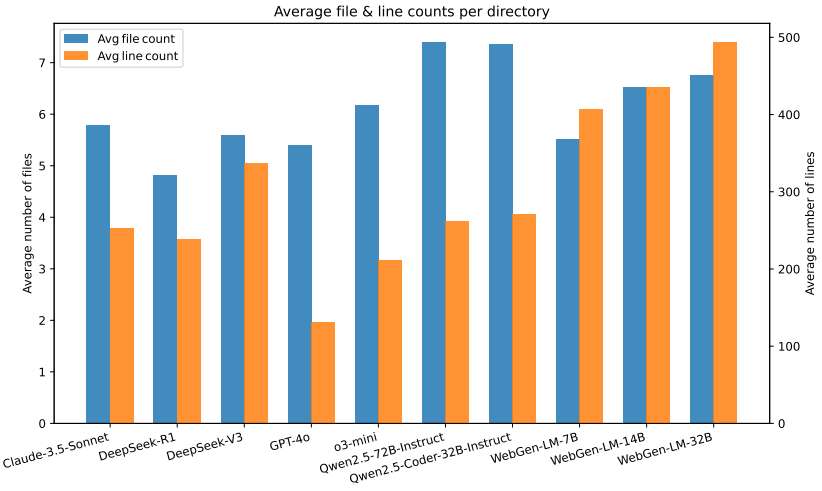


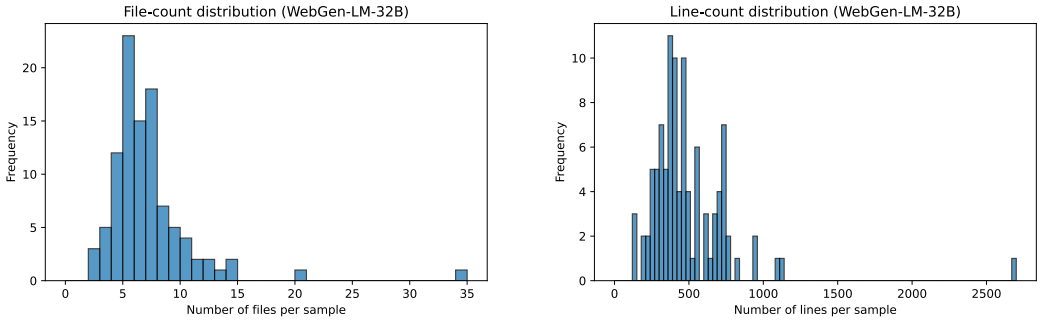
Figure 16: The average file and line counts of each model using Bolt.diy as the framework.

As shown in Fig. 17, most samples contain between 4 and 10 files, while their line counts are largely concentrated between 400 and 500. Only two samples include more than 15 files.

L Test Case Categories

Fig. 18 shows the main category distribution of the task cases. Nearly half of the test cases fall under Functional Testing, around 30% under Data Display Testing, and approximately 20% under Design Validation Testing. This is a reasonable distribution, as functional testing typically constitutes the majority of web page evaluations. Additionally, Tab. 10 presents the detailed subcategories along with their respective frequencies.

Functional testing ensures that all features of an application work as intended. This includes testing form operations such as submission and validation workflows; verifying authentication flows like user registration, login, and permission checks; and validating payment functionalities in e-commerce checkouts or donation processes. It also encompasses search capabilities across various domains such as stock codes, products, or employees, and filtering data based on specific requirements. Additionally, functional testing covers generation tasks such as creating reports or files; file operations including



(a) File-count distribution.

(b) Line-count distribution.

Figure 17: Distributions of the number of files and lines produced by WebGen-LM-32B.

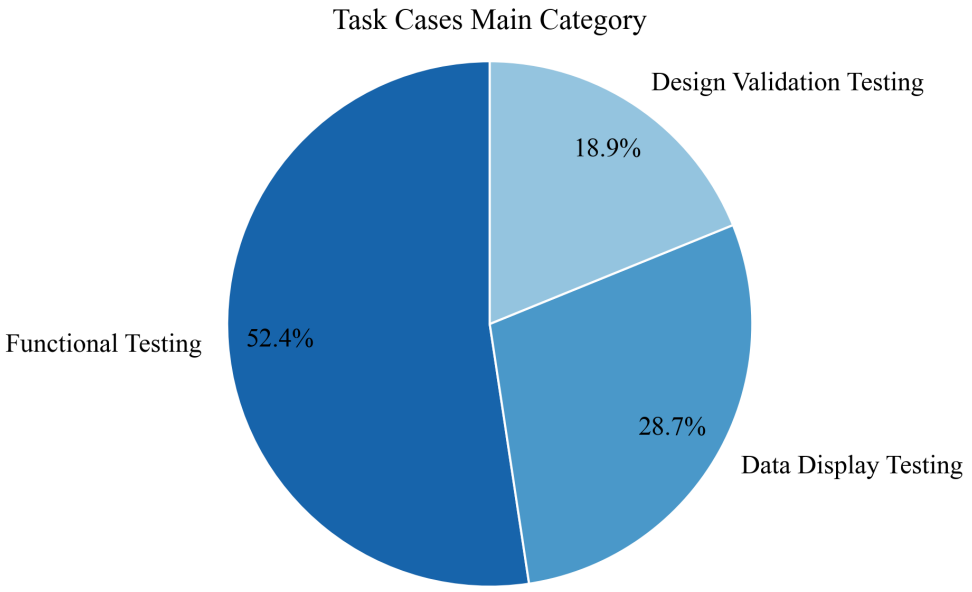


Figure 18: The distribution of the task case categories.

downloading, uploading, and printing; e-commerce activities such as purchasing or booking items; and communication features like sending messages or emails.

Data display testing focuses on how data is presented and updated within an application. This involves ensuring that dynamic content rendering works correctly, including real-time data updates, website navigation, and page refresh mechanisms. It also includes verifying the accuracy of data visualization elements such as charts, graphs, and maps. Furthermore, this type of testing checks the functionality of displaying detailed information when users request more specific data.

Design validation testing focuses on the aesthetic and responsive aspects of an application's user interface. It involves verifying UI consistency across the application and ensuring that color schemes, typography, and spacing are correctly implemented. Responsive behavior is also tested to confirm that the application adapts properly to different devices and screen sizes. Finally, component styling is checked to ensure that elements such as buttons, icons, and cards adhere to the intended design standards.

Table 10: The number of task cases in each category. There are multiple subcategories under each main category. A task case can belong to one main category and multiple subcategories.

Main Categories	Task Number	Sub Category	Task Number
Functional Testing	339	Form Operations	134
		Authentication Flows	48
		Payment	7
		Searching	49
		Filtering	27
		Generation	63
		File Operation	23
		E-commerce	58
		Communication	71
Data Display Testing	186	Dynamic Content Rendering	155
		Data Visualization	30
		Details Information	91
Design Validation Testing	122	UI Consistency	122
		Responsive Behavior	13
		Component Styling	9
Total	667		

M Examples of Websites with Different Appearance Scores

Fig. 19 presents examples of websites with varying appearance scores. As shown in the figure, the visual quality of the websites improves as the appearance score increases. At a score of one, the websites exhibit major rendering errors or contain irrelevant content, whereas at a score of five, the design appears highly harmonious.

N Examples of Websites with Different errors or flaws

Fig. 20 presents the errors or flaws that a generated website may contain. For example, instances (a), (b), and (c) illustrate three types of errors related to website loading failures. Instances (d), (e), and (f) show incomplete websites: instance (d) displays only the background, instance (e) lacks UI components such as buttons, and instance (f) fails to display an image correctly. Additionally, instance (g) is a website that only uses a template without customization; instance (h) shows incorrect placement of webpage content, such as misaligned text; and instance (i) uses an inappropriate background color.

O Examples of UI Agent Testing Processes

In this section, we present examples of UI agent testing trajectories. Fig. 21, Fig. 22, Fig. 23, Fig. 24, and Fig. 25 show examples of test cases that output YES, as the outcome of the operation matches the expected result. Fig. 26, Fig. 27, and Fig. 28 show examples of test cases that output PARTIAL, as the expected result is only partially achieved. Fig. 29, Fig. 30, and Fig. 31 show examples of test cases that output NO, as the website's behavior does not match the expected outcome.

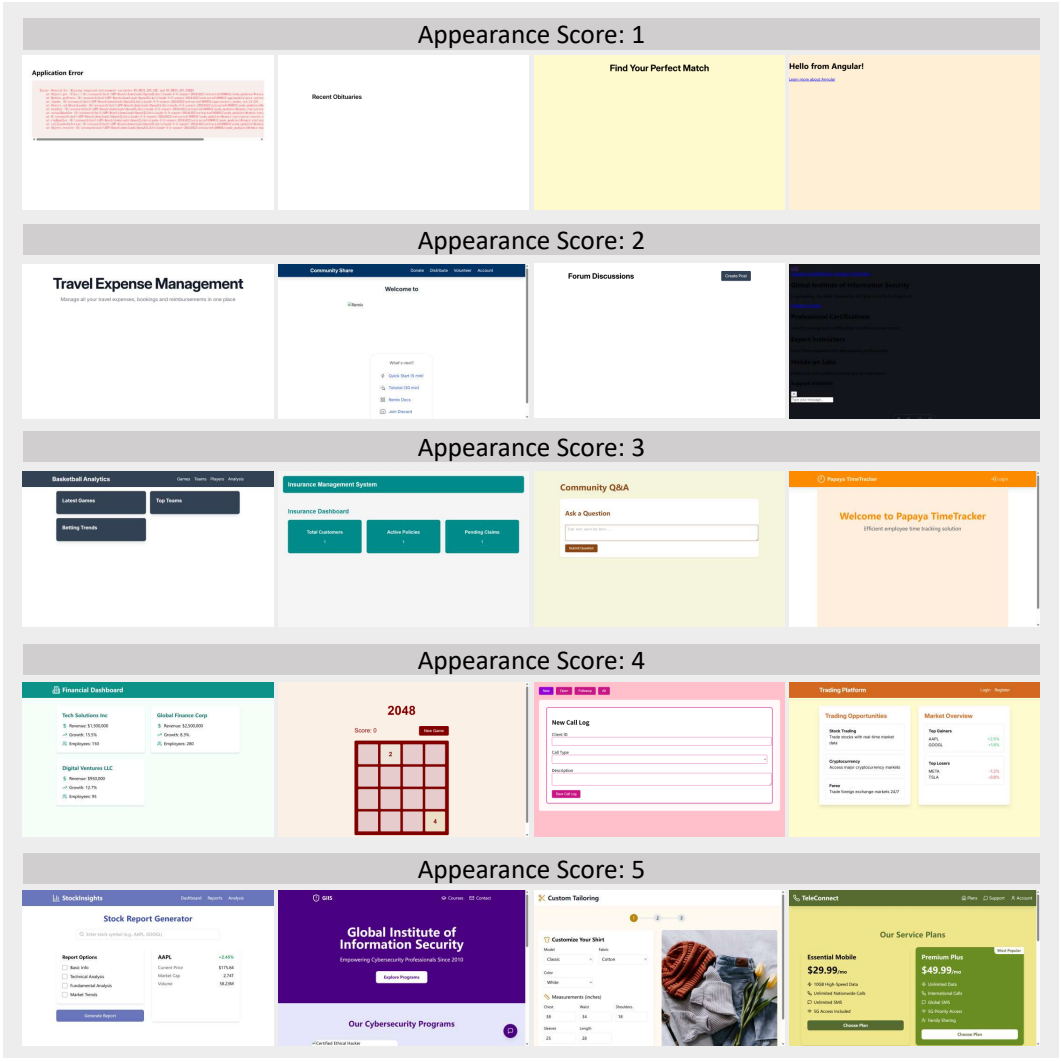


Figure 19: Examples of the screenshots of websites of different appearance scores.

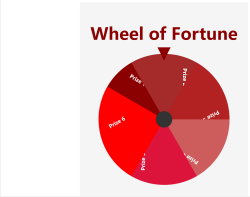
(a) (b) (c)

(d) (e) (f)

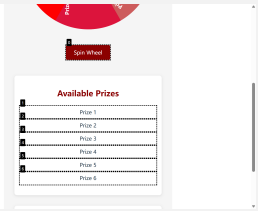
(g) (h) (i)

Figure 20: The examples of errors or flaws that generated webs may include.

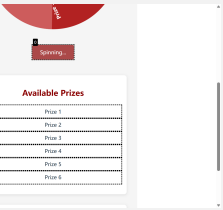
Task: After spinning the wheel, navigate to the section displaying user winning records
Expected Result: The user's winning record is updated to include the latest prize won, with correct details on date and prize.



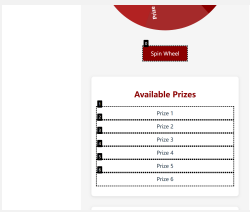
Thought: The current observation shows the "Wheel of Fortune" interface, but there is no visible option to spin the wheel or navigate to the user's winning records. I need to locate the spin button or any navigation link to proceed towards the goal. Action: Scroll WINDOW; down



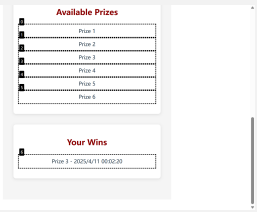
Thought: The "Spin Wheel" button is now visible, allowing me to initiate the spinning process. After spinning, I need to look for a section that displays user winning records to update the latest prize won with correct details. Action: Click [0]



Thought: The wheel is currently spinning, indicating that the spinning process has been initiated. Once the spinning stops, I expect the result to be displayed, and I should then look for a section to view the user's winning records. Action: Wait

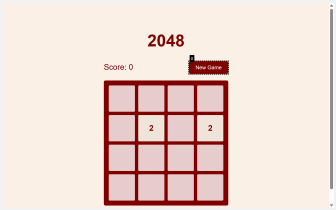


Thought: The spinning process appears to have completed, as the button now reads "Spin Wheel" again. However, there is no visible indication of the result or a section showing user winning records. I need to explore further to find where the winning records might be displayed. Action: Scroll WINDOW; down

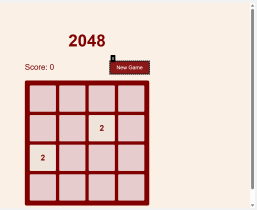


Thought: The "Your Wins" section is now visible, displaying the latest prize won along with the correct date and time. This matches the expected result of updating the user's winning record. Action: ANSWER; YES

Task: Start a New Game
Expected Result: A new game board is initialized with two randomly placed tiles, typically 2s or 4s, and the score is reset to zero.



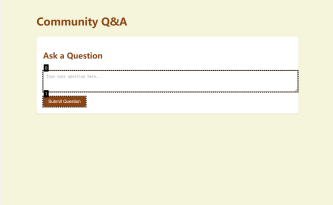
Thought: The current observation shows a 2048 game interface with a "New Game" button visible. The goal is to start a new game, which should initialize the board with two random tiles (typically 2s or 4s) and reset the score to zero. The "New Game" button is clearly labeled and seems to be the appropriate element to interact with to achieve the desired outcome. Action: Click [0]



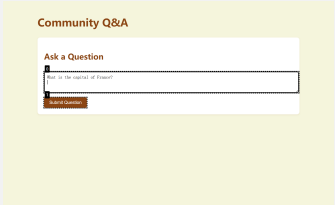
Thought: The current observation shows that the game board has been successfully initialized with two randomly placed tiles (both 2s), and the score is reset to zero. This matches the expected result of starting a new game. Action: ANSWER; YES

Figure 21: The examples of UI agent testing processes resulting in YES.

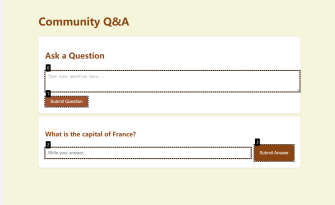
Task: Submit a new question using the question submission form.
Expected Result: The question is successfully posted and appears on the list of submitted questions.



Thought: The webpage provides a clear interface for submitting a question. There is a textarea labeled with the number 0 where the question can be typed, and a button labeled with the number 1 to submit the question. To achieve the goal, I need to type a question into the textarea and then click the submit button. Action: Type [0]; What is the capital of France?

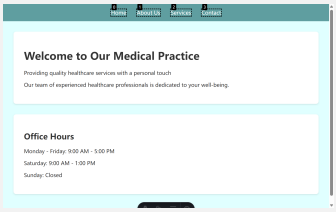


Thought: The question has been successfully typed into the textarea. The next step is to submit the question by clicking the "Submit Question" button, which is labeled with the number 1. Action: Click [1]

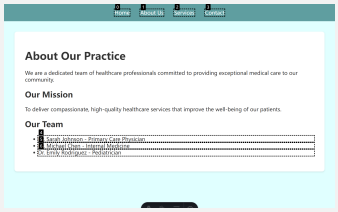


Thought: The question "What is the capital of France?" has been successfully posted and now appears on the list of submitted questions. This indicates that the expected result of submitting a new question has been fully achieved. Action: ANSWER; YES

Task: Click on the "About Us" page link in the navigation menu.
Expected Result: The "About Us" page loads, providing accurate information about the clinical office, such as its history, mission, and team, without any broken links or missing images.

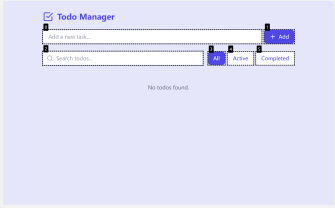


Thought: The goal is to navigate to the "About Us" page. The navigation menu is visible at the top of the page, and the "About Us" link is clearly labeled with the number 1. Clicking this link should take me to the desired page. Action: Click [1]

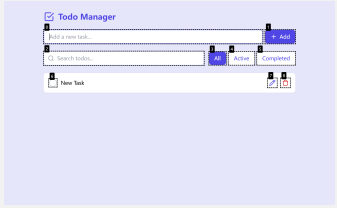


Thought: The "About Us" page has loaded successfully, displaying information about the practice, including its mission and team members. There are no broken links or missing images visible, indicating that the page is functioning correctly. Action: ANSWER; YES

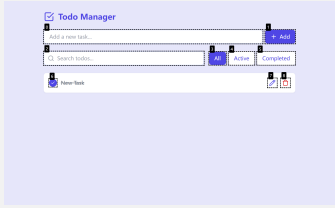
Task: Mark a specific task as completed on the to-do list.
Expected Result: The task is visibly marked as completed, often indicated by a strikethrough, checkbox, or similar visual cue.



Thought: There are no tasks listed, indicating that I need to add a new task before marking it as completed. The next logical step is to add a task by interacting with the input field and the "Add" button. Action: Type [0]; New Task



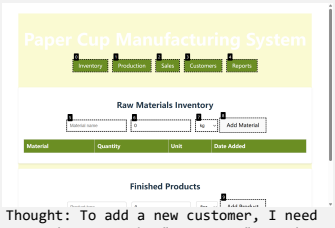
Thought: The task has been successfully added to the list, as indicated by the presence of "New Task" in the list. To mark this task as completed, I need to interact with the checkbox next to the task. Action: Click [6]



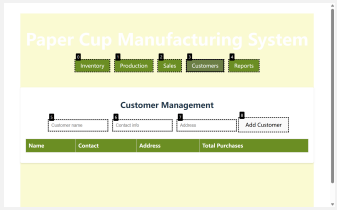
Thought: The task "New Task" has been successfully marked as completed, as indicated by the checked checkbox and the visual styling change. This matches the expected result of marking a task as completed. Action: ANSWER; YES

Figure 22: The examples of UI agent testing processes resulting in YES.

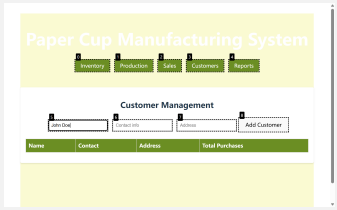
Task: Use the customer management functionality to add a new customer and view their information.
Expected Result: The new customer appears in the customer list with the correct details provided during registration and the system confirms a successful addition.



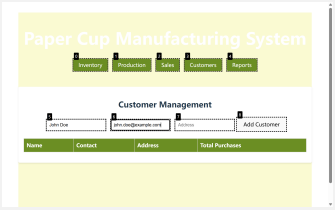
Thought: To add a new customer, I need to navigate to the "Customers" section. The button labeled "Customers" (numerical label 3) seems to be the appropriate choice to access the customer management functionality. Action: Click [3]



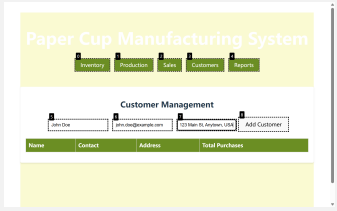
Thought: I am now on the Customer Management page. To add a new customer, I need to fill in the required fields: Customer name, Contact info, and Address. Then, I should click the "Add Customer" button to submit the information. Action: Type [5]; John Doe



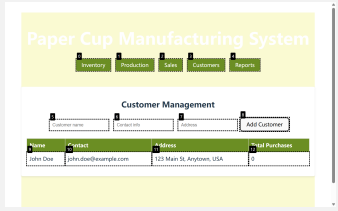
Thought: I have entered the customer's name. Next, I need to fill in the contact information and address fields before clicking the "Add Customer" button to submit the form. Action: Type [6]; john.doe@example.com



Thought: I have filled in the customer's name and contact information. The next step is to enter the address before submitting the form. Action: Type [7]; 123 Main St, Anytown, USA

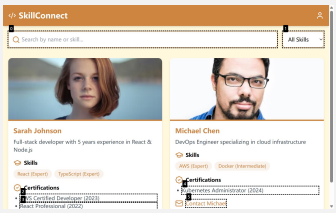


Thought: I have filled in all the required fields for adding a new customer. The next logical step is to click the "Add Customer" button to submit the form and add the customer to the system. Action: Click [8]

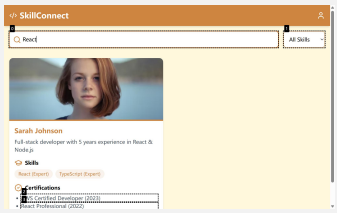


Thought: The new customer has been successfully added to the customer list with the correct details provided during registration. The system confirms a successful addition by displaying the customer's information in the table. Action: ANSWER; YES

Task: Test the search feature by searching for a user based on a specific skill.
Expected Result: The search results should display a list of users who have the specified skill in their profiles.



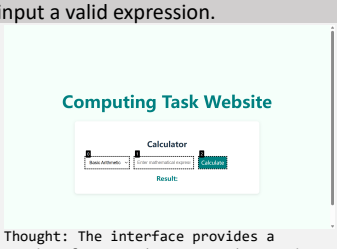
Thought: The goal is to test the search feature by searching for users based on a specific skill. The search bar is visible and ready for input. I will start by entering a skill into the search bar to see if the expected results are displayed. Action: Type [0]; React



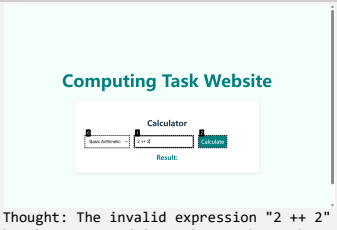
Thought: After entering "React" into the search bar, the search results show a single user, Sarah Johnson, who has React listed as a skill. This indicates that the search feature is working correctly for this specific skill. Action: ANSWER; YES

Figure 23: The examples of UI agent testing processes resulting in YES.

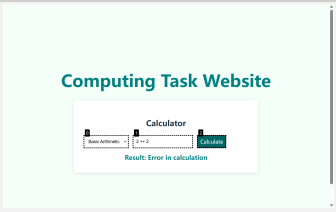
Task: Input an invalid mathematical expression (e.g., "2 ++ 2") and attempt to execute the calculation.
Expected Result: An error message is displayed, indicating that the expression is invalid and guiding the user to input a valid expression.



Thought: The interface provides a textbox for entering expressions and a button to calculate. I will start by entering the invalid expression "2 ++ 2" into the textbox. Action: Type [1]; 2 ++ 2

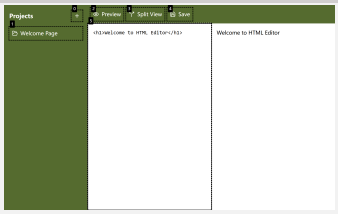


Thought: The invalid expression "2 ++ 2" has been entered into the textbox. The next step is to attempt the calculation by clicking the "Calculate" button to see if an error message is displayed. Action: Click [2]

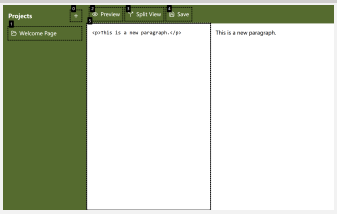


Thought: The expected result of displaying an error message for the invalid expression "2 ++ 2" has been achieved. The website shows "Error in calculation," which indicates that the task's objective has been met. Action: ANSWER; YES

Task: Enter HTML code in the editor and view the live preview of the content.
Expected Result: The live preview immediately updates to accurately display the rendered HTML content as it is typed, without delay or errors.

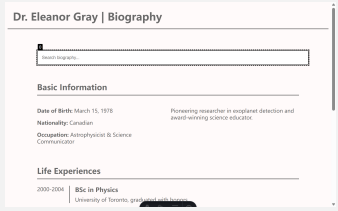


Thought: The goal is to ensure that any changes made in the HTML code are reflected instantly in the live preview. Since the live preview already shows the rendered content of the HTML code, I need to test if modifications in the code update the preview in real-time. Action: Type [5]; This is a new paragraph.

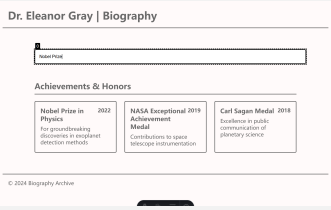


Thought: After entering the new HTML code `<p>This is a new paragraph.</p>` in the editor, the live preview has updated correctly to display the rendered paragraph. This indicates that the live preview functionality is working as expected, updating in real-time without delay or errors. Action: ANSWER; YES

Task: Use the search function to look up a specific keyword related to the person's biography (e.g., "Nobel Prize").
Expected Result: The search results should display relevant sections or pages containing the keyword, enabling easy access to detailed information.



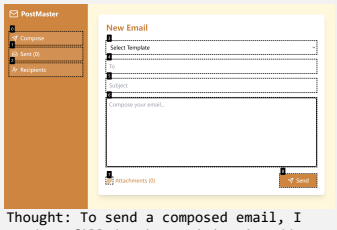
Thought: The page displays a search bar labeled with the number 0, which is likely intended for searching within the biography. To achieve the task, I need to enter the keyword "Nobel Prize" into this search box and trigger the search function. Action: Type [0]; Nobel Prize



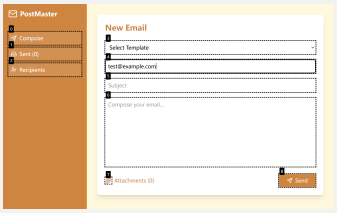
Thought: The search for "Nobel Prize" has successfully returned relevant results, displaying Dr. Eleanor Gray's Nobel Prize in Physics achievement. This matches the expected result of finding sections or pages containing the keyword. Action: ANSWER; YES

Figure 24: The examples of UI agent testing processes resulting in YES.

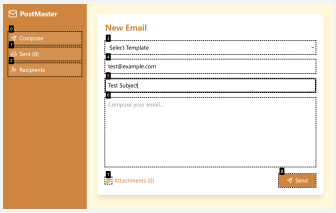
Task: Send a composed email to the selected recipients.
Expected Result: The email is successfully sent, a confirmation message is received, and the email appears in the sent emails folder.



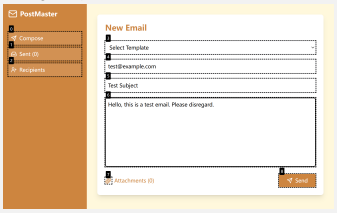
Thought: To send a composed email, I need to fill in the recipient's address, subject, and body of the email, and then click the "Send" button. The current observation shows the "New Email" form where I can enter this information. Action: Type [4]; test@example.com



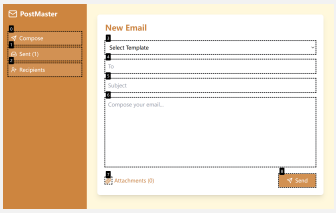
Thought: I have entered the recipient's email address. Next, I need to fill in the subject line and the body of the email before sending it. Action: Type [5]; Test Subject



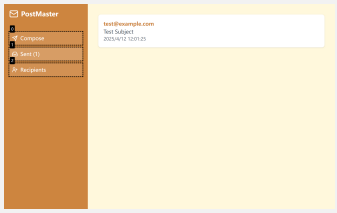
Thought: I have filled in the recipient's email address and the subject line. The next step is to compose the body of the email. Action: Type [6]; Hello, this is a test email. Please disregard.



Thought: I have completed composing the email by filling in the recipient's address, subject, and body. The next logical step is to send the email. Action: Click [8]

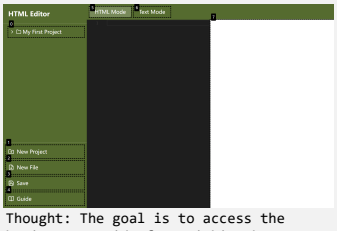


Thought: The email has been successfully sent, as indicated by the updated count in the "Sent" section showing "Sent (1)." This suggests that the email was sent correctly, fulfilling part of the task. However, I need to verify if a confirmation message is received and if the email appears in the "Sent" folder. Action: Click [1]

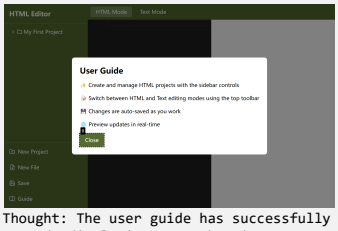


Thought: The email appears in the "Sent" folder, confirming that it was successfully sent. Since the task required sending a composed email to the selected recipients, and the email is now visible in the "Sent" folder, the task seems to be fully achieved. Action: ANSWER; YES

Task: Access the basic user guide from within the HTML editor interface.
Expected Result: The editor provides access to a basic user guide, which opens correctly and provides comprehensive instructions relevant to using the editor's features.



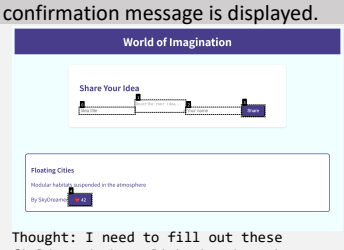
Thought: The goal is to access the basic user guide from within the HTML editor interface. Based on the observation, there is a button labeled "Guide" (numerical label 4), which seems to be the most likely candidate for accessing the user guide. Clicking this button should lead to the desired outcome. Action: Click [4]



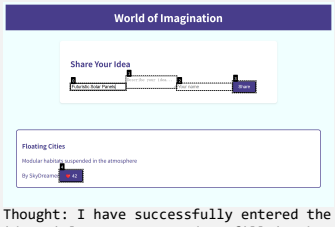
Thought: The user guide has successfully opened, displaying comprehensive instructions relevant to using the editor's features. This matches the expected result of accessing the basic user guide. Action: ANSWER; YES

Figure 25: The examples of UI agent testing processes resulting in YES.

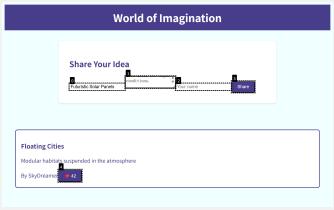
Task: Submit a new idea via the share ideas form.
Expected Result: The form submits successfully, the new idea appears in the shared ideas section, and a confirmation message is displayed.



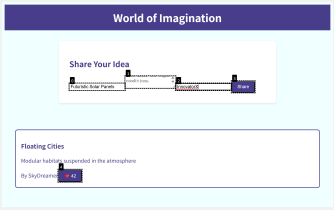
Thought: I need to fill out these fields and then click the Share button to submit the idea. Since this is the first step, I'll start by entering content into the Idea title field. Action: Type [0]; Futuristic Solar Panels



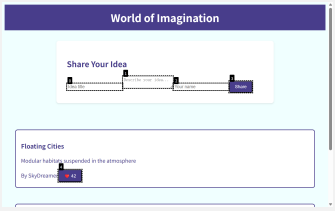
Thought: I have successfully entered the idea title. Next, I need to fill in the description of the idea in the second textbox. This will allow me to proceed with submitting the form. Action: Type [1]; Highly efficient solar panels that can generate energy even under low light conditions.



Thought: I have filled in the idea title and description. The next logical step is to enter my name in the third textbox so that I can proceed to submit the form. Action: Type [2]; InnovatorX

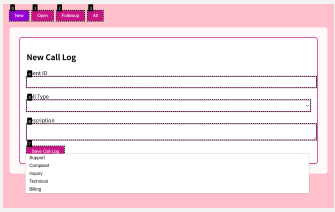


Thought: All required fields in the form have been filled out. The next step is to submit the form by clicking the "Share" button. This should trigger the submission process, leading to the display of the new idea in the shared ideas section and a confirmation message. Action: Click [3]

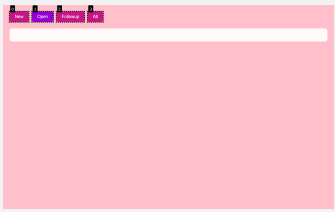


Thought: The form has been submitted successfully, as indicated by the reset state of the form fields. However, I do not see any confirmation message or the newly shared idea appearing in the shared ideas section. This suggests that while the form submission was processed, the expected result of displaying the new idea and showing a confirmation message was not fully achieved. Action: ANSWER; PARTIAL

Task: Check the navigation to view open calls
Expected Result: Open calls are displayed in a list format, and the page loads without errors.



Thought: The goal is to check the navigation to view open calls. The current observation shows a form for creating a new call log, with buttons labeled "New," "Open," "Followup," and "All." To view open calls, I should click the "Open" button, as it likely filters the calls to display only those that are open. Action: Click [1]

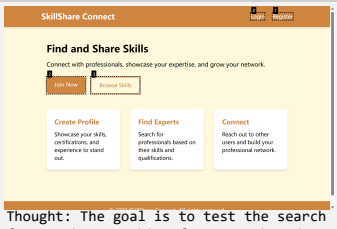


Thought: After clicking the "Open" button, the page appears to have loaded without any visible content or errors. However, since there is no visible list of open calls, it might indicate that either there are no open calls available or the page requires further interaction to display the list. Action: Scroll WINDOW; down

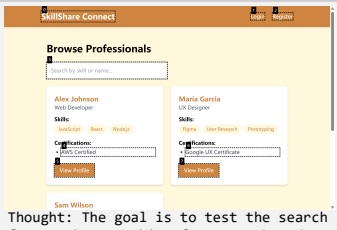
Thought: After scrolling down, the page still does not show any content related to open calls. This indicates that either there are no open calls available, or the page requires additional interaction to load the content. Since the page has loaded without errors and the navigation seems functional, but the expected list of open calls is not visible, the result is partially achieved. Action: ANSWER; PARTIAL

Figure 26: The examples of UI agent testing processes resulting in PARTIAL.

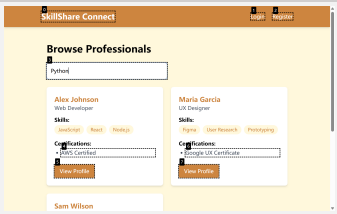
Task: Test the search feature by searching for a user based on a specific skill.
Expected Result: The search results should display a list of users who have the specified skill in their profiles.



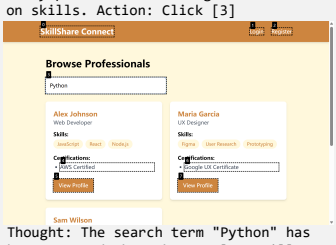
Thought: The goal is to test the search feature by searching for users based on a specific skill. The current page provides options to either join now or browse skills. To proceed with the search, I should click on the "Browse Skills" button, which seems to be the entry point for searching users based on skills. Action: Click [3]



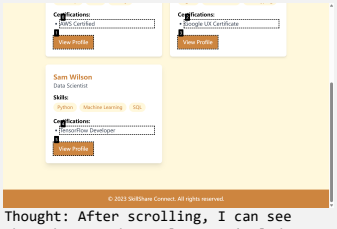
Thought: The goal is to test the search feature by searching for users based on a specific skill. The current page shows a search bar where I can enter a skill or name. To proceed, I should type a specific skill into the search bar and see if the expected result is achieved. Action: Type [3]; Python



Thought: The search term "Python" has been entered into the search bar, but the results do not yet reflect this search. I need to trigger the search process to see the updated results. Action: Wait

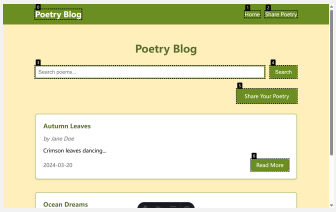


Thought: The search term "Python" has been entered, but the results still show professionals without Python as a listed skill. Since the expected result is not achieved, I need to determine if further interaction is required or if the issue lies with the platform itself. Action: Scroll WINDOW; down



Thought: After scrolling, I can see that the search results now include a user named Sam Wilson, who has "Python" listed as one of their skills. However, the initial results did not reflect the search query, suggesting that the search might not be updating dynamically or filtering correctly. Action: ANSWER; PARTIAL

Task: Verify the poetry browsing functionality.
Expected Result: The user can access a list of poetry, with each entry displaying the title and author clearly.



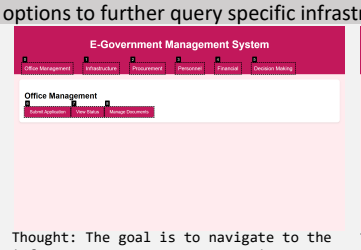
Thought: The current observation shows a webpage titled "Poetry Blog," which displays a list of poetry entries. Each entry includes the title, author, and a snippet of the poem. There is a "Read More" button next to each entry, suggesting further details can be accessed. The goal is to verify the poetry browsing functionality, ensuring users can view a list of poems with titles and authors. Action: Click [6] to explore the details of the "Autumn Leaves" poem and verify the browsing functionality.

Thought: The current observation indicates a 404 Not Found error page, meaning the requested resource (in this case, the poem details) is unavailable. This suggests that while the list of poetry entries is visible, attempting to access detailed information about a specific poem leads to an error. This impacts the ability to fully verify the poetry browsing functionality as expected. Action: GoBack

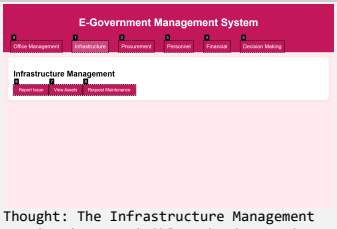
Thought: Upon returning to the main page, the poetry browsing functionality appears to work partially. The list of poetry entries is visible, showing titles and authors, which aligns with part of the expected result. However, attempting to access detailed views of individual poems resulted in a 404 error, indicating incomplete functionality. Action: ANSWER; PARTIAL

Figure 27: The examples of UI agent testing processes resulting in PARTIAL.

Task: Navigate to the infrastructure management section and browse through the available information.
Expected Result: User can view detailed and accurate information related to infrastructure management, with options to further query specific infrastructure details.

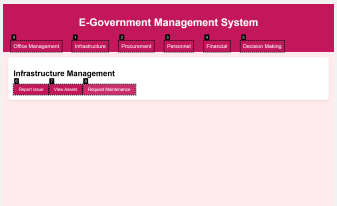


Thought: The goal is to navigate to the infrastructure management section. Currently, the page is displaying the Office Management section. To proceed, I need to click on the "Infrastructure" tab to access the relevant information. Action: Click [1]



Thought: The Infrastructure Management section is now visible, showing options such as "Report Issue," "View Assets," and "Request Maintenance." Since the goal is to browse through the available information, clicking on any of these buttons would provide further details. Action: Click [7]

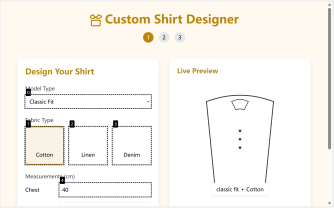
Thought: The Infrastructure Management section is currently active, and the "View Assets" button has been clicked. This should display detailed information about the assets under infrastructure management. Since the goal is to browse through the available information, this action aligns well with the objective. Action: Click [8]



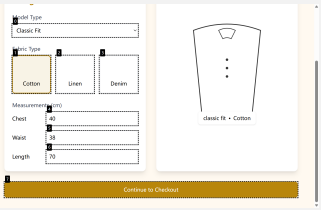
Thought: The Infrastructure Management section is still active, and the "Request Maintenance" button has been clicked. This option allows users to request maintenance services for infrastructure assets. Since the goal is to browse through the available information, interacting with this button provides additional functionality related to infrastructure management. Action: Click [8]

Thought: The Infrastructure Management section is still active, and the "Request Maintenance" button has been clicked again. However, since the goal is to browse through the available information, and the current state shows no new content or changes, it seems that the browsing aspect has been completed. Action: ANSWER; PARTIAL

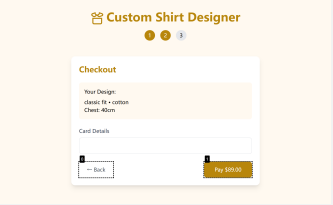
Task: Fill in shipping and payment information on the checkout page and submit the order.
Expected Result: The system should accept valid credit card information, process the order without errors, and provide an order confirmation message.



Thought: Since the goal is to fill in shipping and payment details on the checkout page, I need to navigate away from the current page. There might be a button or link leading to the checkout page, but it's not visible in the current screenshot. I will attempt to scroll to see if the checkout option appears. Action: Scroll WINDOW; down



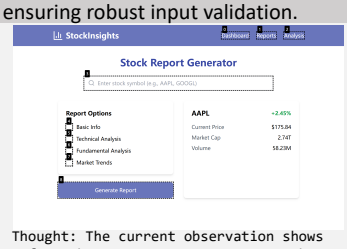
Thought: The current observation shows the design page with options for selecting the model type, fabric type, and entering measurements. At the bottom, there is a prominent button labeled "Continue to Checkout," which is likely the next step to proceed to the checkout page where shipping and payment information can be entered. Clicking this button should take me to the desired checkout page. Action: Click [7]



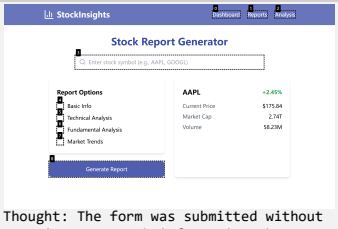
Answer: PARTIAL Explanation: During the interactions, I attempted to fill in the card details as required. However, due to repeated failures in executing the action to input the card number, I was unable to proceed further to submit the order and receive an order confirmation message. Therefore, the expected result was only partially achieved as the card details were not successfully entered.

Figure 28: The examples of UI agent testing processes resulting in PARTIAL.

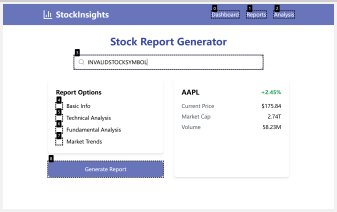
Task: Evaluate the website's form inputs by submitting incomplete or incorrect stock information.
Expected Result: The system displays appropriate error messages and guidance on how to correct the input errors, ensuring robust input validation.



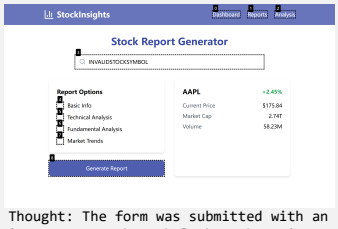
Thought: The current observation shows a form where users can enter a stock symbol and generate a report. To test the input validation, I'll start by attempting to submit the form without entering any stock information. Action: Click [8]



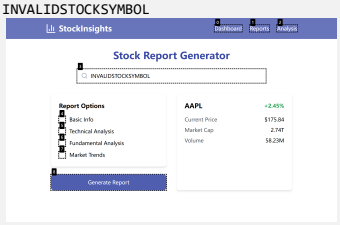
Thought: The form was submitted without entering any stock information, but there doesn't seem to be any visible error message or feedback indicating that the input was invalid. To further test the input validation, I'll attempt to enter an incorrect stock symbol and submit the form again. Action: Type [3]; INVALIDSTOCKSYMBOL



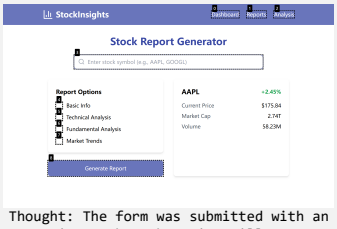
Thought: An incorrect stock symbol has been entered into the input field. The next step is to submit the form to see if the system provides appropriate error messages or guidance for correcting the input error. Action: Click [8]



Thought: The form was submitted with an incorrect stock symbol, but there is still no visible error message or feedback indicating that the input was invalid. Since no error messages are shown, I will attempt another submission with a completely empty input to see if the system responds differently. Action: Click [8]

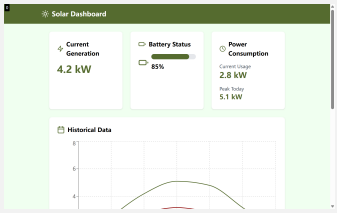


Thought: The form was submitted with an incorrect stock symbol, but there is still no visible error message or feedback indicating that the input was invalid. Since no error messages are shown, I will attempt another submission with a completely empty input to see if the system responds differently. Action: Type [3]; ""

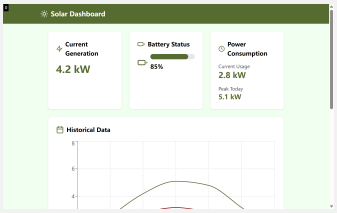


Thought: The form was submitted with an empty input, but there is still no visible error message or feedback indicating that the input was invalid. Since no error messages are shown even with an empty input, I will conclude that the expected result of displaying appropriate error messages and guidance is not being achieved. Action: ANSWER; NO

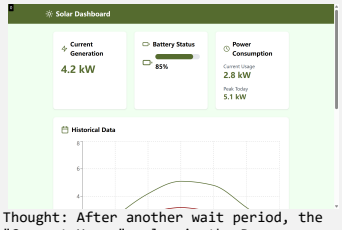
Task: Test the display of real-time power consumption data.
Expected Result: The dashboard should present a real-time updating figure or chart that accurately reflects the current power consumption levels.



Thought: The Power Consumption section displays "Current Usage" as 2.8 kW and "Peak Today" as 5.1 kW. There is also a historical data graph below. To test the real-time power consumption data, I need to observe if the "Current Usage" value updates over time. Since this is the initial observation, I will wait to see if the value changes. Action: Wait



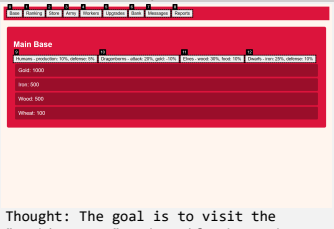
Thought: After waiting, the "Current Usage" value in the Power Consumption section remains at 2.8 kW, indicating no change from the previous observation. To determine if the data is updating in real-time, I need to continue monitoring the value over time. Action: Wait



Thought: After another wait period, the "Current Usage" value in the Power Consumption section still shows 2.8 kW, with no indication of real-time updates. Since the value has not changed across multiple observations, it suggests that the real-time data is not being updated as expected. Action: ANSWER; NO

Figure 29: The examples of UI agent testing processes resulting in NO.

Task: Visit the "Ranking Page" and verify the rank listing of players
Expected Result: The page should display a sorted list of players by rank, with accurate and clear presentation relevant to their performance metrics.

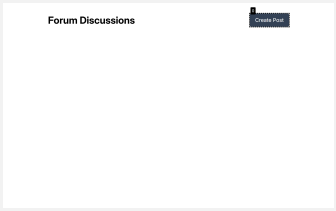


Thought: The goal is to visit the "Ranking Page" and verify the rank listing of players. Currently, we are on the "Main Base" page. To proceed, I need to click on the "Ranking" button, which is labeled as [1]. Action: Click [1]

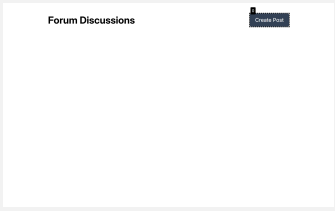
Thought: After clicking the "Ranking" button, the page appears to be loading or has navigated to a blank state. There are no visible rankings or player details present. Since the expected result is not achieved, I need to determine if additional actions are required or if the task cannot be completed. Action: Wait

Thought: After waiting, the page still shows no content related to the ranking of players. Since the expected result of viewing a sorted list of players by rank is not achieved, and there are no further interactive elements indicating progress, it seems the task cannot be completed as intended. Action: ANSWER; NO

Task: Test the search feature by entering a specific keyword like health.
Expected Result: Search results page displays posts containing the health keyword, and results are relevant to the query.



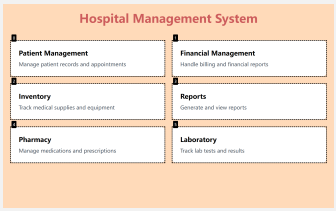
Thought: Since the goal is to test the search feature, I need to locate a search bar or a way to trigger its appearance. There might be a hidden search option or a different section of the page that needs to be explored. Action: Scroll WINDOW; down



Thought: After scrolling down, the page still does not display a search bar or any interactive elements related to searching. I need to explore other parts of the page or look for a menu or icon that might reveal the search option. Action: Scroll WINDOW; down

Thought: Despite multiple attempts to scroll, the search bar or any relevant search-related elements are still not visible. Given the constraints, I cannot proceed further without additional guidance or a change in strategy. Action: ANSWER; NO

Task: Navigate to the pharmacy management section and update the inventory count for a specific medicine.
Expected Result: The inventory count is correctly updated and reflected in the system's current stock records.

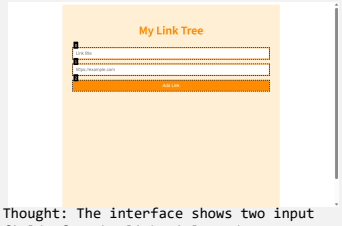


Thought: The goal is to navigate to the pharmacy management section to update the inventory count for a specific medicine. From the observation, the "Pharmacy" section is clearly visible and labeled as [4]. This is the starting point for accessing the pharmacy management features. Action: Click [4]

Thought: The attempt to access the pharmacy management section resulted in a 404 Not Found error, indicating that the requested page does not exist or cannot be accessed. Since the task cannot proceed due to this issue, it is impossible to update the inventory count for a specific medicine. Action: ANSWER; NO

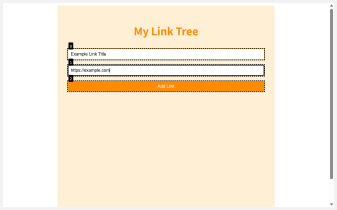
Figure 30: The examples of UI agent testing processes resulting in NO.

Task: Test the link addition functionality by adding a new link to the user's link tree.
Expected Result: The newly added link appears in the user's link tree with the inputted title and URL.



Thought: The interface shows two input fields for the link title and URL, followed by an "Add Link" button. I need to fill in the input fields with appropriate values and then click the "Add Link" button to see if the new link appears in the link tree. Action: Type [0]; Example Link Title

Thought: I have successfully entered the link title. Next, I need to enter the URL in the second input field to proceed with adding the link. Action: Type [1]; https://example.com



Thought: Both the link title and URL have been entered correctly. The next step is to click the "Add Link" button to submit the new link and observe if it appears in the link tree. Action: Click [2]

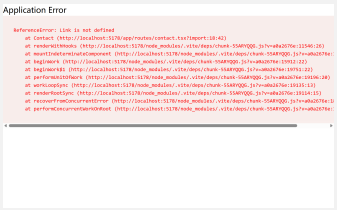
Thought: The "Add Link" button has been clicked, indicating that the link addition process has been initiated. I need to verify if the new link appears in the link tree to determine if the expected result was achieved. Action: Wait

Thought: The "Add Link" button has been clicked, but there is no visible indication that the new link has been added to the link tree. I need to re-evaluate the process to ensure the expected result is achieved. Action: Click [2]

Thought: The "Add Link" button has been clicked again, but there is still no visible indication that the new link has been added to the link tree. Since the expected result has not been achieved, I need to conclude the testing process. Action: ANSWER; NO

Task: Access and fill out the contact form.
Expected Result: The contact form allows the user to submit their name, email, and message, with successful submission confirmed by an acknowledgment message.

Thought: To access the contact form, I need to click on the "Contact Us" button, which is clearly visible on the screen. This aligns with the goal of filling out the contact form. Action: Click [5]



Thought: The application encountered an error when attempting to access the contact form, indicating that the expected result cannot be achieved due to this issue. Action: ANSWER; NO

Figure 31: The examples of UI agent testing processes resulting in NO.