

# WEBGEN-AGENT: ENHANCING INTERACTIVE WEBSITE GENERATION WITH MULTI-LEVEL FEEDBACK AND STEP-LEVEL REINFORCEMENT LEARNING

006 **Anonymous authors**

007 Paper under double-blind review

## ABSTRACT

013 Agent systems powered by large language models (LLMs) have demonstrated  
 014 impressive performance on repository-level code-generation tasks. However, for  
 015 tasks such as website codebase generation, which depend heavily on visual effects  
 016 and user-interaction feedback, current code agents rely only on simple code execu-  
 017 tion for feedback and verification. This approach fails to capture the actual quality  
 018 of the generated code. In this paper, we propose *WebGen-Agent*, a novel website-  
 019 generation agent that leverages comprehensive and multi-level visual feedback  
 020 to iteratively generate and refine the website codebase. Detailed and expressive  
 021 text descriptions and suggestions regarding the screenshots and GUI-agent test-  
 022 ing of the websites are generated by a visual language model (VLM), together  
 023 with scores that quantify their quality. The screenshot and GUI-agent scores are  
 024 further integrated with a backtracking and select-best mechanism, enhancing the  
 025 performance of the agent. Utilizing the accurate visual scores inherent in the  
 026 WebGen-Agent workflow, we further introduce *Step-GRPO with Screenshot and*  
 027 *GUI-Agent Feedback* to improve the ability of LLMs to act as the agent-engine  
 028 model. By using the screenshot and GUI-agent scores at each step as the reward in  
 029 Step-GRPO, we provide a dense and reliable process supervision signal, which ef-  
 030 fectively improves the model’s website-generation ability. On the WebGen-Bench  
 031 dataset, WebGen-Agent increases the accuracy of Claude 3.5 Sonnet from 26.4%  
 032 to 51.9% and its appearance score from 3.0 to 3.9, outperforming the previous  
 033 state-of-the-art agent system. Additionally, our training approach increases the  
 034 accuracy of Qwen2.5-Coder-7B-Instruct from 34.8% to 45.4% and raises the ap-  
 035 pearance score from 3.4 to 3.7.

## 1 INTRODUCTION

038 Recent studies on code agents have shown great advancements in repository-level code-generation  
 039 tasks, such as fixing GitHub issues (Yang et al., 2024b) and implementing new features (Miserendino  
 040 et al., 2025). However, for tasks like website code generation, which depend heavily on visual aes-  
 041 thetics and the fluency of user interactions, current code-agent systems fail to fully capture the  
 042 actual quality of the generated code base, because they mostly rely on simple code-execution feed-  
 043 back. This limitation can lead to various rendering and functional problems in the generated web  
 044 applications, such as misaligned components, disharmonious coloring, unresponsive buttons, and  
 045 broken links.

046 To enable the code agent to effectively handle such tasks, we introduce **WebGen-Agent**, a code-  
 047 generation system that generates websites from natural-language instructions that specify appear-  
 048 ance and functional requirements, thus offering a highly automated website-development process.  
 049 To ensure that the generated websites meet both functional requirements and aesthetic standards,  
 050 we leverage both execution feedback and visual feedback to refine the project. Specifically, we  
 051 leverage a visual language model (VLM) to assess the visual appeal and aesthetic quality of the cur-  
 052 rent website, and a Graphical User Interface (GUI) agent to evaluate the correctness and intended  
 053 functionality of the website’s code base, thereby gathering accurate information and providing tar-  
 054 geted suggestions and reflections. By iteratively applying this feedback and editing the code base,  
 055 WebGen-Agent builds websites with appealing designs and smooth interactive functionality.

054 As shown in Fig. 1, WebGen-Agent adopts an iterative, multi-step paradigm in which each step  
 055 consists of three actions: code generation, code execution, and feedback gathering. The agent begins  
 056 each step by creating and editing files in the code base in a manner similar to Bolt.diy (stackblitz  
 057 labs, 2024). During code execution, dependencies are installed and the website service is started. If  
 058 execution emits errors, the errors are returned to the agent, which starts the next step to fix them. If  
 059 five consecutive error steps occur, the agent backtracks to a previous non-erroneous step.

060 In the feedback-gathering process, a screenshot of the website is first captured. A VLM then pro-  
 061 vides a description and an appearance score based on the screenshot. If the screenshot has room  
 062 for improvement, the model supplies suggestions, and the next step is undertaken to implement  
 063 them, thereby explicitly refining the website’s visual aesthetics. Otherwise, a GUI-agent session  
 064 is initiated to explore the website, which evaluates the functional requirements and generates cor-  
 065 responding feedback. If the testing is successful, the task is complete; otherwise, suggestions for  
 066 fixing the website are generated, and the agent continues to edit the code base in the next step. At  
 067 the end of the task trajectory, the best step is selected on the basis of the screenshot and GUI-agent  
 068 scores, and the code base is restored to the state of that step. Based on the pipeline, various models  
 069 achieve better performance on WebGen-Bench (Lu et al., 2025b), consistently outperforming other  
 070 code agents. Remarkably, Claude-3.5-Sonnet improved its accuracy from 26.4% to 51.9% and its  
 071 appearance score from 3.0 to 3.9, outperforming Bolt.diy.

072 To equip code agents with enhanced reasoning ability, we further propose **Step-GRPO with Screen-**  
 073 **shot and GUI-agent Feedback.** As shown in Fig. 2, given an instruction, multiple WebGen-Agent  
 074 trajectories are generated. Each step is accompanied by a screenshot score and a GUI-agent testing  
 075 score, and an accurate and reliable step-level reward can be computed by summing these two scores.  
 076 This dual supervision of website appearance and functionality effectively optimizes the model to  
 077 generate high-quality website codebases, providing stepwise, process-level guidance for the agent  
 078 trajectory. Training a Qwen2.5-Coder-7B-Instruct model with this approach increases the accuracy  
 079 from 34.8% to 45.4% and raises the appearance score from 3.4 to 3.7 on WebGen-Bench, greatly  
 080 improving both the functionality and the appearance of the generated websites. We name the trained  
 081 family of models **WebGenAgent-LM**.

082 Our contributions include:

- 083 • We propose WebGen-Agent, a code-agent system that leverages screenshots and GUI-agent test-  
 084 ing to provide reflection signals and iteratively improve the quality of generated websites.
- 085 • We introduce Step-GRPO with Screenshot and GUI-agent Feedback, which uses screenshots and  
 086 GUI-agent scores as step-level supervision in the GRPO training process, significantly improving  
 087 the performance of smaller open-source models.
- 088 • Extensive experiments demonstrate the effectiveness of the proposed method. The system in-  
 089 creases the accuracy of Claude-3.5-Sonnet from 26.4% to 51.9% and its appearance score  
 090 from 3.0 to 3.9, outperforming Bolt.diy. And our training approach increases the accuracy of  
 091 Qwen2.5-Coder-7B-Instruct from 34.8% to 45.4% and raises the appearance score from 3.4 to  
 092 3.7.

## 094 2 METHOD

096 In this section, we first introduce WebGen-Agent, a novel website generation system that leverages  
 097 screenshots and GUI-agent testing as reliable feedback to iteratively refine both the appearance and  
 098 functionality of the generated website. Building on the dense, reliable visual scores produced by  
 099 WebGen-Agent, we then propose Step-GRPO with Screenshot and GUI-Agent Feedback, a method  
 100 that uses these scores to provide process supervision during GRPO training. This approach signifi-  
 101 cantly enhances the model’s ability to generate high-quality websites.

### 103 2.1 WEBGEN-AGENT WORKFLOW

105 The WebGen-Agent workflow consists of multiple steps: each step includes code generation, code  
 106 execution, and feedback gathering. As shown in Fig. 1, the agent trajectory starts from a website  
 107 generation instruction ( $\mathcal{I}$ ), denoted as  $\mathcal{T} = [\mathcal{I}]$ , and an empty code base  $\mathcal{C}_0$ . The agent-engine  
 LLM generates code  $\Delta\mathcal{C}_1$  to edit the code base, resulting in  $\mathcal{C}_1$ . Then, the dependencies of the code

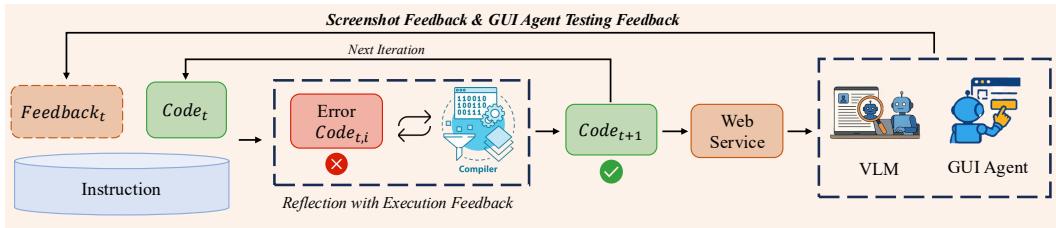


Figure 1: Iterative website generation with screenshot- and GUI-agent-based reflection. A backtracking and best-step-selection mechanism is applied on the basis of the screenshot and GUI-agent testing scores.

base are installed, and the website service is started. The code execution output is denoted as  $\mathcal{O}_1$ , which contains both stdout and stderr. If the dependency installation or service initialization fails, the output message  $\mathcal{O}_1$  is returned to the agent as feedback, so that the agent can fix the error in the next step. If no error occurs, a screenshot of the website is captured and presented to a VLM, which is requested to provide a description of the screenshot and, if needed, suggestions to improve the website’s appearance. The prompt for acquiring screenshot feedback is provided in Fig. 4 of Appendix B. A score of the website appearance based on the screenshot is also generated and, together with the description and suggestions, composes the screenshot feedback. The feedback can be denoted as:

$$\mathcal{F}_{\text{shot}} = \langle \text{Description}, \text{Score}_{\text{shot}}, \text{Suggestions}_{\text{shot}} \rangle \quad (1)$$

$\mathcal{F}_{\text{shot}}$  is used to reflects the integrity and aesthetics of the website’s appearance. Here, a separate VLM is used besides the agent-engine LLM to make the system more cost-effective, as we observe that a relatively small open-source VLM is sufficient for the task, while the code generation requires an LLM with strong code-generation abilities. We use Qwen2.5-VL-32B-Instruct as the VLM in our experiments unless stated otherwise. The code execution and screenshot feedback are appended to the agent trajectory, resulting in  $\mathcal{T} = [\mathcal{I}, \Delta\mathcal{C}_1, \mathcal{O}_1, \mathcal{F}_{\text{shot},1}]$ . Then, the agent judges whether the website’s appearance is satisfactory based on the trajectory. If it is unsatisfactory, the agent continues to generate code  $\Delta\mathcal{C}_2$  to improve the website’s appearance. Otherwise, the agent initiates a GUI-agent testing session, generating an instruction for the GUI-agent to explore various website functionalities specified in the instruction  $\mathcal{I}$ , resulting in a GUI-agent testing trajectory. The prompt used to generate the GUI-agent instructions is shown in Fig.6 of Appendix B. It instructs the model to produce a GUI-agent instruction that comprehensively checks all website-development requirements and includes a one-shot example. As shown in Table7 of Appendix E, a manual inspection indicates that 98.3% of the sampled instructions achieve high coverage of the requirements. Based on the GUI-agent testing result, the agent-engine LLM judges whether the testing is successful and provides a score, denoted as  $\text{Score}_{\text{gui}}$ . The prompt for acquiring the GUI-agent testing feedback is provided in Fig. 7 of Appendix B. If the testing result is unsatisfactory, suggestions are also made to improve the functionality. Thus, the GUI-agent testing feedback can be denoted as

$$\mathcal{F}_{\text{gui}} = \langle \text{Score}_{\text{gui}}, \text{Suggestions}_{\text{gui}} \rangle \quad (2)$$

$\mathcal{F}_{\text{gui}}$  is also appended to the trajectory, resulting in  $\mathcal{T} = [\mathcal{I}, \Delta\mathcal{C}_1, \mathcal{O}_1, \mathcal{F}_{\text{shot},1}, \mathcal{F}_{\text{gui},1}] = [\mathcal{I}, \Delta\mathcal{C}_1, \mathcal{O}_1, \mathcal{F}_1]$ . Here,  $\mathcal{F}_1$  denotes  $[\mathcal{F}_{\text{shot},1}, \mathcal{F}_{\text{gui},1}]$ . In this way, WebGen-Agent continues to improve the appearance and functionality of the website, resulting in a trajectory  $\mathcal{T}$ , denoted as:

$$\mathcal{T} = [\mathcal{I}, \Delta\mathcal{C}_1, \mathcal{O}_1, \mathcal{F}_1, \Delta\mathcal{C}_2, \mathcal{O}_2, \mathcal{F}_2, \dots, \Delta\mathcal{C}_K, \mathcal{O}_K, \mathcal{F}_K] \quad (3)$$

The process ends when the website passes the GUI-agent testing, or the maximum iteration number is reached. During the iteration, at step  $i \in \{1, 2, \dots\}$ , the code base state  $\mathcal{C}_i$ , the edit  $\Delta\mathcal{C}_i$ , together with the  $\text{Score}_{\text{shot},i}$  and  $\text{Score}_{\text{gui},i}$ , are stored in a list. If five consecutive steps contain code execution errors, a backtracking mechanism is triggered, and the agent trajectory and the code base are returned to the state at the best previous step. The best previous step is selected by first choosing the steps with the highest  $\text{Score}_{\text{gui}}$ , and then among these steps, the ones with the highest  $\text{Score}_{\text{shot}}$  are chosen. If there are still more than one chosen step, then the latest one among them is selected. Considering that later code edits might not always improve the previous code base, at the end of the agent workflow,

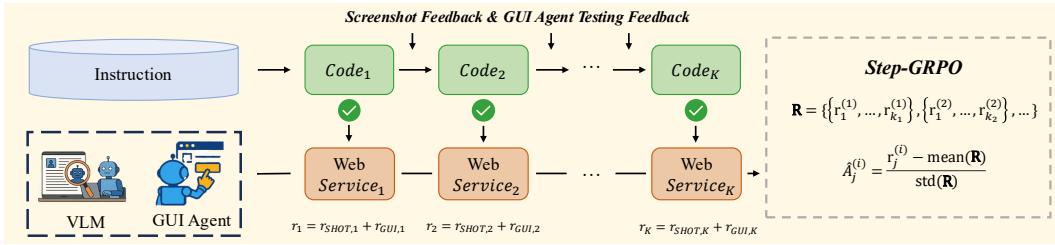


Figure 2: Step-GRPO with Screenshot and GUI-agent Feedback. Multiple WebGen-Agent trajectories are produced, and the reward for each step is computed by summing the screenshot score and the GUI-agent score.

the best step among all the steps is selected in the same way as mentioned above. A more detailed algorithmic presentation can be found in AppendixA.

## 2.2 STEP-GRPO WITH SCREENSHOT AND GUI-AGENT FEEDBACK

While using strong proprietary models as the agent-engine LLM in WebGen-Agent can produce high performance, the agent workflow would be more cost-efficient if smaller open-source models of 7B-8B parameters can be used instead. However, current small open-source language models still lag behind proprietary models in website code generation. Therefore, we introduce Step-GRPO with Screenshot and GUI-agent Feedback, leveraging the  $Score_{shot}$  and  $Score_{gui}$  inherently produced in the WebGen-Agent workflow to train them with step-level process supervision in GRPO training.

Before the GRPO-based training, we first perform a light supervised fine-tuning (SFT) using  $\sim 700$  WebGen-Agent trajectories generated by DeepSeek-V3, training for one epoch to serve as a warm start. Then, Step-GRPO is performed on the fine-tuned model. The Step-GRPO training objective is as follows:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)]} \left[ \frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \min \left[ \frac{\pi_{\theta}(o_{i,t}|q, o_{i,< t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,< t})} \hat{A}_{i,t}, \text{clip} \left( \frac{\pi_{\theta}(o_{i,t}|q, o_{i,< t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,< t})}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{i,t} \right] \right\} \right], \quad (4)$$

Here,  $q$  denotes the website generation instruction, and  $o_i \sim 1^G$  denotes the group of trajectories generated from the instruction  $q$ . We remove the KL loss to encourage the model to more freely adapt its behavior to the reward signals (Qian et al., 2025).  $o_i$  can be denoted as  $[\Delta\mathcal{C}_1, \mathcal{O}_1, \mathcal{F}_1, \dots, \Delta\mathcal{C}_{K_i}, \mathcal{O}_{K_i}, \mathcal{F}_{K_i}]$ .  $\hat{A}_j^{(i)}$  denotes the advantage of  $o^{(i)}$  at the  $j$ -th step. Different from the naive GRPO, which sets the advantages on all tokens in a trajectory to the same value, the Step-GRPO sets advantages on tokens in different steps to different values. In our work, the GRPO loss is only applied to the model outputs  $\Delta\mathcal{C}_1, \Delta\mathcal{C}_2, \dots, \Delta\mathcal{C}_K$ . We denote the reward of all tokens in the  $j$ -th step of  $o^{(i)}$  as  $r_j^{(i)}$ , which is computed by summing the  $Score_{shot}$  and  $Score_{gui}$  of that step, generated in the WebGen-Agent workflow:

$$r_j^{(i)} = Score_{shot,j}^{(i)} + Score_{gui,j}^{(i)} \quad (5)$$

The rewards for all steps in the trajectories sampled from  $q$  can be written as  $\mathbf{R} = \{\{r_1^{(1)}, \dots, r_{K_1}^{(1)}\}, \dots, \{r_1^{(G)}, \dots, r_{K_G}^{(G)}\}\}$ . The advantage for step  $j$  of the  $i$ -th trajectory is computed by standardizing its immediate reward:  $\hat{A}_j^{(i)} = \frac{r_j^{(i)} - \text{mean}(\mathbf{R})}{\text{std}(\mathbf{R})}$ . We do not accumulate normalized rewards from future steps as in Shao et al. (2024), because in the website-generation task  $Score_{shot}$  and  $Score_{gui}$  directly reflect the quality of the website at the current step, which is more appropriate for representing the desirability of the current code. The Step-GRPO training process is illustrated in Fig. 2. This Step-GRPO method, with screenshot and GUI-agent feedback, incorporates accurate step-level supervision and effectively helps the model learn to generate websites with an appealing appearance and smooth functionality.

216 **3 EXPERIMENTS**

217

218 In this section, we first present the performance of WebGen-Agent on WebGen-Bench using a variety  
 219 of proprietary and open-source LLMs, as well as models trained using Step-GRPO with Screenshot  
 220 and GUI-agent Feedback. Then, we conduct comprehensive ablation studies on the design choices  
 221 in the WebGen-Agent workflow and the Step-GRPO training process.

222

223 **3.1 MAIN RESULTS**

224

225 **Benchmark Dataset and Baselines.** We evaluate WebGen-Agent using WebGen-Bench (Lu et al.,  
 226 2025b), a benchmark containing 101 website-generation instructions in natural language and 647  
 227 GUI-agent test cases, covering a wide range of web applications. We compare WebGen-Agent  
 228 with three other popular code agents: OpenHands (Wang et al., 2024), Aider (Aider-AI, 2024),  
 229 and Bolt.diy (stackblitz labs, 2024). We present the results of OpenHands and Aider in combination  
 230 with DeepSeek-V3 (Liu et al., 2024), Claude-3.5-Sonnet (Anthropic, 2024), and DeepSeek-R1 (Guo  
 231 et al., 2025a), as well as the results of Bolt.diy with DeepSeek-V3 (Liu et al., 2024), Claude-  
 232 3.5-Sonnet (Anthropic, 2024), DeepSeek-R1 (Guo et al., 2025a), GPT-4o (Hurst et al., 2024), o3-  
 233 mini (OpenAI, 2025b), Qwen2.5-Coder-32B (Hui et al., 2024), Qwen2.5-72B-Instruct (Yang et al.,  
 234 2024a), WebGen-LM-7B, WebGen-LM-14B, and WebGen-LM-32B (Lu et al., 2025b). The values  
 235 are taken from (Lu et al., 2025b).

236

237 **Models and WebGen-Agent Inference Settings.** We evaluate WebGen-Agent using a wide range  
 238 of proprietary and open-source models as the engine LLMs. The proprietary models we tested in-  
 239 clude Claude-3.5-Sonnet (Anthropic, 2024), DeepSeek-R1 (Guo et al., 2025a), DeepSeek-V3 (Liu  
 240 et al., 2024), o3 (OpenAI, 2025a), Claude-4-Sonnet (Anthropic, 2025), Gemini-2.5-Pro (Comanici  
 241 et al., 2025), and Qwen3-Coder-480B-A35B-Instruct (Yang et al., 2025a). The smaller open-source  
 242 models we tested include Qwen2.5-Coder-32B-Instruct (Hui et al., 2024), Qwen3-Coder-30B-A3B-  
 243 Instruct (Yang et al., 2025a), Qwen2.5-72B-Instruct (Yang et al., 2024a), Qwen2.5-Coder-7B-  
 244 Instruct (Hui et al., 2024), and Qwen3-8B (Yang et al., 2025a), as well as 7B and 8B WebGenAgent-  
 245 LM models trained with supervised fine-tuning (SFT) and Step-GRPO. The maximum number of  
 246 iterations for WebGen-Agent is set to 20, and the model temperature is set to 0.5. Analysis of the  
 247 maximum iteration number is presented in Appendix G.

248

249 **Training Settings.** We first train Qwen2.5-Coder-7B-Instruct and Qwen3-8B on seven hundred  
 250 WebGen-Agent trajectories collected from DeepSeek-V3 for one epoch with a learning rate of  
 251  $4e-5$  and a batch size of 32, resulting in WebGenAgent-LM-7B-SFT and WebGenAgent-LM-8B-  
 252 SFT. This serves as a warm start for the Step-GRPO training. We then train the SFT models  
 253 on five hundred website generation instructions randomly sampled from WebGen-Instruct for one  
 254 epoch, with each instruction sampled five times, resulting in WebGenAgent-LM-7B-Step-GRPO  
 255 and WebGenAgent-LM-8B-Step-GRPO. The training rate is set to  $1e-6$  and the batch size is 16.  
 256 We removed the login requirements in the instructions to make the GUI-agent testing more accurate  
 257 and manually filtered out ambiguous or underspecified instructions. We observed that this relatively  
 258 small number of high-quality instructions is sufficient for the Step-GRPO training, likely due to the  
 259 reliable step-level feedback from screenshots and the GUI-agent. Training on more samples is costly  
 260 and does not yield noticeable gains.

261

262 **Results.** The WebGen-Agent test results are presented in Tab. 1. Based on the results, we make  
 263 the following observations: (1) WebGen-Agent demonstrates superior performance across various  
 264 proprietary models compared to other code agent systems. On Claude-3.5-Sonnet, DeepSeek-R1,  
 265 and DeepSeek-V3, WebGen-Agent significantly outperforms OpenHands, Aider, and Bolt.diy when  
 266 using the same model. Across all seven proprietary models from five different providers, WebGen-  
 267 Agent achieves consistently high performance, demonstrating the generalizability of the method.  
 268 Notably, Qwen3-Coder-480B-A35B-Instruct achieves the highest accuracy of 58.2% and an appear-  
 269 ance score of 4.3. (2) With 30B-72B sized open-source models, WebGen-Agent also achieves high  
 270 performance. On Qwen2.5-Coder-32B-Instruct and Qwen2.5-72B-Instruct, WebGen-Agent outper-  
 271 forms the previous state-of-the-art, Bolt.diy, by 22.5% and 22.1% in accuracy, and by 2.2 and 2.0  
 272 in appearance scores, respectively. Qwen3-Coder-30B-A3B-Instruct achieves the best performance  
 273 among 30B-72B models, with 52.8% accuracy and an appearance score of 4.0. (3) Step-GRPO with

270 Table 1: The performance of WebGen-Agent with various proprietary and open-source models on  
 271 WebGen-Bench (Lu et al., 2025b), compared with other code agent systems. The highest **Accuracy**  
 272 and **Appearance Score** are highlighted in bold.

274 <b>Test Name</b>	275 <b>Yes</b>	275 <b>Partial</b>	275 <b>No</b>	275 <b>Start Failed</b>	275 <b>Accuracy</b>	275 <b>Appearance Score</b>
<b>OpenHands</b>						
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
<b>Aider</b>						
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	1.3
<b>Bolt.diy</b>						
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
<b>WebGen-Agent</b>						
<b>Proprietary Models</b>						
Claude-3.5-Sonnet	45.6	12.7	40.6	1.1	51.9	3.9
DeepSeek-R1	40.2	12.4	45.9	1.5	46.4	3.8
DeepSeek-V3	46.1	13.1	40.6	0.2	52.6	3.8
o3	45.7	11.9	41.6	0.8	51.7	3.5
Gemini-2.5-Pro	44.5	12.7	39.4	3.4	50.9	3.8
Claude-4-Sonnet	48.8	15.3	33.4	2.5	56.5	4.1
Qwen3-Coder-480B-A35B-Inst.	50.5	15.3	34.2	0.0	<b>58.2</b>	<b>4.3</b>
<b>Open-Source Models (30B–72B)</b>						
Qwen2.5-Coder-32B-Inst.	26.7	10.5	60.3	2.5	32.0	3.3
Qwen3-Coder-30B-A3B-Inst.	45.7	14.1	40.2	0.0	<b>52.8</b>	<b>4.0</b>
Qwen2.5-72B-Instruct	29.1	13.8	57.2	0.0	35.9	3.4
<b>Open-Source Models (7B–8B)</b>						
Qwen2.5-Coder-7B-Inst.	10.0	4.8	60.9	24.3	12.4	1.6
WebGenAgent-LM-7B-SFT	33.8	10.2	56.0	0.0	38.9	3.4
WebGenAgent-LM-7B-Step-GRPO	40.2	10.5	49.3	0.0	<b>45.4</b>	<b>3.7</b>
Qwen3-8B	29.5	9.1	61.4	0.0	34.1	3.2
WebGenAgent-LM-8B-SFT	32.8	11.6	55.6	0.0	38.6	3.4
WebGenAgent-LM-8B-Step-GRPO	37.4	12.1	50.5	0.0	<b>43.4</b>	<b>3.6</b>

317 Screenshot and GUI-agent Feedback significantly improves the performance of Qwen2.5-Coder-  
 318 7B-Instruct and Qwen3-8B. For Qwen2.5-Coder-7B-Instruct, SFT improves accuracy from 12.4%  
 319 to 38.9% and the appearance score from 1.6 to 3.4; Step-GRPO further improves accuracy from  
 320 38.9% to 45.4% and the appearance score from 3.4 to 3.7. For Qwen3-8B, SFT improves accuracy  
 321 from 34.1% to 38.6% and the appearance score from 3.2 to 3.4; Step-GRPO further improves accu-  
 322 racy from 38.6% to 43.4% and the appearance score from 3.4 to 3.6. Qualitative analysis of SFT  
 323 and Step-GRPO’s effect in improving the performance is presented in Appendix H. These results  
 demonstrate the effectiveness of Step-GRPO with Screenshot and GUI-agent Feedback in improving

324 Table 2: Ablation study on the WebGen-Agent workflow. The configuration starts from execution-  
 325 only and incrementally adds capabilities.  
 326

327 <b>Test Name</b>	328 <b>Yes</b>	328 <b>Partial</b>	328 <b>No</b>	328 <b>Start Failed</b>	328 <b>Accuracy</b>	328 <b>Appearance Score</b>
329 Execution-only	329 39.7	329 12.4	329 43.3	329 4.6	329 45.9	329 3.0
330 Screenshot	330 41.3	330 10.7	330 45.9	330 2.2	330 46.6	330 3.6
331 Screenshot+GUI-agent	331 43.0	331 13.9	331 41.3	331 1.9	331 49.9	331 3.4
332 Screenshot+GUI-agent+Backtrack	332 45.6	332 11.1	332 43.1	332 0.2	332 51.2	332 3.7
333 <b>Screenshot+GUI-agent+Backtrack+Select-best</b>	333 46.1	333 13.1	333 40.6	333 0.2	333 <b>52.6</b>	333 <b>3.8</b>

334 Table 3: Training-strategy ablation for the Qwen2.5-Coder-7B-Instruct model. The configuration  
 335 starts from the raw model and successively introduces supervised fine-tuning (SFT) and various  
 336 reinforcement-learning variants.  
 337

339 <b>Test Name</b>	340 <b>Yes</b>	340 <b>Partial</b>	340 <b>No</b>	340 <b>Start Failed</b>	340 <b>Accuracy</b>	340 <b>Appearance Score</b>
341 No Additional Training	341 10.0	341 4.8	341 60.9	341 24.3	341 12.4	341 1.6
342 SFT for 1 Epoch	342 33.8	342 10.2	342 56.0	342 0.0	342 38.9	342 3.4
343 SFT for 2 Epochs	343 32.1	343 14.2	343 53.5	343 0.2	343 39.3	343 3.4
344 Naive Outcome GRPO	344 38.0	344 9.0	344 53.0	344 0.0	344 42.5	344 3.5
345 Step-GRPO w/ Cumulative Advantage	345 32.6	345 12.2	345 55.2	345 0.0	345 38.7	345 3.5
346 Step-GRPO w/ Screenshot Reward Only	346 34.9	346 10.5	346 53.9	346 0.6	346 40.2	346 3.5
347 Step-GRPO w/ GUI-agent Reward Only	347 34.8	347 11.3	347 53.6	347 0.3	347 40.4	347 3.4
348 <b>Step-GRPO w/ Screenshot+GUI-agent (ours)</b>	348 40.2	348 10.5	348 49.3	348 0.0	348 <b>45.4</b>	348 <b>3.7</b>

351 both the functionality and appearance of the generated websites. Categorical results are presented  
 352 in Tab. 7 of Appendix F.  
 353

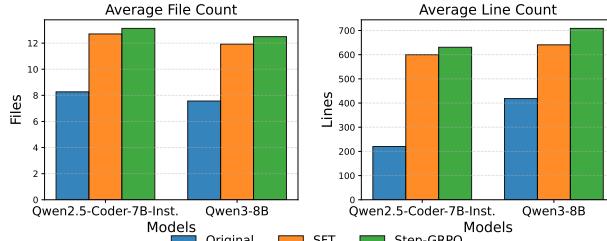
### 354 3.2 ABLATION STUDIES

356 **Analysis of the WebGen-Agent Workflow.** We analyze various design choices in the WebGen-  
 357 Agent workflow in Tab.2. We incrementally add the designs, starting from using only the  
 358 code execution response messages  $\mathcal{O}$  (“Execution-only”), then gradually add screenshot feed-  
 359 back  $\mathcal{F}$ shot (“Screenshot”), GUI-agent testing feedback  $\mathcal{F}$ gui (“Screenshot+GUI-agent”), the back-  
 360 tracking mechanism (“Screenshot+GUI-agent+Backtrack”), and finally the select-best mechanism  
 361 (“Screenshot+GUI-agent+Backtrack+Select-best”), which makes up the full WebGen-Agent work-  
 362 flow. As shown in Tab.2, each of the designs yields notable gains in accuracy and appearance. The  
 363 GUI-agent testing contributes the largest accuracy gain of 3.3%, showing its effectiveness in guiding  
 364 the functionality of the generated websites. The addition of screenshot feedback greatly improves  
 365 the appearance score, raising it from 3.0 to 3.6, demonstrating its effect in enhancing website appear-  
 366 ance. Adding GUI-agent testing slightly impairs the appearance score, likely because modifying the  
 367 code base for functional fulfillment sometimes damages the website appearance or causes errors.  
 368 This negative effect is mitigated by the addition of the backtracking and select-best mechanisms.  
 369 Qualitative analysis of the effect of screenshot and GUI-agent feedback is provided in Appendix I.  
 370

371 **Analysis of Step-GRPO with Screenshot and GUI-agent Feedback.** We analyze the design  
 372 choices in the Step-GRPO with Screenshot and GUI-agent Feedback training process in Tab.3. The  
 373 first line shows the result of Qwen2.5-Coder-7B-Instruct with no additional training. The analysis  
 374 based on Tab.3 is as follows: (1) The second and third lines present SFT training for one epoch and  
 375 two epochs, showing that training with SFT for two epochs does not notably improve performance  
 376 compared to training for only one epoch. Therefore, we trained for only one epoch in the SFT  
 377 stage. (2) The fourth and fifth lines show the results of using naive outcome GRPO and Step-GRPO  
 378 with cumulative advantage. The rewards in these two variants are the same as in our final design  
 379 ( $Score_{shot} + Score_{gui}$ ); only the advantage computation method differs. Naive outcome GRPO uses

378  
 379 Table 4: Impact of the feedback VLM on WebGen-Agent performance. For every agent–VLM pair  
 380 we report WebGen-Bench results; the highest **Accuracy** and **Appearance Score** are highlighted in  
 381 bold.

382 <b>Agent-engine LLM</b>	383 <b>Feedback VLM</b>	384 <b>Yes</b>	385 <b>Partial</b>	386 <b>No</b>	387 <b>Start Failed</b>	388 <b>Accuracy</b>	389 <b>Appearance</b>	390 <b>Score</b>
384 Qwen2.5-VL-32B-Inst.	385 Qwen2.5-VL-32B-Inst.	386 4.5	387 2.2	388 78.8	389 14.5	390 5.6	391 1.3	392
384 DeepSeek-V3	385 GPT-4o	386 46.4	387 11.4	388 42.0	389 0.2	390 52.1	391 3.6	392
384 <b>DeepSeek-V3</b>	385 <b>Qwen2.5-VL-32B-Inst.</b>	386 <b>46.1</b>	387 <b>13.1</b>	388 <b>40.6</b>	389 <b>0.2</b>	390 <b>52.6</b>	391 <b>3.8</b>	392



393 Figure 3: Comparison of the average file count and average line count among the original, SFT, and  
 394 Step-GRPO models for Qwen2.5-Coder-7B-Instruct and Qwen3-8B.

400 the maximum value of the step-level rewards in a trajectory as the outcome reward, setting the  
 401 advantages to the normalized outcome rewards. Step-GRPO with cumulative advantage calculates  
 402 the advantage of each token as the sum of the normalized rewards from the subsequent steps, as  
 403 introduced in Shao et al. (2024). Both GRPO advantage computation variants perform notably  
 404 worse than our Step-GRPO setting. The sixth and seventh lines present the results of using only  
 405 the screenshot scores ( $Score_{shot}$ ) or only the GUI-agent testing scores ( $Score_{gui}$ ) as the rewards.  
 406 Both are lower than using  $Score_{shot} + Score_{gui}$ , demonstrating the necessity of incorporating both  
 407 screenshot and GUI-agent testing feedback. We also gather statistics on the average file count and  
 408 average line count for the Original, SFT, and Step-GRPO models. The results are shown in Fig. 3.  
 409 For both Qwen2.5-Coder-7B-Instruct and Qwen3-8B, the average file count and average line count  
 410 consistently increase with SFT and GRPO. This shows that the SFT stage and the Step-GRPO stage  
 411 both increase the complexity of the generated websites, which is consistent with their improved  
 412 performance.

413 **Analysis of the Agent-Engine LLM and Feedback VLM.** We analyze the choice of the agent-  
 414 engine LLM and feedback VLM in Tab.4. In our experiments, we use a relatively small and inex-  
 415 pensive VLM, Qwen2.5-VL-32B-Instruct, to provide screenshot and GUI-agent testing feedback, while  
 416 employing a strong LLM capable of generating high-quality code, such as DeepSeek-V3, as the  
 417 agent-engine LLM. As shown in the second row of Tab.4, replacing Qwen2.5-VL-32B-Instruct with  
 418 a proprietary VLM, GPT-4o, as the feedback VLM does not notably improve accuracy or appearance  
 419 scores. This demonstrates that Qwen2.5-VL-32B-Instruct is already sufficient for providing accu-  
 420 rate screenshot and GUI-agent testing feedback, while being more cost-effective than proprietary  
 421 VLMs. As shown in the first row of Tab. 4, replacing DeepSeek-V3 with Qwen2.5-VL-32B-Instruct  
 422 results in significantly worse performance, indicating that the agent-engine LLM cannot be replaced  
 423 by smaller open-source VLMs. The design choice of decoupling the agent-engine LLM and feed-  
 424 back VLM ensures that code is generated by a strong LLM to maintain quality, while screenshot and  
 425 GUI-agent testing feedback is handled by a smaller open-source VLM for cost efficiency. Further  
 426 analysis of the accuracy of the screenshot and GUI-agent scores provided by the feedback VLM is  
 427 included in Tab. 5 of Appendix D, demonstrating the reliability of the scores.

## 4 RELATED WORK

431 **Visual Code Generation.** Code generation that is associated with visual effects exists in a wide  
 432 range of application scenarios, such as web page development (Lu et al., 2025b; Xu et al., 2025)

432 and GitHub-issue fixing (Yang et al., 2024d; Guo et al., 2025b). Previous work has proposed vari-  
 433 ous ways to treat visual elements in code generation and other reasoning-intensive tasks (Su et al.,  
 434 2025), such as generating code to represent images in problem statements (Huang et al., 2025; Wang  
 435 et al., 2025b) and using natural language to describe images(Zhang et al., 2024b). We also apply  
 436 natural language descriptions when providing screenshot feedback. More related to our work, a line  
 437 of studies(Guo et al., 2024; Si et al., 2025; Yun et al., 2024; Beltramelli, 2017; Sun et al., 2025;  
 438 Gui et al., 2025; Laurençon et al., 2024; Wan et al., 2024) explores MLLMs’ ability to reconstruct  
 439 single-file HTML code from webpage screenshots. Other studies benchmark MLLMs’ performance  
 440 in implementing interactive elements in existing web projects (Xiao et al., 2025a) or performing  
 441 web development tasks in a pre-defined sequential manner with detailed technical settings (Xiao  
 442 et al., 2025b; Xu et al., 2025). The web development tasks in these works are often solved in a  
 443 single HTML file (Zhang et al., 2025a) or contain rigid pipelines (Xu et al., 2025), which are more  
 444 suitable for testing MLLMs rather than code agents for end-to-end, repository-level website devel-  
 445 opment, as proposed in our work. We evaluate our agent workflow with WebGen-Bench (Lu et al.,  
 446 2025b), which measures a code agent’s ability to create multi-file website code bases from scratch  
 447 and includes diverse website generation instructions.  
 448

449 **Code Agents.** Equipped with various tools and powered LLMs(Soni et al., 2025; Yao et al., 2023;  
 450 Zhang et al., 2024a), code agents can perform a variety of tasks, such as developing websites(Lu  
 451 et al., 2025b) and fixing GitHub issues (Jimenez et al., 2024; Yang et al., 2024c). Some code agents  
 452 specialize in a specific field, such as bug fixing (Zhang et al., 2024c) or machine learning (Jiang  
 453 et al., 2025). Similar to our work, Bolt.diy (stackblitz labs, 2024) specializes in multi-file website  
 454 generation. Others, such as OpenHands (Wang et al., 2024) and Aider (Aider-AI, 2024), are general-  
 455 purpose code agents that are not limited to a single field, though their performance on a specific  
 456 task might not match that of specialist code agents (Lu et al., 2025b). Our WebGen-Agent is a  
 457 code agent specializing in end-to-end website generation, with screenshot feedback and GUI-agent  
 458 testing features specifically designed for this task, achieving state-of-the-art performance.  
 459

460 **Fine-tuning and Reinforcement Learning for Agents.** Supervised fine-tuning (Pan et al., 2025;  
 461 Yang et al., 2025b) and reinforcement learning (Dong et al., 2025; Qian et al., 2025) are two methods  
 462 widely used to improve the agentic and tool-calling abilities of LLMs. In the field of code agents,  
 463 various works (Pan et al., 2025; Yang et al., 2025b; Zhang et al., 2025b; Wang et al., 2025a; Ma et al.,  
 464 2024; Xie et al., 2025; Jain et al., 2025; Guo et al., 2025c; Ma et al., 2025a) leverage supervised  
 465 fine-tuning combined with software engineering data synthesis and rejection sampling to improve  
 466 the performance of open-source models. Similar to these works, we also use rejection sampling  
 467 and supervised fine-tuning in the warm-up stage before Step-GRPO. Other works use reinforcement  
 468 learning with rewards acquired through comparison with the ground truth (Wei et al., 2025a; Ma  
 469 et al., 2025c; Zhuang et al., 2025), determined by the code execution output (Gehring et al., 2025;  
 470 Ma et al., 2025b; Golubev et al., 2025), or dependent on task success (Wei et al., 2025b; Lu et al.,  
 471 2025a; Chen et al., 2025). These works either use outcome supervision, which is sparse in providing  
 472 training signals, or require detailed ground truth to provide step supervision, which is rigid and  
 473 difficult to obtain. In contrast to these methods, our work leverages screenshot and GUI-agent  
 474 testing scores at each step, which are inherent in the WebGen-Agent pipeline, to provide accurate  
 475 step-level supervision in Step-GRPO training.  
 476

## 477 5 LIMITATIONS AND FUTURE WORK

478  
 479 WebGen-Agent is specifically designed to generate websites based on natural language instructions  
 480 from non-expert users. We do not consider response speed or complex network conditions when  
 481 generating and evaluating the websites; these are interesting questions for future work. In the su-  
 482 pervised fine-tuning and Step-GRPO experiments, we trained only 7B- and 8B-parameter models  
 483 due to limited computing power and GPU memory, as Step-GRPO training would take more than  
 484 24 hours on 16 NVIDIA A800 GPUs, and we currently do not have enough GPUs to train larger  
 485 models. The results on the 7B and 8B models show great potential for our method, and we plan to  
 486 apply our training approach to 30B–72B models in the future.  
 487

486 

## 6 CONCLUSION

488 In this paper, we introduce WebGen-Agent, a code agent that leverages screenshot and GUI-agent  
 489 testing feedback, combined with backtracking and select-best mechanisms, to iteratively generate  
 490 websites with appealing appearance and smooth functionality. We also propose Step-GRPO with  
 491 Screenshot and GUI-agent Feedback, which leverages inherent screenshot and GUI-agent testing  
 492 scores to provide step-level supervision in the GRPO training process. Testing WebGen-Agent  
 493 on WebGen-Bench shows significant improvements across a wide range of proprietary and open-  
 494 source LLMs compared with other code agent systems. WebGen-Agent with Qwen3-Coder-480B-  
 495 A35B-Instruct achieves the best performance, with 58.2% accuracy and a 4.3 appearance score.  
 496 Training Qwen2.5-Coder-7B-Instruct and Qwen3-8B first with supervised fine-tuning and then with  
 497 Step-GRPO with Screenshot and GUI-agent Feedback notably improves accuracies and appearance  
 498 scores, demonstrating the effectiveness of our training approach.

499 

## 500 REFERENCES

- 501 Aider-AI. Ai pair programming in your terminal, 2024. URL <https://github.com/Aider-AI/aider>. Accessed: 2025-04-22.
- 504 Anthropic. Introducing claude 3.5 sonnet, 2024. URL <https://www.anthropic.com/news/clause-3-5-sonnet>. Accessed: 2025-04-22.
- 507 Anthropic. Claude sonnet 4, 2025. URL <https://www.anthropic.com/clause/sonnet>. Accessed: 2025-08-11.
- 509 Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot, 2017. URL  
 510 <https://arxiv.org/abs/1705.07962>.
- 512 Yongchao Chen, Yueying Liu, Junwei Zhou, Yilun Hao, Jingquan Wang, Yang Zhang, and Chuchu  
 513 Fan. R1-code-interpreter: Training llms to reason with code via supervised and reinforcement  
 514 learning, 2025. URL <https://arxiv.org/abs/2505.21668>.
- 515 Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit  
 516 Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the  
 517 frontier with advanced reasoning, multimodality, long context, and next generation agentic capa-  
 518 bilities. *arXiv preprint arXiv:2507.06261*, 2025.
- 520 Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia  
 521 Chen, Jiazen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and  
 522 Zhicheng Dou. Agentic reinforced policy optimization, 2025. URL <https://arxiv.org/abs/2507.19849>.
- 524 Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and  
 525 Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning,  
 526 2025. URL <https://arxiv.org/abs/2410.02089>.
- 528 Alexander Golubev, Maria Trofimova, Sergei Polezhaev, Ibragim Badertdinov, Maksim Nekrashe-  
 529 vich, Anton Shevtsov, Simon Karasik, Sergey Abramov, Andrei Andriushchenko, Filipp Fisin,  
 530 Sergei Skvortsov, and Boris Yangel. Training long-context, multi-turn software engineering  
 531 agents with reinforcement learning, 2025. URL <https://arxiv.org/abs/2508.03501>.
- 532 Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan  
 533 Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. Webcode2m: A real-world dataset  
 534 for code generation from webpage designs. In *Proceedings of the ACM on Web Conference*  
 535 2025, WWW '25, pp. 1834–1845. ACM, April 2025. doi: 10.1145/3696410.3714889. URL  
 536 <http://dx.doi.org/10.1145/3696410.3714889>.
- 538 Daya Guo, Dejian Yang, Huawei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,  
 539 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms  
 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

- 540 Hongcheng Guo, Wei Zhang, Junhao Chen, Yaonan Gu, Jian Yang, Junjia Du, Binyuan Hui, Tianyu  
 541 Liu, Jianxin Ma, Chang Zhou, and Zhoujun Li. Iw-bench: Evaluating large multimodal models  
 542 for converting image-to-web, 2024. URL <https://arxiv.org/abs/2409.18980>.
- 543 Lianghong Guo, Wei Tao, Runhan Jiang, Yanlin Wang, Jiachi Chen, Xilin Liu, Yuchi Ma, Mingzhi  
 544 Mao, Hongyu Zhang, and Zibin Zheng. Omnidgirl: A multilingual and multimodal benchmark for  
 545 github issue resolution, 2025b. URL <https://arxiv.org/abs/2505.04606>.
- 546 Lianghong Guo, Yanlin Wang, Caihua Li, Pengyu Yang, Jiachi Chen, Wei Tao, Yingtian Zou, Duyu  
 547 Tang, and Zibin Zheng. Swe-factory: Your automated factory for issue resolution training data  
 548 and evaluation benchmarks, 2025c. URL <https://arxiv.org/abs/2506.10954>.
- 549 Kai Huang, Jian Zhang, Xiaofei Xie, and Chunyang Chen. Seeing is fixing: Cross-modal reasoning  
 550 with multimodal llms for visual software issue fixing, 2025. URL <https://arxiv.org/abs/2506.16136>.
- 551 Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang,  
 552 Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*,  
 553 2024.
- 554 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-  
 555 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint*  
 556 *arXiv:2410.21276*, 2024.
- 557 Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym:  
 558 Procedural environments and hybrid verifiers for scaling open-weights swe agents, 2025. URL  
 559 <https://arxiv.org/abs/2504.07164>.
- 560 Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixin Xu, Ian Kaplan, Deniss Jacenko, and  
 561 Yuxiang Wu. Aide: Ai-driven exploration in the space of code, 2025. URL <https://arxiv.org/abs/2502.13138>.
- 562 Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik  
 563 Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL  
 564 <https://arxiv.org/abs/2310.06770>.
- 565 Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into  
 566 html code with the websight dataset, 2024. URL <https://arxiv.org/abs/2403.09029>.
- 567 Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao,  
 568 Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint*  
 569 *arXiv:2412.19437*, 2024.
- 570 Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo:end-to-end policy opti-  
 571 mization for gui agents with experience replay, 2025a. URL <https://arxiv.org/abs/2505.16282>.
- 572 Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun  
 573 Zhou, Mingjie Zhan, and Hongsheng Li. Webgen-bench: Evaluating llms on generating interac-  
 574 tive and functional websites from scratch, 2025b. URL <https://arxiv.org/abs/2505.03733>.
- 575 Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua  
 576 Li, Fei Huang, and Yongbin Li. Lingma swe-gpt: An open development-process-centric language  
 577 model for automated software improvement, 2024. URL <https://arxiv.org/abs/2411.00622>.
- 578 Yingwei Ma, Yongbin Li, Yihong Dong, Xue Jiang, Rongyu Cao, Jue Chen, Fei Huang, and Binhua  
 579 Li. Thinking longer, not larger: Enhancing software engineering agents via scaling test-time  
 580 compute, 2025a. URL <https://arxiv.org/abs/2503.23803>.
- 581 Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzhen Zou, and Bing Xie. Sorft: Issue  
 582 resolving with subtask-oriented reinforced fine-tuning, 2025b. URL <https://arxiv.org/abs/2502.20127>.

- 594 Zexiong Ma, Chao Peng, Qunhong Zeng, Pengfei Gao, Yanzhen Zou, and Bing Xie. Tool-integrated  
 595 reinforcement learning for repo deep search, 2025c. URL <https://arxiv.org/abs/2508.03012>.
- 597 Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. Swe-lancer: Can  
 598 frontier llms earn 1 million from real-world freelance software engineering?, 2025. URL <https://arxiv.org/abs/2502.12115>.
- 601 OpenAI. Introducing openai o3 and o4-mini, 2025a. URL <https://openai.com/index/introducing-o3-and-o4-mini/>. Accessed: 2025-08-11.
- 603 OpenAI. Openai o3-mini, 2025b. URL <https://openai.com/index/openai-o3-mini>.  
 604 Accessed: 2025-04-22.
- 606 Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe  
 607 Zhang. Training software engineering agents and verifiers with swe-gym, 2025. URL <https://arxiv.org/abs/2412.21139>.
- 609 Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan  
 610 Tur, and Heng Ji. Toolrl: Reward is all tool learning needs, 2025. URL <https://arxiv.org/abs/2504.13958>.
- 613 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,  
 614 Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathe-  
 615 matical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- 617 Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Difyi Yang. Design2code:  
 618 Benchmarking multimodal code generation for automated front-end engineering, 2025. URL  
 619 <https://arxiv.org/abs/2403.03163>.
- 620 Aditya Bharat Soni, Boxuan Li, Xingyao Wang, Valerie Chen, and Graham Neubig. Coding agents  
 621 with multimodal browsing are generalist problem solvers, 2025. URL <https://arxiv.org/abs/2506.03011>.
- 624 stackblitz labs. bolt.diy, 2024. URL <https://github.com/stackblitz-labs/bolt-diy>. Accessed: 2025-04-22.
- 626 Zhaochen Su, Peng Xia, Hangyu Guo, Zhenhua Liu, Yan Ma, Xiaoye Qu, Jiaqi Liu, Yanshu Li,  
 627 Kaide Zeng, Zhengyuan Yang, Linjie Li, Yu Cheng, Heng Ji, Junxian He, and Yi R. Fung. Think-  
 628 ing with images for multimodal reasoning: Foundations, methods, and future frontiers, 2025.  
 629 URL <https://arxiv.org/abs/2506.23918>.
- 631 Haoyu Sun, Huichen Will Wang, Jiawei Gu, Linjie Li, and Yu Cheng. Fullfront: Benchmarking  
 632 mllms across the full front-end engineering workflow, 2025. URL <https://arxiv.org/abs/2505.17399>.
- 634 Yuxuan Wan, Yi Dong, Jingyu Xiao, Yintong Huo, Wenxuan Wang, and Michael R. Lyu. Mrweb:  
 635 An exploration of generating multi-page resource-aware web code from ui designs, 2024. URL  
 636 <https://arxiv.org/abs/2412.15310>.
- 637 Haoran Wang, Zhenyu Hou, Yao Wei, Jie Tang, and Yuxiao Dong. Swe-dev: Building software  
 638 engineering agents with training and inference scaling, 2025a. URL <https://arxiv.org/abs/2506.07636>.
- 641 Ke Wang, Junting Pan, Linda Wei, Aojun Zhou, Weikang Shi, Zimu Lu, Han Xiao, Yunqiao Yang,  
 642 Houxing Ren, Mingjie Zhan, and Hongsheng Li. Mathcoder-vl: Bridging vision and code for  
 643 enhanced multimodal mathematical reasoning, 2025b. URL <https://arxiv.org/abs/2505.10557>.
- 645 Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan,  
 646 Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software  
 647 developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2024.

- 648 Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried,  
 649 Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via re-  
 650 enforcement learning on open software evolution, 2025a. URL <https://arxiv.org/abs/2502.18449>.
- 652 Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu,  
 653 Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. Webagent-rl: Training web agents via end-  
 654 to-end multi-turn reinforcement learning, 2025b. URL <https://arxiv.org/abs/2505.16421>.
- 656 Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu,  
 657 Yuhang Wang, and Michael R. Lyu. Interaction2code: Benchmarking mllm-based interactive  
 658 webpage code generation from interactive prototyping, 2025a. URL <https://arxiv.org/abs/2411.03292>.
- 660 Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R.  
 661 Lyu. Designbench: A comprehensive benchmark for mllm-based front-end code generation,  
 662 2025b. URL <https://arxiv.org/abs/2506.06251>.
- 664 Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. Swe-fixer:  
 665 Training open-source llms for effective and efficient github issue resolution, 2025. URL <https://arxiv.org/abs/2501.05040>.
- 667 Kai Xu, YiWei Mao, XinYi Guan, and ZiLong Feng. Web-bench: A llm code benchmark based on  
 668 web standards and frameworks, 2025. URL <https://arxiv.org/abs/2505.07473>.
- 670 An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li,  
 671 Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint*  
 672 *arXiv:2412.15115*, 2024a.
- 673 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu,  
 674 Chang Gao, Chengan Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint*  
 675 *arXiv:2505.09388*, 2025a.
- 677 John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik  
 678 Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated  
 679 software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Pa-  
 680 quet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Process-  
 681 ing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024b. URL  
 682 [https://proceedings.neurips.cc/paper\\_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf).
- 683 John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan,  
 684 and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering,  
 685 2024c. URL <https://arxiv.org/abs/2405.15793>.
- 687 John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press,  
 688 Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and  
 689 Ofir Press. Swe-bench multimodal: Do ai systems generalize to visual software domains?, 2024d.  
 690 URL <https://arxiv.org/abs/2410.03859>.
- 691 John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang,  
 692 Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software  
 693 engineering agents, 2025b. URL <https://arxiv.org/abs/2504.21798>.
- 695 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.  
 696 React: Synergizing reasoning and acting in language models, 2023. URL <https://arxiv.org/abs/2210.03629>.
- 698 Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang,  
 699 Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy  
 700 Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. Web2code: A  
 701 large-scale webpage-to-code dataset and evaluation framework for multimodal llms, 2024. URL  
<https://arxiv.org/abs/2406.20098>.

702 Chenchen Zhang, Yuhang Li, Can Xu, Jiaheng Liu, Ao Liu, Shihui Hu, Dengpeng Wu, Guanhua  
 703 Huang, Kejiao Li, Qi Yi, Ruibin Xiong, Haotian Zhu, Yuanxing Zhang, Yuhao Jiang, Yue Zhang,  
 704 Zenan Xu, Bohui Zhai, Guoxiang He, Hebin Li, Jie Zhao, Le Zhang, Lingyun Tan, Pengyu Guo,  
 705 Xianshu Pang, Yang Ruan, Zhifeng Zhang, Zhonghu Wang, Ziyan Xu, Zuopu Yin, Wiggin Zhou,  
 706 Chayse Zhou, and Fengzong Lian. Artifactsbench: Bridging the visual-interactive gap in llm code  
 707 generation evaluation, 2025a. URL <https://arxiv.org/abs/2507.04952>.

708 Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with  
 709 tool-integrated agent systems for real-world repo-level coding challenges, 2024a. URL <https://arxiv.org/abs/2401.07339>.

710 Kechi Zhang, Huangzhao Zhang, Ge Li, Jinliang You, Jia Li, Yunfei Zhao, and Zhi Jin. Sealign:  
 711 Alignment training for software engineering agent, 2025b. URL <https://arxiv.org/abs/2503.18455>.

712 Linhao Zhang, Daoguang Zan, Quanshun Yang, Zhirong Huang, Dong Chen, Bo Shen, Tianyu Liu,  
 713 Yongshun Gong, Pengjie Huang, Xudong Lu, Guangtai Liang, Lizhen Cui, and Qianxiang Wang.  
 714 Codev: Issue resolving with visual data, 2024b. URL <https://arxiv.org/abs/2412.17315>.

715 Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous  
 716 program improvement, 2024c. URL <https://arxiv.org/abs/2404.05427>.

717 Yuchen Zhuang, Di Jin, Jiaao Chen, Wenqi Shi, Hanrui Wang, and Chao Zhang. Workforceagent-  
 718 r1: Incentivizing reasoning capability in llm-based web agents via reinforcement learning, 2025.  
 719 URL <https://arxiv.org/abs/2505.22942>.

## 720 A WEBGEN-AGENT ALGORITHM

721 Algorithm 1 demonstrates the WebGen-Agent inference workflow in detail. Algorithms 2 and 3 are  
 722 two helper functions for Algorithm 1, presented separately for clarity.

## 723 B WEBGEN-AGENT PROMPTS

724 The prompts for acquiring screenshot and GUI-agent testing feedback are presented in Fig. 4, Fig. 5,  
 725 Fig. 6, and Fig. 7.

## 726 C EXAMPLES OF WEBGEN-AGENT TRAJECTORIES

727 To demonstrate the WebGen-Agent workflow in a straightforward way, we present three example  
 728 trajectories in Fig.8, Fig.9, and Fig. 10. As shown in these examples, WebGen-Agent iteratively  
 729 improves the appearance and functionality of the generated website based on screenshot and GUI-  
 730 agent testing feedback.

## 731 D ACCURACY OF SCREENSHOT AND GUI-AGENT TESTING SCORES

732 To analyze the accuracy of the screenshot and GUI-agent testing scores given by the feedback VLM  
 733 in the WebGen-Agent pipeline, we evaluated the results of Claude-4-Sonnet, Qwen3-Coder-30B-  
 734 A3B-Instruct, Qwen3-Coder-480B-A35B-Instruct, and DeepSeek-V3 as agent-engine LLMs, with  
 735 Qwen2.5-VL-32B-Instruct as the feedback VLM, as well as DeepSeek-V3 as the agent-engine LLM  
 736 and GPT-4o as the feedback VLM. We manually verified the accuracy of the screenshot and GUI-  
 737 agent testing scores. Human annotators were provided with the score and the screenshot or GUI-  
 738 agent trajectory at each step and asked to judge whether the score was accurate. If the score was  
 739 inaccurate, they provided the correct score. The results are presented in Table 5.

740 The accuracies of the screenshot scores across all experiments ranged from 93% to 96%, while the  
 741 accuracies of the GUI-agent scores ranged from 89% to 93%. The standard errors of the screenshot

756  
757  
758  
759  
760  
761  
762  
763  
764

**Prompt:**

You are given a single website screenshot as input.

**Task**

1. Examine the screenshot closely for any rendering or runtime errors (e.g., “404 Not Found”, stack traces, missing styles, blank areas).
2. Decide whether the screenshot *shows a rendering or runtime error*.
  - If **yes**, set “`is_error`”: `true`, extract or paraphrase the visible error message(s) into “`error_message`”, and leave “`screenshot_description`” empty.
  - If **no**, set “`is_error`”: `false`, leave “`error_message`” as an empty string (“”), and write a concise but thorough “`screenshot_description`” that covers:
    - Overall layout (e.g., header/sidebar/footer, grid, flex, single-column).
    - Key UI components (navigation bar, buttons, forms, images, cards, tables, modals, etc.).
    - Color scheme and visual style (dominant colors, light/dark theme, gradients, shadows).
    - Visible content and text (headings, labels, sample data).
    - Notable design details (icons, spacing, font style) that help someone understand what the page looks like).
3. Suggest ways to improve the appearance of the website, for example:
  - Separate incorrectly overlapping components.
  - Adjust layout to avoid large blank areas.
  - Adjust text or background color to avoid text color being too similar to the background color.
  - If no improvement is necessary, leave “`suggestions`” as an empty string (“”); otherwise, briefly list the suggestion(s) in “`suggestions`”.
4. Grade the response.

**Output format (valid JSON)**

```
'''json
{
  "is_error": <boolean>,
  "error_message": "<string>",
  "screenshot_description": "<string>",
  "suggestions": "<string>"
}
'''
```

Return **only** this JSON object—no additional commentary, markdown, or code fences.

800  
801 Figure 4: The prompt for generating the description and suggestions based on the website screen-  
shot.  
802  
803  
804  
805  
806  
807  
808  
809

810  
811  
812  
813  
814  
815  
816  
817

**Prompt:**

You are tasked with evaluating the functional design of a webpage. Grade the webpage's appearance on a scale of 0 to 5 (5 being highest), considering the following criteria:

- **Successful Rendering:** Are there any components in the page or is it completely blank? 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:**

- **0 (Blank Page):** The screenshot is completely blank or does not contain any visible content. It may only have a background color or display an error message.
- **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 concise analysis of a few sentences and then assign a grade (0–5) based on your analysis. Highlight strengths, weaknesses, and how well the design adheres to the specifications.

**Your Response Format**

```
```json
{
  "analysis": "<string>",
  "grade": <int>
}
```
```

**Your Response:**

855  
856  
857  
858  
859  
860  
861  
862  
863

Figure 5: Prompt for evaluating the functional design and visual quality of a webpage.

864

**Algorithm 1** WebGen-Agent (edits vs. code-base states made explicit)

865

**Require:** Initial instruction  $\mathcal{I}$ , maximum steps  $T$   
**Ensure:** Final code base  $\mathcal{C}^*$ 

866

```

867 1:  $\mathcal{T} \leftarrow [\mathcal{I}]$                                  $\triangleright$  trajectory: instruction, edit, feedback, ...
868 2:  $Steps \leftarrow \emptyset$                              $\triangleright$  archive of step snapshots
869 3:  $\mathcal{C} \leftarrow \emptyset$                              $\triangleright$  current code base
870 4:  $t \leftarrow 1$ ,  $consecErr \leftarrow 0$ 
871 5: while  $t \leq T$  do
872 6:    $\Delta\mathcal{C}_t \leftarrow \text{GENERATEEDIT}(\mathcal{T})$ 
873 7:    $\mathcal{T} \leftarrow \mathcal{T} \cup \Delta\mathcal{C}_t$ 
874 8:    $\mathcal{C} \leftarrow \text{APPLYEDIT}(\mathcal{C}, \Delta\mathcal{C}_t)$ 
875 9:    $\mathcal{O} \leftarrow \text{EXECUTE}(\mathcal{C})$ 
876 10:  if  $\mathcal{O} = \text{error}$  then
877 11:     $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{O}$ 
878 12:     $consecErr \leftarrow consecErr + 1$ 
879 13:    if  $consecErr = 5$  then
880 14:       $\langle t^*, \mathcal{C}^*, *, * \rangle \leftarrow \text{SELECTBESTSTEP}(Steps)$ 
881 15:       $\mathcal{C} \leftarrow \mathcal{C}^*$                                  $\triangleright$  restore code base
882 16:       $\mathcal{T} \leftarrow \text{TRUNCATE}(\mathcal{T}, t^*)$ 
883 17:       $t \leftarrow t^* + 1$ ,  $consecErr \leftarrow 0$ 
884 18:    else
885 19:       $t \leftarrow t + 1$ 
886 20:    end if
887 21:    continue
888 22:  else
889 23:     $consecErr \leftarrow 0$ 
890 24:  end if
891 25:   $img \leftarrow \text{SCREENSHOT}(\mathcal{C})$ 
892 26:   $\langle desc, sugg_{shot}, score_{shot} \rangle \leftarrow \text{VLM\_JUDGE}(img)$ 
893 27:   $\mathcal{T} \leftarrow \mathcal{T} \cup \langle desc, sugg_{shot} \rangle$ 
894 28:   $goNext \leftarrow \text{AGENTDECISION}(\mathcal{T})$ 
895 29:  if not  $goNext$  then
896 30:     $t \leftarrow t + 1$ ; continue
897 31:  end if
898 32:   $\langle pass, sugg_{gui}, score_{gui} \rangle \leftarrow \text{GUI\_AGENT}(\mathcal{C})$ 
899 33:   $\mathcal{T} \leftarrow \mathcal{T} \cup \langle pass, sugg_{gui} \rangle$ 
900 34:   $Steps \leftarrow \mathcal{T}$ 
901 35:  if  $pass$  then
902 36:    break
903 37:  else
904 38:     $t \leftarrow t + 1$ 
905 39:  end if
906 40: end while
907 41:  $\langle *, \mathcal{C}^*, *, * \rangle \leftarrow \text{SELECTBESTSTEP}(Steps)$ 
908 42: return  $\mathcal{C}^*$ 

```

903

904

**Algorithm 2** SELECTBESTSTEP

905

**Require:**  $Steps = \{(t, \mathcal{C}, score_{shot}, score_{gui})\}$   
1:  $g_{\max} \leftarrow \max_{s \in Steps} score_{gui}$   
2:  $\mathcal{S}_g \leftarrow \{s \mid score_{gui} = g_{\max}\}$   
3: **return**  $\arg \max_{s \in \mathcal{S}_g} score_{shot}$ 

910

911

912

913 scores range from 0.20 to 0.26, while the standard errors of the GUI-agent scores range from 0.31  
914 to 0.44. This demonstrates that the scores are highly accurate, supporting the effectiveness of the  
915 WebGen-Agent pipeline and the Step-GRPO training process. Compared with using Qwen2.5-VL-  
916 32B-Instruct, using GPT-4o as the feedback VLM only marginally improved the screenshot score  
917 accuracy from 94.8% to 95.5% and the GUI-agent score accuracy from 91.2% to 92.2%. This shows  
918 that Qwen2.5-VL-32B-Instruct is sufficient for the task while being significantly more cost-effective.

---

**Algorithm 3** TRUNCATE

**Require:** Trajectory  $\mathcal{T}$ , step id  $t^*$

1: **return** prefix of  $\mathcal{T}$  ending just after the edit and feedback of step  $t^*$

Based on the original website development instruction, you should identify **all the requirements** of the website generation and create a comprehensive instruction for a web-navigation GUI agent to test the generated website. The following is an example of triggering the GUI agent testing based on the original instruction:

### *Example*

Original instruction:

Please implement a self-driving tour website that provides self-driving tour products and services. The website should have functionalities for browsing self-driving tour routes, booking self-driving tour hotels, and self-help self-driving tour packages. Users should be able to browse different types of self-driving tour routes, book hotels and packages, and query self-driving club information. The website should also provide search and filtering functions to help users quickly find the self-driving tour products they need. Define background as cream; define components with dark teal.

```
<boltAction type="qui_agent_test">
```

Verify cream background and dark-teal buttons. Browse different types of self-driving tour routes, book hotels and packages, and query self-driving club information. Search and filter for self-driving tour products.

</boltAction>

The following is the original website development instruction:

<instruction>{instruction}</instruction>

Trigger the GUI agent testing based on the original instruction in a way similar to the example. **Do not generate additional comments.**

Figure 6: Prompt for generating a GUI-agent testing instruction from the original website specification.

## E ANALYSIS OF THE COMPREHENSIVENESS OF GUI-AGENT TESTING INSTRUCTIONS

To analyze the comprehensiveness of the GUI-agent testing instructions generated by the agent, we manually evaluated the instructions from the experiment runs using Claude-4-Sonnet, Qwen3-Coder-30B-A3B-Instruct, Qwen3-Coder-480B-A35B-Instruct, and DeepSeek-V3. We graded each GUI-agent instruction on a 1–5 scale, determined by how completely the instruction translates each website requirement into concrete GUI-agent checks. The grading guidelines are presented in Fig. 11.

As shown in Tab. 6, 77.2% of the GUI-agent testing instructions across the four models receive a score of 5 (Complete,  $\approx 100\%$  of requirements). Instructions with a score of 4 or higher (High, 75–90%) account for 98.3% of the total, while only 1.7% receive a score of 3 (Moderate, 50–75%); none score below 3. These results indicate that the GUI-agent instructions comprehensively cover most of the website requirements.

## F CATEGORICAL RESULTS

Tab. 7 shows the categorical results of WebGen-Agent with various proprietary and open-source models on WebGen-Bench. As shown in the table, WebGen-Agent consistently achieves superior performance across all instruction and test-case categories compared to other code agent systems. For both the 7B and 8B models, Step-GRPO improves performance in most categories compared to the original instruct model and the SFT model. This demonstrates the effectiveness of the WebGen-

```

972
973
974
975
976
977
978
979
980
981 Prompt: You are given a GUI-agent testing trajectory.
982
983 The GUI agent testing trajectory:
984
985 GUI-Agent Testing Instruction:
986 {gui_instruction}
987 Trajectory:
988 {result}
989
990 Task
991 1. Examine the trajectory for any failed actions that indicate a problem in the website design.
992 2. Decide whether the GUI-agent testing trajectory reveals any flaw in the website implementation.
993     • If yes, set "test_passed": true, and leave "improvementSuggestions" empty.
994     • If no, set "test_passed": false, and write a concise but thorough
995       "improvementSuggestions" that covers the suggested improvements targeting
996       the problems revealed by the testing result.
997 3. Evaluate the results of the GUI-agent test run and assign one integer grade from 1 to 5:
998     • 1: The vast majority of tested functions fail or behave incorrectly.
999     • 2: Many functions fail; only a few behave as expected.
1000     • 3: About half of the functions work as expected; success is mixed.
1001     • 4: Most functions work as expected; only minor issues remain.
1002     • 5: All tested functions work exactly as expected; no issues observed.
1003
1004     Assign the grade to "grade".
1005 Output format (valid JSON)
1006
1007     ````json
1008     {
1009       "test_passed": <boolean>,
1010       "improvementSuggestions": "<string>",
1011       "grade": <int>
1012     }
1013     ````
```

You can first make a short analysis of two or three sentences, then output this JSON object.

Figure 7: Prompt for evaluating GUI-agent testing trajectories and providing improvement suggestions.

1026 Table 5: Accuracy of the screenshot and GUI-agent scores using human annotation as ground truth.  
 1027 For every experiment we report the accuracy together with its standard error.  
 1028

Score Type	Agent-engine LLM	Feedback VLM	Accuracy (%)	Std. Error
Screenshot	Claude-4-Sonnet	Qwen2.5-VL-32B-Inst.	93.6	0.25
	Qwen3-Coder-30B-A3B-Inst.	Qwen2.5-VL-32B-Inst.	93.9	0.26
	Qwen3-Coder-480B-A35B-Inst.	Qwen2.5-VL-32B-Inst.	95.6	0.20
	DeepSeek-V3	Qwen2.5-VL-32B-Inst.	94.8	0.22
	DeepSeek-V3	GPT-4o	95.5	0.20
GUI agent	Claude-4-Sonnet	Qwen2.5-VL-32B-Inst.	90.1	0.31
	Qwen3-Coder-30B-A3B-Inst.	Qwen2.5-VL-32B-Inst.	91.4	0.44
	Qwen3-Coder-480B-A35B-Inst.	Qwen2.5-VL-32B-Inst.	89.6	0.41
	DeepSeek-V3	Qwen2.5-VL-32B-Inst.	91.2	0.36
	DeepSeek-V3	GPT-4o	92.2	0.33

1041  
 1042 Table 6: Distribution (%) of human scores regarding the comprehensiveness of the GUI-agent testing  
 1043 instructions and the resulting average score. The definition of the scores are presented in Fig. 11.  
 1044 The scores range from 1 to 5.

Model	5	4	3	2	1	Avg. Score
Claude-4-Sonnet	84.2	13.9	2.0	0.0	0.0	4.82
DeepSeek-V3	73.3	24.8	2.0	0.0	0.0	4.71
Qwen3-Coder-30B-A3B-Inst.	75.2	23.8	1.0	0.0	0.0	4.74
Qwen3-Coder-480B-A35B-Inst.	76.2	21.8	2.0	0.0	0.0	4.74
Total	77.2	21.0	1.7	0.0	0.0	4.75

1053  
 1054 Agent workflow and the Step-GRPO training process, which incorporates screenshots and GUI-  
 1055 agent feedback.  
 1056

## 1058 G ANALYSIS OF MAXIMUM ITERATION NUMBERS

1060 To analyze the effect of the maximum iteration number parameter on the performance of WebGen-  
 1061 Agent, we test the accuracy, appearance score, and the percentage of samples that exceed the max-  
 1062 imum iteration limit (exceed rate) at different maximum iteration numbers. The agent-engine LLM  
 1063 used is DeepSeek-V3.

1064 As shown in Fig. 12 and Tab. 8, the accuracy and appearance score show a rising trend as the maxi-  
 1065 mum iteration number increases, while the exceed rate continuously decreases. When the maximum  
 1066 iteration number is between 14 and 20, the accuracy, appearance score, and exceed rate all begin to  
 1067 converge. This is because most samples finish before reaching the iteration limit, as reflected by the  
 1068 exceed rate, and the impact of the maximum iteration number on performance diminishes.  
 1069

## 1071 H QUALITATIVE ANALYSIS OF SUPERVISED FINETUNING AND STEP-GRPO

1073 To provide a qualitative analysis of the effects of supervised fine-tuning and Step-GRPO with screen-  
 1074 shot and GUI-agent feedback, we present examples of websites generated by Qwen2.5-Coder-7B-  
 1075 Instruct, Qwen2.5-Coder-7B-Instruct-SFT, and Qwen2.5-Coder-7B-Instruct-Step-GRPO in Figs.13  
 1076 and14. We also include examples of websites generated by Qwen3-8B, Qwen3-8B-SFT, and  
 1077 Qwen3-8B-Step-GRPO in Figs.15 and16. As demonstrated in the examples, supervised fine-tuning  
 1078 greatly reduces the models’ tendency to generate erroneous or malformed websites and improves  
 1079 their ability to follow the appearance requirements specified in the instructions. Step-GRPO further  
 refines the aesthetics and harmony of the generated websites.

1080 Table 7: Categorical results of WebGen-Agent with various proprietary and open-source models on  
 1081 WebGen-Bench (Lu et al., 2025b), compared with other code agent systems. The highest score of  
 1082 each column is marked in **bold**.

Test Name	Instruction Categories			Test-case Categories		
	Content Presentation	User Interaction	Data Management	Functional Testing	Data-Display Testing	Design-Validation
<b>OpenHands</b>						
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
<b>Aider</b>						
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
<b>Bolt.diy</b>						
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-Inst.	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
<b>WebGen-Agent</b>						
<b>Proprietary Models</b>						
Claude-3.5-Sonnet	57.8	48.7	51.9	38.5	60.5	76.2
DeepSeek-R1	57.8	44.2	38.1	35.0	53.8	66.8
DeepSeek-V3	58.0	53.2	45.6	40.9	61.0	72.5
o3	59.2	46.6	53.4	43.7	55.1	68.9
Claude-4-Sonnet	<b>68.7</b>	51.8	52.5	<b>44.0</b>	69.4	71.7
Gemini-2.5-Pro	60.3	48.2	45.6	37.9	60.2	72.5
Qwen3-Coder-480B-A35B-Inst.	64.7	<b>55.8</b>	<b>55.9</b>	43.2	<b>71.2</b>	<b>79.9</b>
<b>Open-Source Models (30B-72B)</b>						
Qwen2.5-Coder-32B-Inst.	35.6	28.8	34.4	20.9	32.3	62.3
Qwen3-Coder-30B-A3B-Inst.	<b>55.2</b>	<b>54.3</b>	<b>47.2</b>	<b>39.1</b>	<b>62.1</b>	<b>76.6</b>
Qwen2.5-72B-Instruct	43.4	30.4	38.8	23.0	39.8	66.0
<b>Open-Source Models (7B-8B)</b>						
Qwen2.5-Coder-7B-Inst.	20.7	8.6	10.9	7.4	15.9	21.3
Qwen2.5-Coder-7B-Inst.-SFT	<b>53.4</b>	33.5	33.8	23.5	48.4	67.6
Qwen2.5-Coder-7B-Inst.-Step-GRPO	51.1	<b>41.1</b>	<b>47.8</b>	<b>30.7</b>	<b>56.7</b>	<b>69.3</b>
Qwen3-8B	37.4	34.3	30.0	26.8	34.1	54.1
Qwen3-8B-SFT	41.7	34.2	<b>43.8</b>	26.8	43.8	63.1
Qwen3-8B-Step-GRPO	<b>52.0</b>	<b>38.8</b>	43.1	<b>30.2</b>	<b>51.1</b>	<b>68.4</b>

## I QUALITATIVE ANALYSIS OF THE WEBGEN-AGENT WORKFLOW

To demonstrate how the WebGen-Agent workflow functions, we provide examples of steps in WebGen-Agent trajectories where the agent improves the website’s appearance based on screenshot or GUI-agent feedback. As shown in Fig. 17, Fig. 18, Fig. 19, Fig. 20, and Fig. 21, the agent enhances the website’s visual appeal by incorporating suggested improvements. Similarly, Fig. 22,

1134  
1135

Table 8: Influence of the maximum number of iterations on agent performance.

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

Fig. 23, Fig. 24, Fig. 25, and Fig. 26 illustrates how the agent refines the website’s functionality based on feedback from the GUI-agent testing process. The steps are simplified due to space constraints.

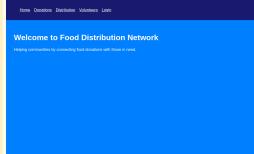
1188

1189

1190 **System Prompt:** You are Bolt, an expert AI assistant and exceptional senior software developer with vast knowledge across  
 1191 multiple programming languages, frameworks, and best practices. <system\_constraints> You are operating in an environment  
 1192 called WebContainer .....

1193 **User:** Please implement a food distribution website for managing and distributing food. The website should have  
 1194 functionalities for food donation, food distribution, and volunteer management. Users should be able to browse food donation  
 1195 information, apply for food distribution, view volunteer information, and manage their own accounts. Use azure for the  
 1196 background and midnight blue for components.

1197 **Assistant:** I'll create a food distribution website with the requested functionalities and color scheme. Here's the  
 1198 implementation plan: 1. Set up Vite + React with required dependencies 2. Create components for donation, distribution, and  
 1199 volunteer management .....

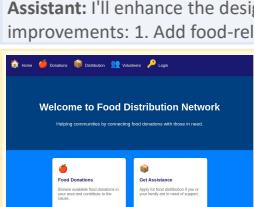


1200 Installation of dependencies was successful. Starting the service was successful. **The**  
 1201 **screenshot description:** The webpage has a clean and simple layout with a dark blue header  
 1202 containing navigation links: 'Home', 'Donations', 'Distribution', 'Volunteers', and 'Login' .....

1203 **Suggestions for Improvement:** Consider adding more visual elements such as icons .....

1204 If no further modification is necessary, output <boltAction type="screenshot\_validated"/> to signal  
 1205 that the screenshot is satisfactory.

Score<sub>shot</sub> = 3, Score<sub>gui</sub> = 0



1206 Installation of dependencies was successful. Starting the service was successful. **The**  
 1207 **screenshot description:** The screenshot shows a clean and organized homepage for a website  
 1208 called 'Food Distribution Network.' The overall layout includes a top navigation bar with icons  
 1209 and links labeled 'Home,' 'Donations,' 'Distribution,' 'Volunteers,' and 'Login.' Below the  
 1210 navigation bar, there is .....

1211 If no further modification is necessary, output <boltAction type="screenshot\_validated"/> to signal that the screenshot is satisfactory.

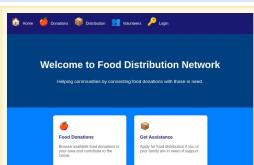


1212 **The suggestions based on the GUI-agent testing result:**  
 1213 Address the root cause of the blank page issue, which could  
 1214 involve server-side errors, frontend rendering problems, or  
 1215 resource loading failures. Implement robust error handling  
 1216 and loading indicators to provide feedback to users. Ensure  
 1217 the website is thoroughly tested in staging environments to  
 1218 catch such critical issues before deployment .....

Score<sub>shot</sub> = 5, Score<sub>gui</sub> = 1



1219 **Assistant:** I'll add error handling and loading states to prevent blank page issues. Here are the key improvements: 1. Add  
 1220 loading states for all pages .....

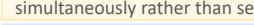


1221 Installation of dependencies was successful. Starting the service was successful. **The**  
 1222 **screenshot description:** The screenshot shows a clean and organized homepage for a food  
 1223 distribution network. The overall layout includes a top navigation bar with icons and links  
 1224 labeled 'Home', 'Donations', 'Distribution', 'Volunteers', and 'Login'. Below the navigation bar,  
 1225 there is a prominent heading 'Welcome to.....' If no further modification is necessary, output  
 1226 <boltAction type="screenshot\_validated"/> to signal that the screenshot is satisfactory.

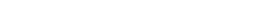
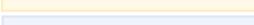
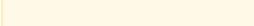
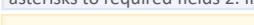


1227 **The suggestions based on the GUI-agent testing result:** Improve the Food Distribution Application form by: 1) Clearly  
 1228 marking all mandatory fields with an asterisk or placeholder text before submission, 2) Providing a summary of required  
 1229 fields upfront to prevent partial submissions, and 3) Implementing client-side validation to highlight missing fields  
 1230 simultaneously rather than sequentially.

Score<sub>shot</sub> = 5, Score<sub>gui</sub> = 3



1231 **Assistant:** I'll enhance the distribution form with better validation and field requirements. Here are the improvements: 1. Add  
 1232 asterisks to required fields 2. Implement client-side validation 3. Show all validation errors at once 4. Add a summary of .....

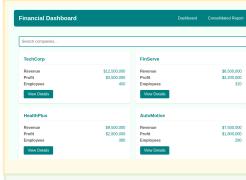


1242  
1243  
1244  
1245  
1246

**System Prompt:** You are Bolt, an expert AI assistant and exceptional senior software developer with vast knowledge across multiple programming languages, frameworks, and best practices. <system\_constraints> You are operating in an environment called WebContainer .....

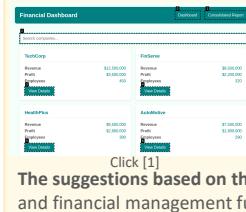
**User:** Please implement a multi-company dashboard for managing and displaying financial data from multiple companies. The dashboard should be able to collect and display financial information from each company, provide consolidated reports, and support cross-company comparisons and reporting. Users 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.

**Assistant:** I'll create a multi-company financial dashboard using React with Vite. Here's the implementation plan: 1. Set up Vite + React project 2. Create components for company data display and comparison 3. Implement mock financial data storage 4. Style with mint cream background and teal components.....



Installation of dependencies was successful. Starting the service was successful. **The screenshot description:** The screenshot shows a financial dashboard with a clean and organized layout. The top section features a teal header with the title 'Financial Dashboard' and navigation options labeled 'Dashboard' and 'Consolidated Report'. Below the header is a search bar for filtering companies. The main content area is divided into four equal-sized cards ..... If no further modification is necessary, output <boltAction type="screenshot\_validated"/> to signal that the screenshot is satisfactory.

<boltAction type="screenshot\_validated"/> ✓

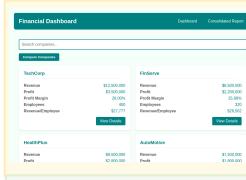


Installation of dependencies was successful. Starting the service was successful. **The screenshot description:** The screenshot shows a financial dashboard with a clean and organized layout. The top section features a teal header with the title 'Financial Dashboard' and navigation options labeled 'Dashboard' and 'Consolidated Report'. Below the header is a search bar for filtering companies. The main content area is divided into four equal-sized cards ..... If no further modification is necessary, output <boltAction type="screenshot\_validated"/> to signal that the screenshot is satisfactory.

**The suggestions based on the GUI-agent testing result:** Improve navigation and accessibility of cross-company comparison and financial management functionalities to allow thorough testing. Ensure all critical features are easily reachable and not hidden behind repetitive interactions. Consider adding more interactive elements or drill-down options for detailed financial data exploration .....

*Score<sub>shot</sub> = 4, Score<sub>gui</sub> = 2*

**Assistant:** I'll enhance the dashboard with better comparison features and financial management tools. Here are the key improvements: 1. Add comparison mode to select multiple companies 2. Include financial ratios and metrics 3. Add export functionality for reports.....



Installation of dependencies was successful. Starting the service was successful. **The screenshot description:** The screenshot shows a financial dashboard with a clean and organized layout. The top section includes a header with the title 'Financial Dashboard' and navigation options labeled 'Dashboard' and 'Consolidated Report'. Below the header is a search bar labeled 'Search companies...'. A button labeled 'Compare Companies' is ..... If no further modification is necessary, output <boltAction type="screenshot\_validated"/> to signal that the screenshot is satisfactory.

<boltAction type="screenshot\_validated"/> ✓



Installation of dependencies was successful. Starting the service was successful. **The screenshot description:** The screenshot shows a financial dashboard with a clean and organized layout. The top section includes a header with the title 'Financial Dashboard' and navigation options labeled 'Dashboard' and 'Consolidated Report'. Below the header is a search bar labeled 'Search companies...'. A button labeled 'Compare Companies' is ..... If no further modification is necessary, output <boltAction type="screenshot\_validated"/> to signal that the screenshot is satisfactory.

Click [1] <boltAction type="screenshot\_validated"/> ✓

Click [2]

The trajectory shows that the GUI agent successfully verified the mint cream background and teal components, browsed financial data, viewed consolidated reports, and performed cross-company comparisons. The GUI agent testing is successful and no further improvement is necessary. Observe the above feedback and decide whether further modifications to the code are needed based on the GUI-agent testing summary. If no further modification is necessary, output <boltAction type="finish"/>

*Score<sub>shot</sub> = 4, Score<sub>gui</sub> = 5*

<boltAction type="finish"/> ✓

Figure 9: Example of a WebGen-Agent trajectory.

1290  
1291  
1292  
1293  
1294  
1295

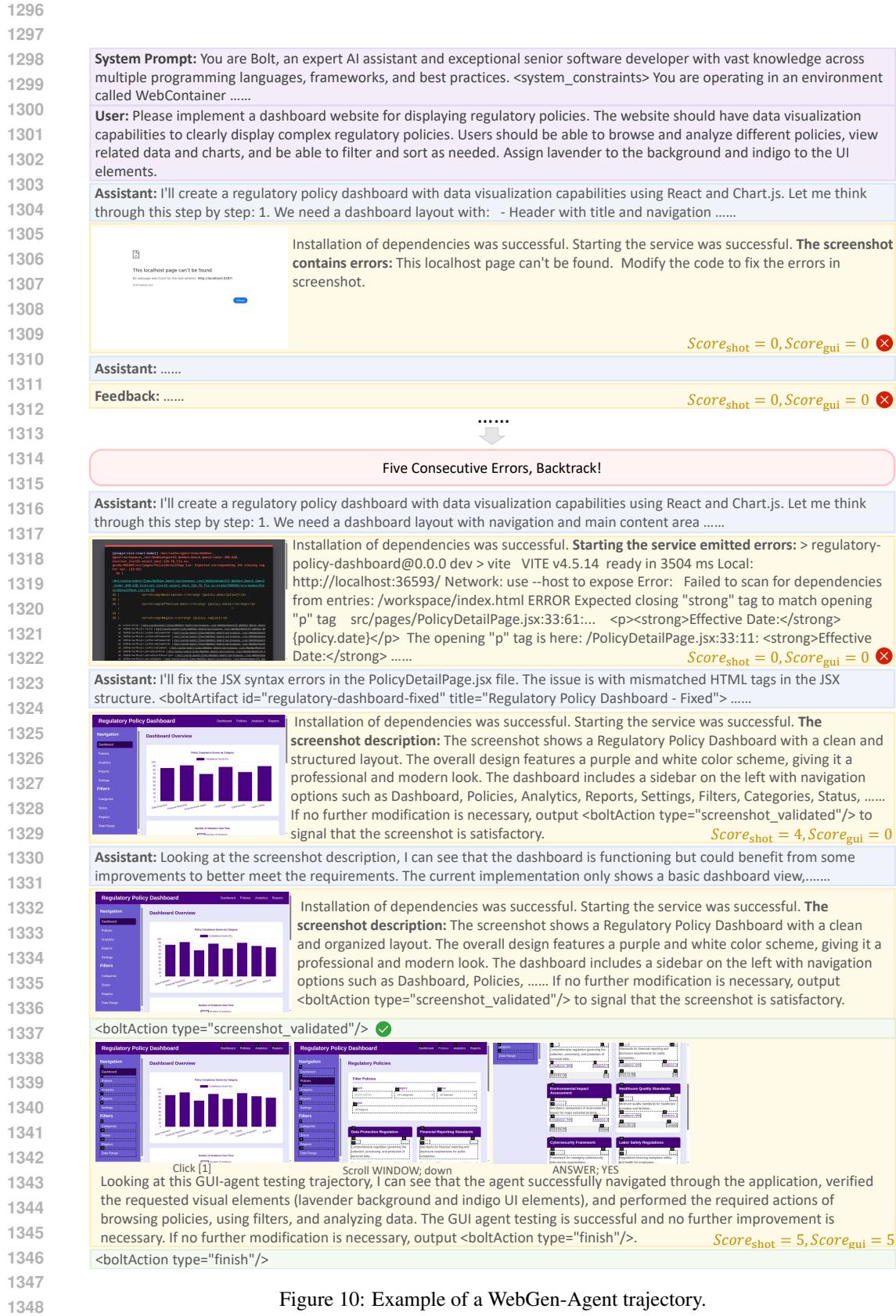


Figure 10: Example of a WebGen-Agent trajectory.

1350  
1351  
1352  
1353  
1354  
1355**GUI-agent Instruction Evaluation Guidelines:**

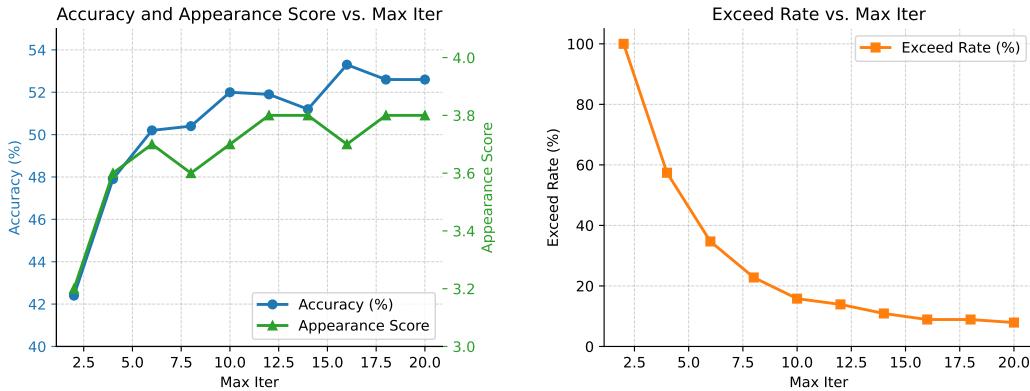
Score the instruction 1 – 5, where 5 = best. The dominant criterion is comprehensiveness: how completely the instruction translates every website requirement into concrete GUI-agent checks.

1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370**Grading Scale:**

- **1 (Minimal, < 25 %):** The instruction overlooks most of the stated requirements.
- **2 (Low, 25 – 50 %):** Only some primary requirements are mentioned; many important items are absent.
- **3 (Moderate, 50 – 75 %):** Core functionalities are covered, but several secondary features or style rules are skipped.
- **4 (High, 75 – 90 %):** All major functional requirements plus most visual or secondary ones are included; only a few minor details are missing.
- **5 (Complete,  $\approx 100\%$  of requirements):** Every requirement is turned into checks. Nothing significant is left out.

1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403

Figure 11: Grading guidelines for manually evaluating GUI-agent testing instructions



(a) Accuracy (%) and Appearance Score as a function of the maximum number of iterations. (b) Exceed Rate (%) versus the maximum number of iterations.

Figure 12: Effect of the maximum iteration number hyper-parameter on different performance metrics.

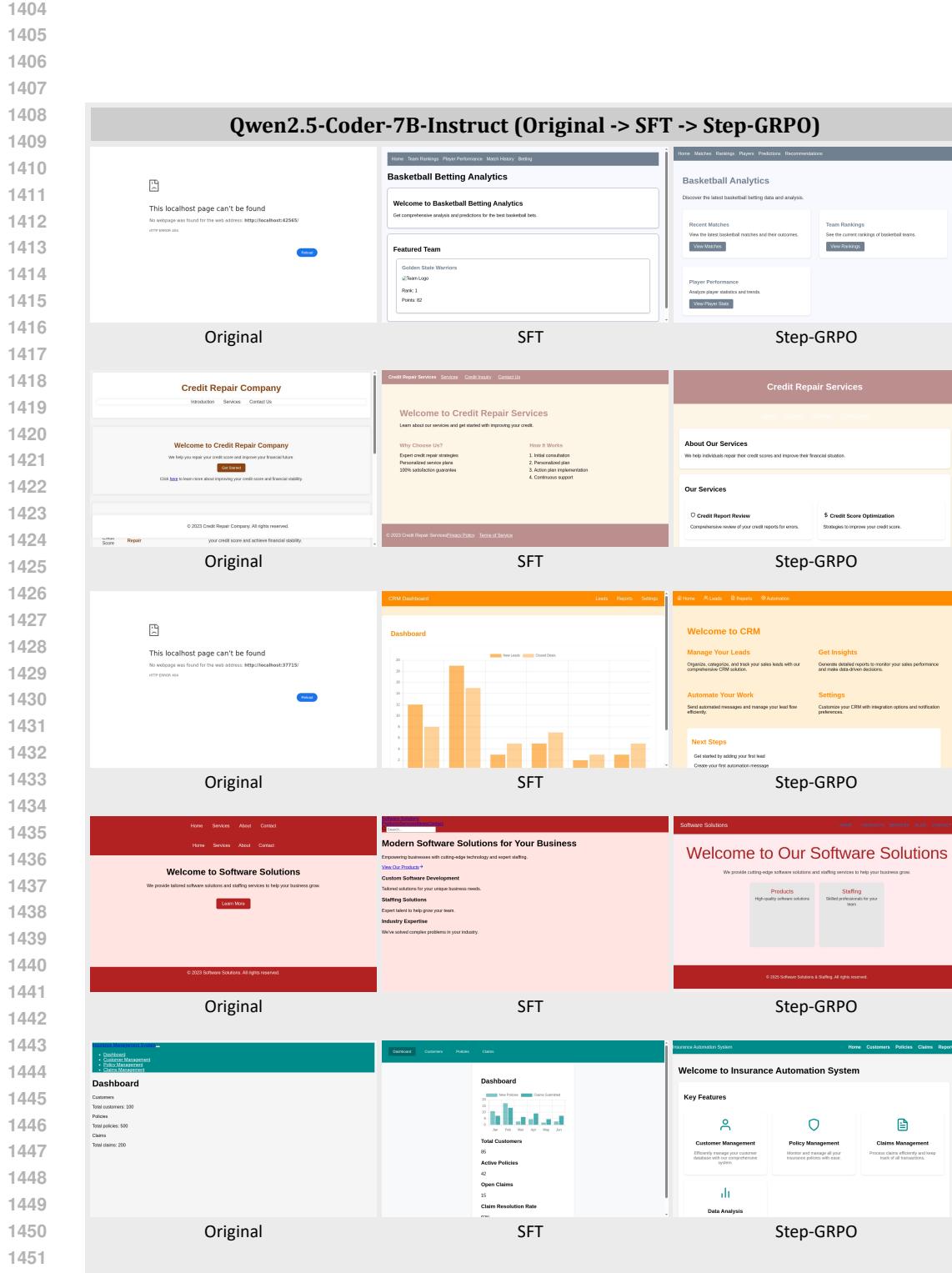


Figure 13: Screenshots of websites created by Qwen2.5-Coder-7B-Instruct, Qwen2.5-Coder-7B-Instruct-SFT, and Qwen2.5-Coder-7B-Instruct-Step-GRPO.

1458

1459

1460

1461

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

1472

1473

1474

1475

1476

1477

1478

1479

1480

1481

1482

1483

1484

1485

1486

1487

1488

1489

1490

1491

1492

1493

1494

1495

1496

1497

1498

1499

1500

1501

1502

1503

1504

1505

1506

1507

1508

1509

1510

1511

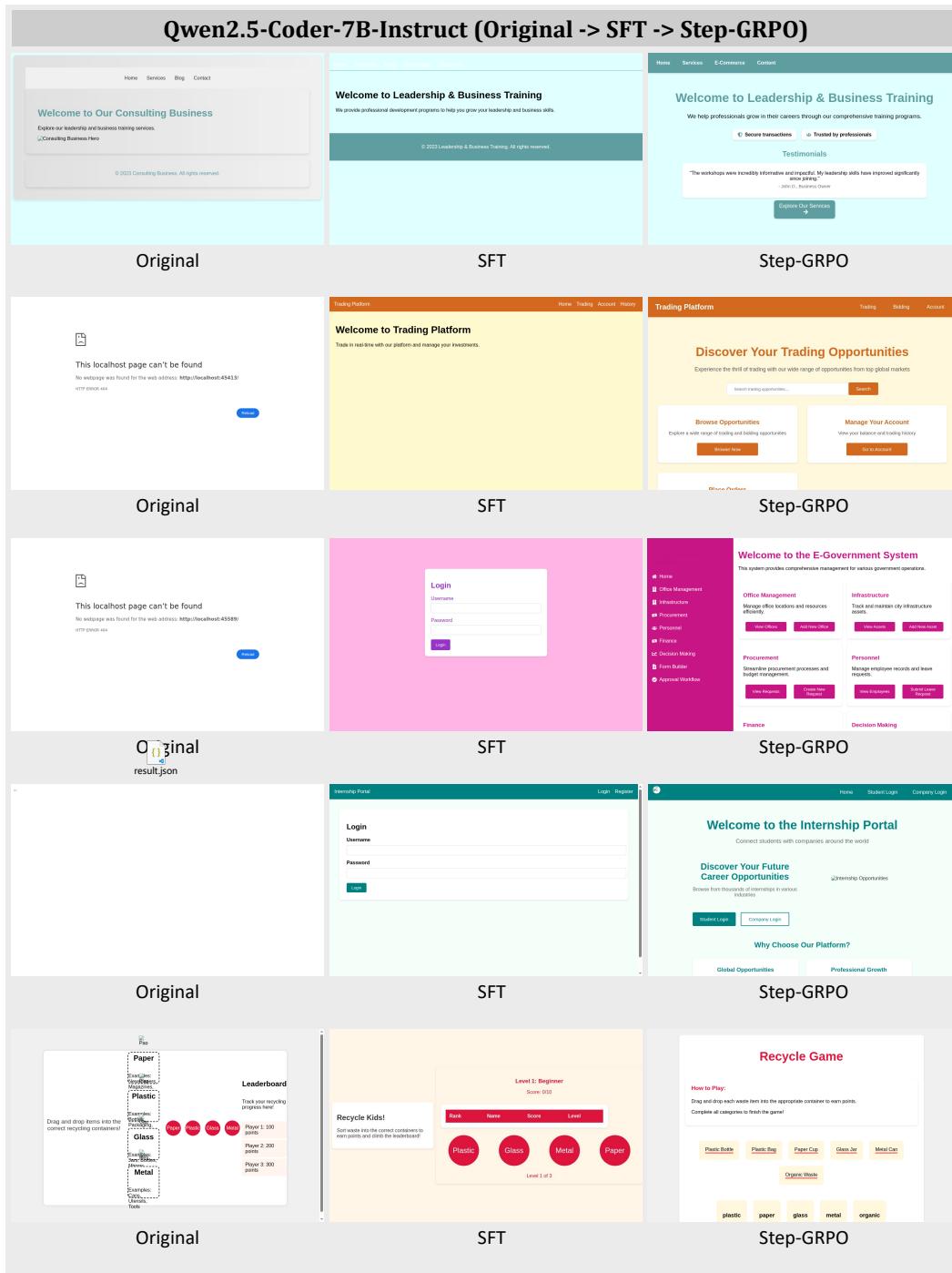


Figure 14: Screenshots of websites created by Qwen2.5-Coder-7B-Instruct, Qwen2.5-Coder-7B-Instruct-SFT, and Qwen2.5-Coder-7B-Instruct-Step-GRPO.

1512

1513

1514

1515

1516

1517

1518

1519

1520

1521

1522

1523

1524

1525

1526

1527

1528

1529

1530

1531

1532

1533

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

1548

1549

1550

1551

1552

1553

1554

1555

1556

1557

1558

1559

1560

1561

1562

1563

1564

1565

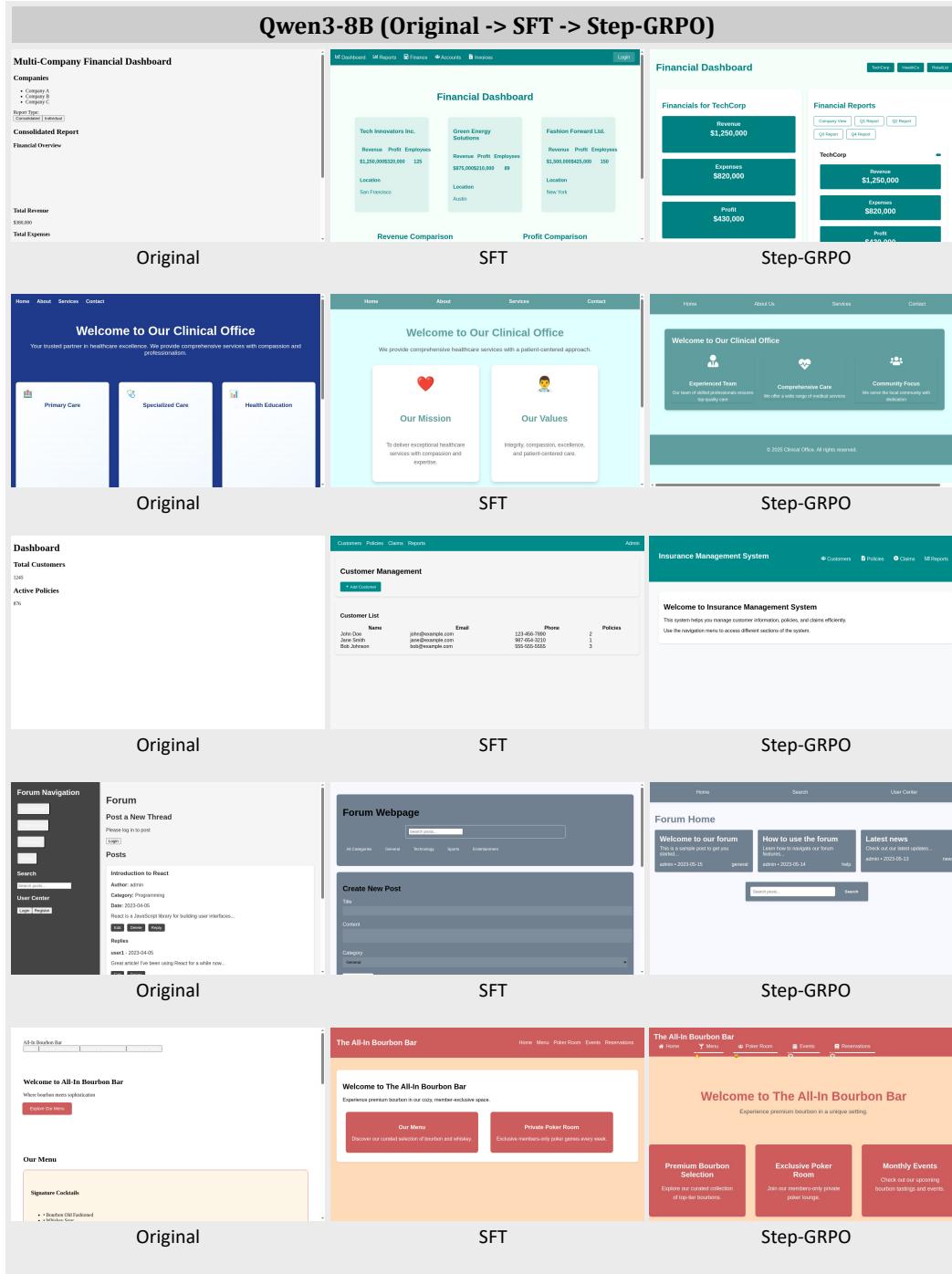


Figure 15: Screenshots of websites created by Qwen3-8B, Qwen3-8B-SFT, and Qwen3-8B-Step-GRPO.

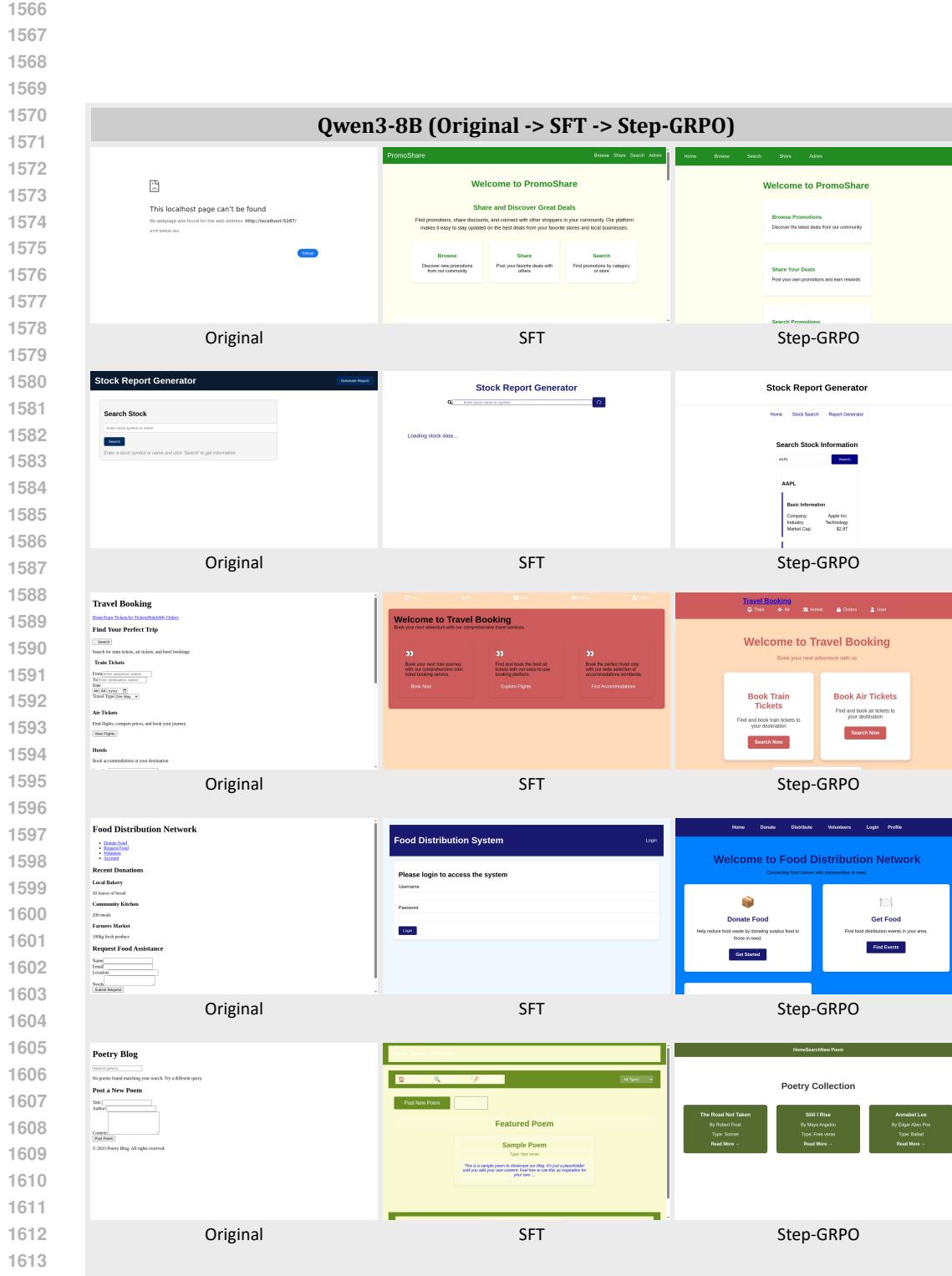


Figure 16: Screenshots of websites created by Qwen3-8B, Qwen3-8B-SFT, and Qwen3-8B-Step-GRPO.

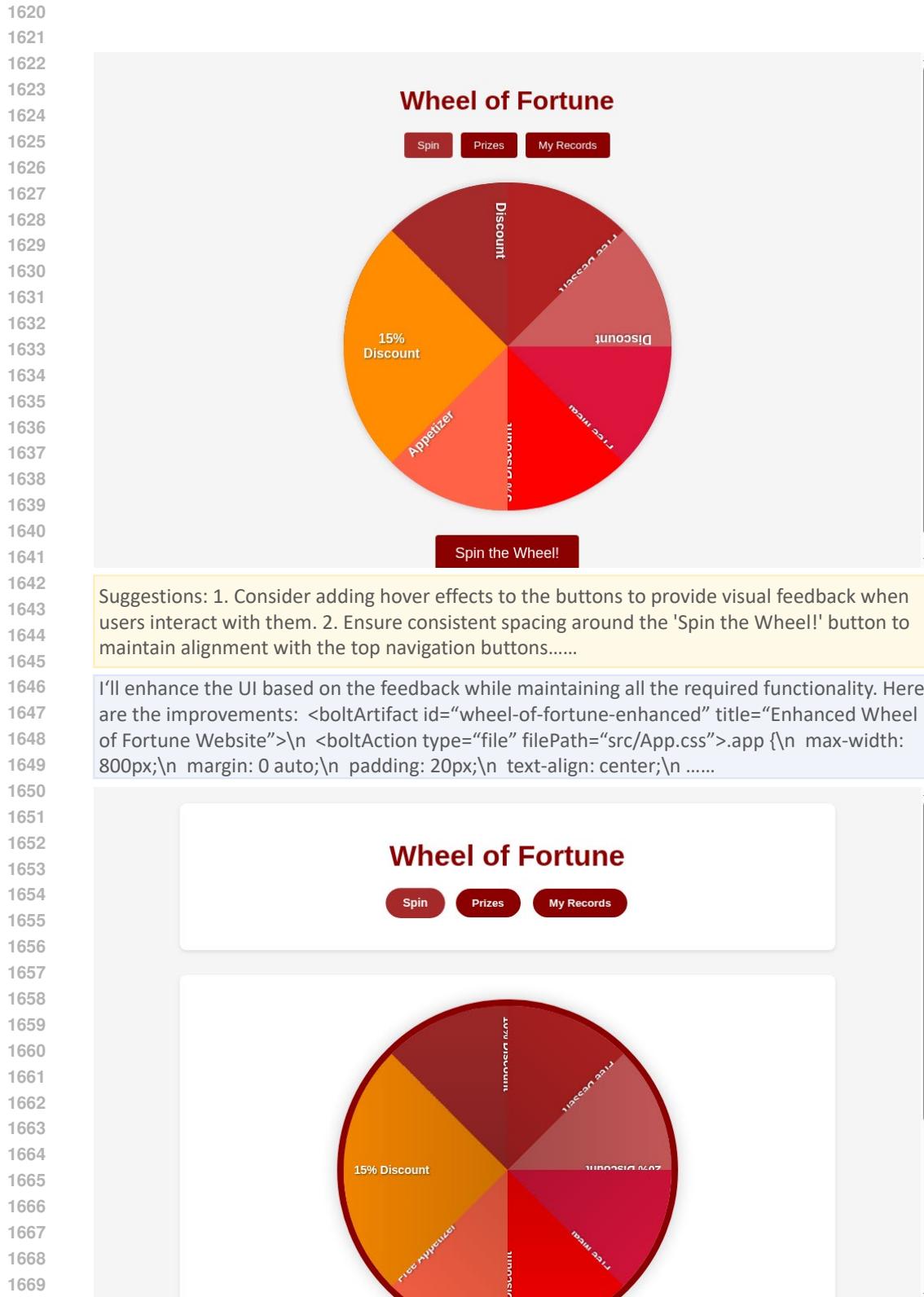


Figure 17: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on screenshot feedback. The step is simplified due to space constraints.

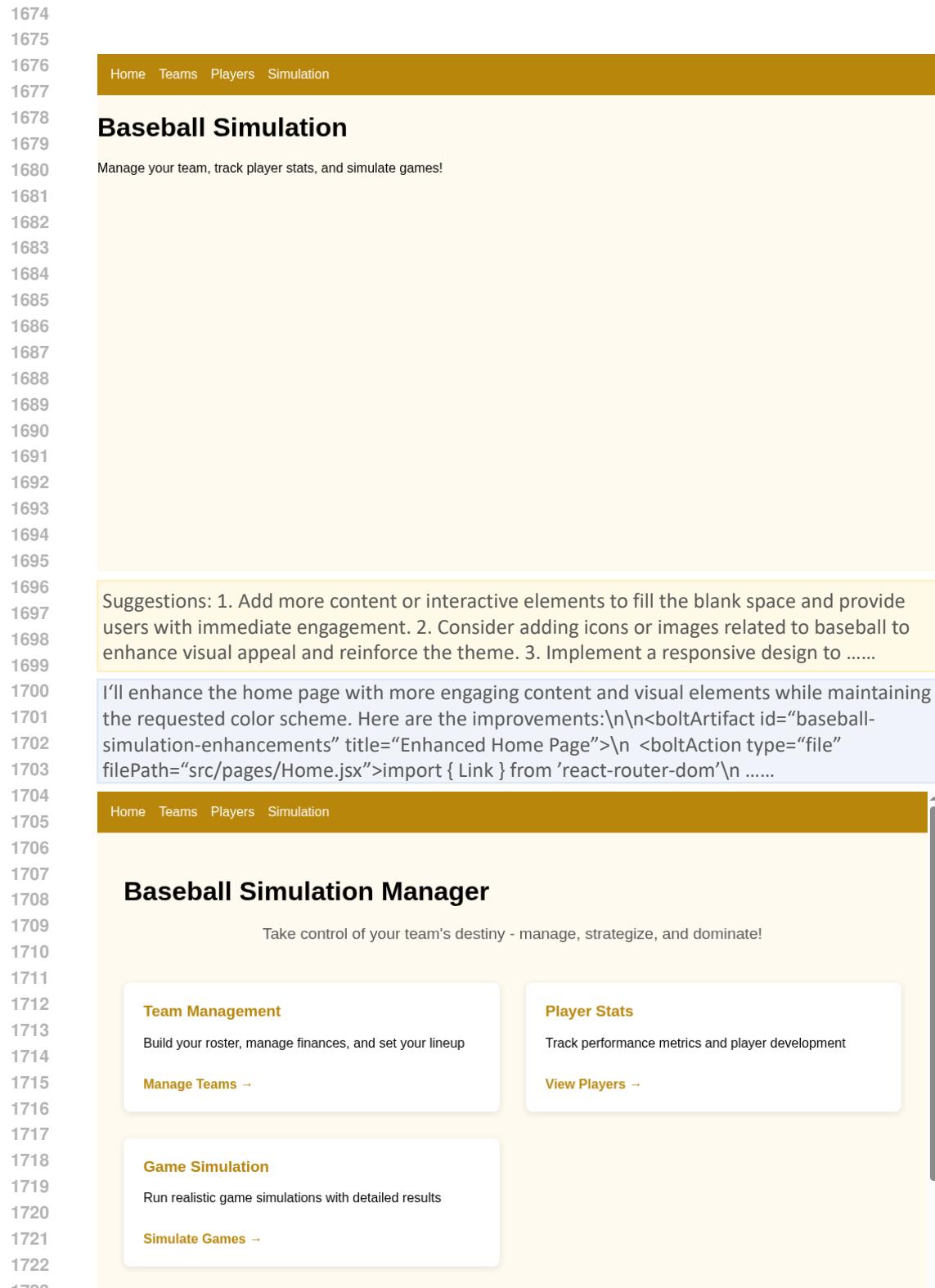


Figure 18: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on screenshot feedback. The step is simplified due to space constraints.

1728

1729

1730 [Home](#) [About Us](#) [Services](#) [Contact Us](#)

1731

1732

1733 **Welcome to Our Clinical Office**

1734

1735 We provide comprehensive healthcare services with a patient-centered approach.

1736 Our team of experienced professionals is dedicated to your well-being.

1737

1738

1739 **Office Hours**

1740 Monday - Friday: 8:00 AM - 5:00 PM

1741 Saturday: 9:00 AM - 1:00 PM

1742 Sunday: Closed

1743

1744

1745

1746

1747

1748

1749

1750 Suggestions: Consider adding some visual elements such as icons or images related to

1751 healthcare to enhance engagement. Additionally, introducing a footer with contact information

1752 or social media links could improve user experience.....

1753

1754 I'll enhance the website by adding a footer with contact info and some healthcare icons to

1755 improve visual appeal. Here are the modifications:  
<boltArtifact id=\"clinical-office-website-enhanced\" title=\"Enhanced Clinical Office Website\"><boltAction type=\"file\" filePath=\"src/index.css\">\* {\nmargin: 0;\n.....

1756

1757

1758 [Home](#) [About Us](#) [Services](#) [Contact Us](#)

1759

1760

1761 **Welcome to Our Clinical Office**

1762

1763

1764 We provide comprehensive healthcare services with a patient-centered approach.

1765 Our team of experienced professionals is dedicated to your well-being.

1766

1767 **Office Hours**

1768 Monday - Friday: 8:00 AM - 5:00 PM

1769 Saturday: 9:00 AM - 1:00 PM

1770 Sunday: Closed

1771

1772

1773

1774  123 Medical Center Drive

1775 Suite 100

1776

1777  (555) 123-4567

1778 Mon-Fri: 8AM-5PM

1779  info@clinicaloffice.com

1780 Emergency: (555) 987-6543

1781

Figure 19: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on screenshot feedback. The step is simplified due to space constraints.

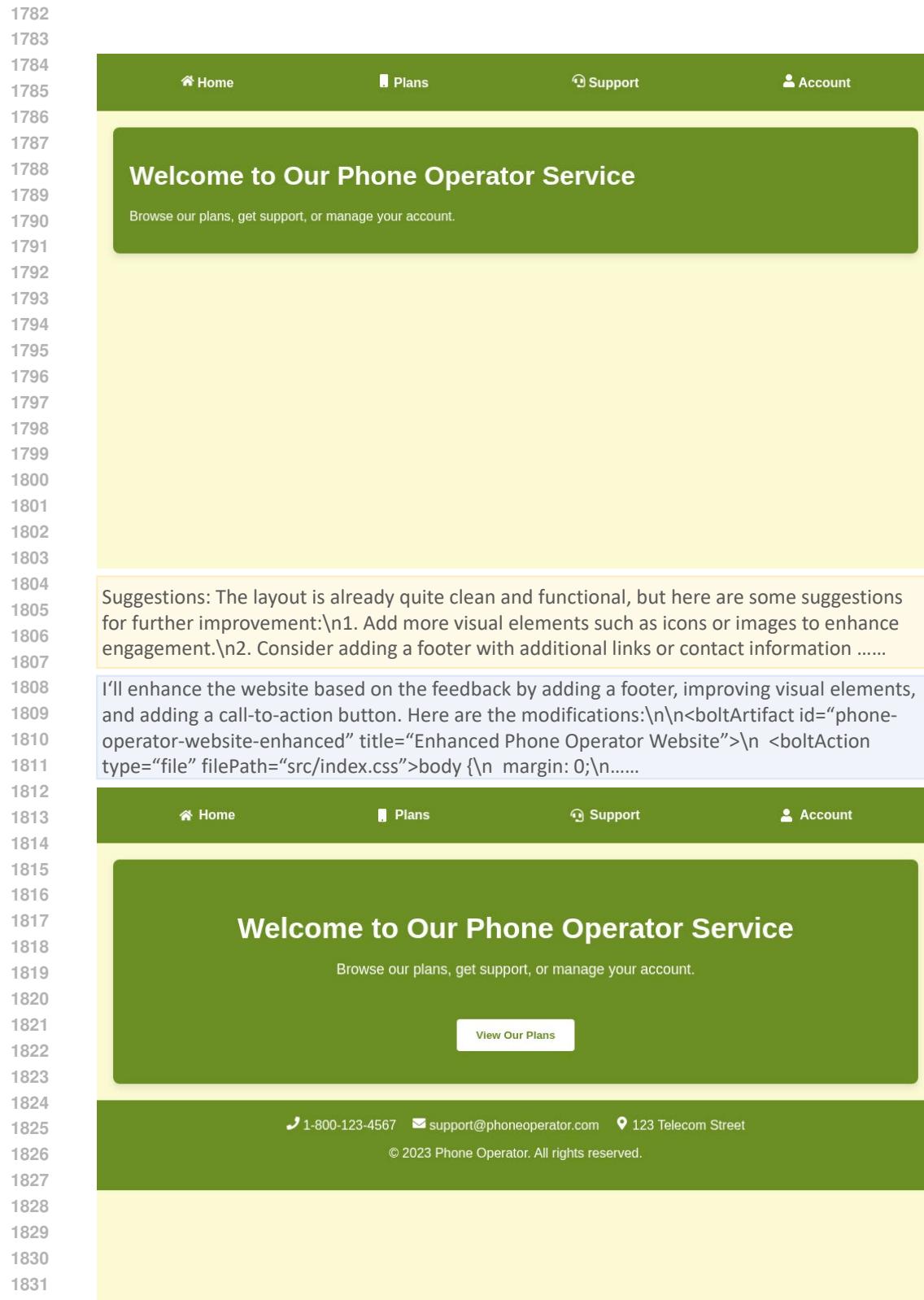


Figure 20: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on screenshot feedback. The step is simplified due to space constraints.

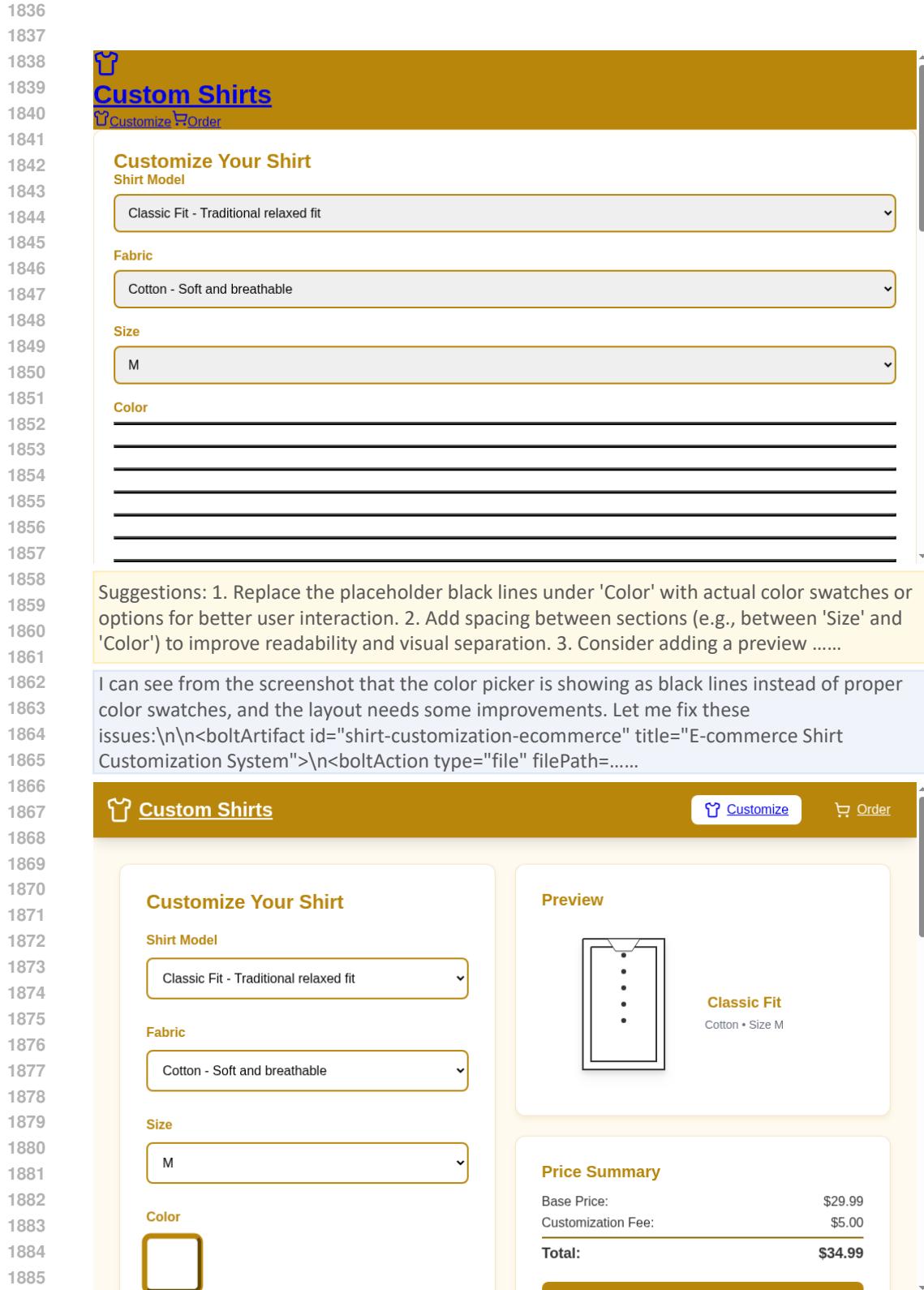


Figure 21: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on screenshot feedback. The step is simplified due to space constraints.

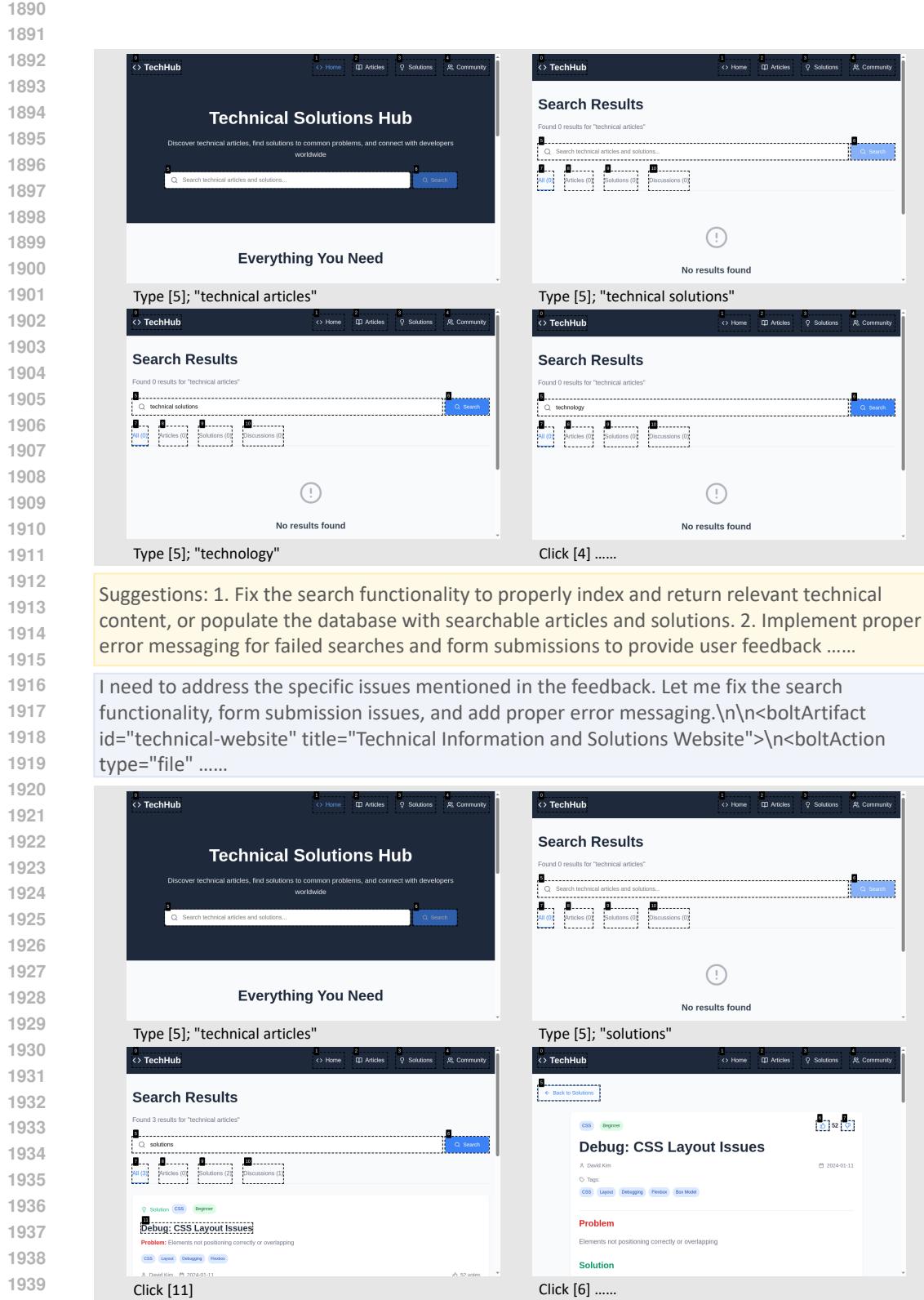


Figure 22: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on GUI-agent testing feedback. The step is simplified due to space constraints.



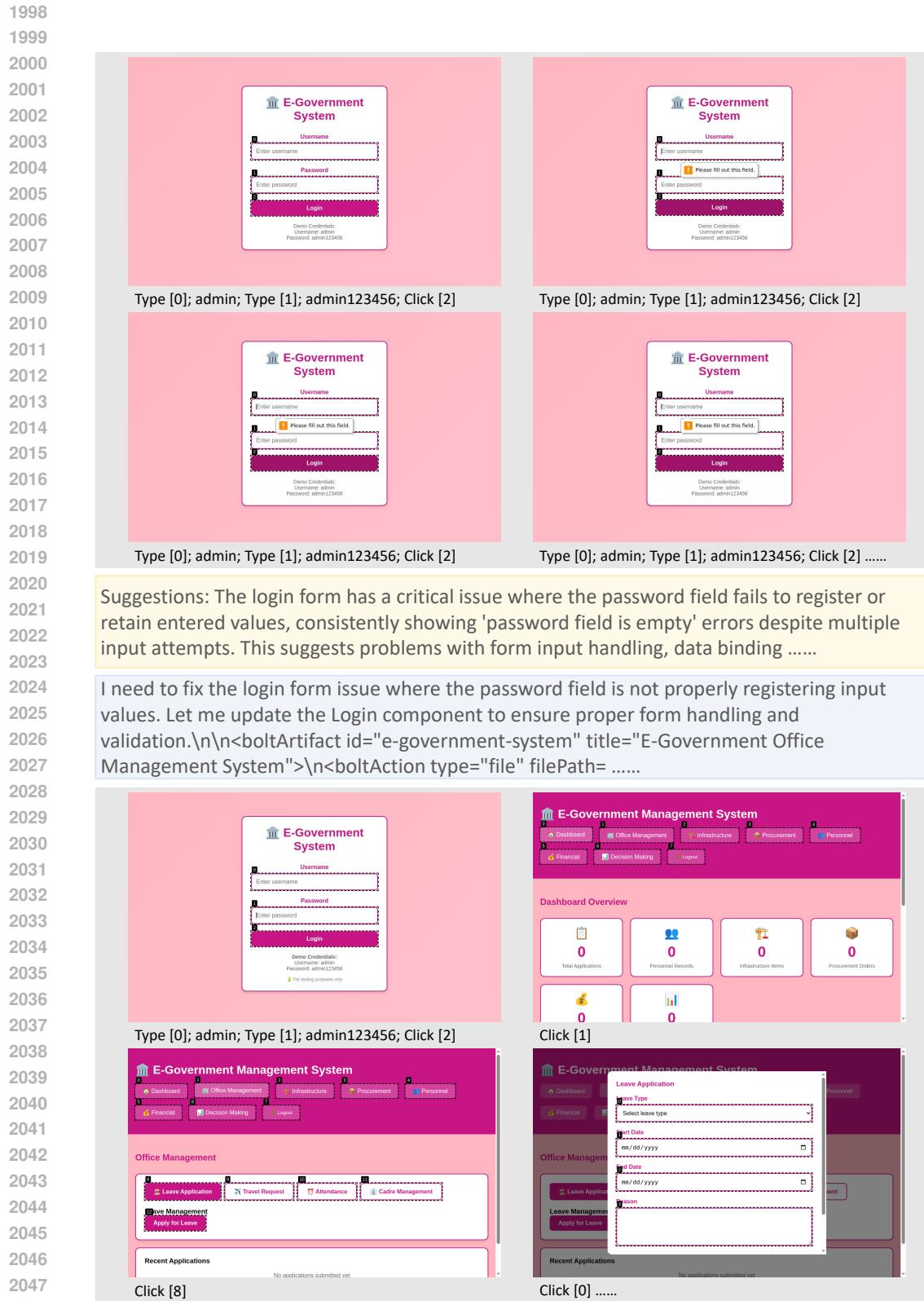


Figure 24: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on GUI-agent testing feedback. The step is simplified due to space constraints.

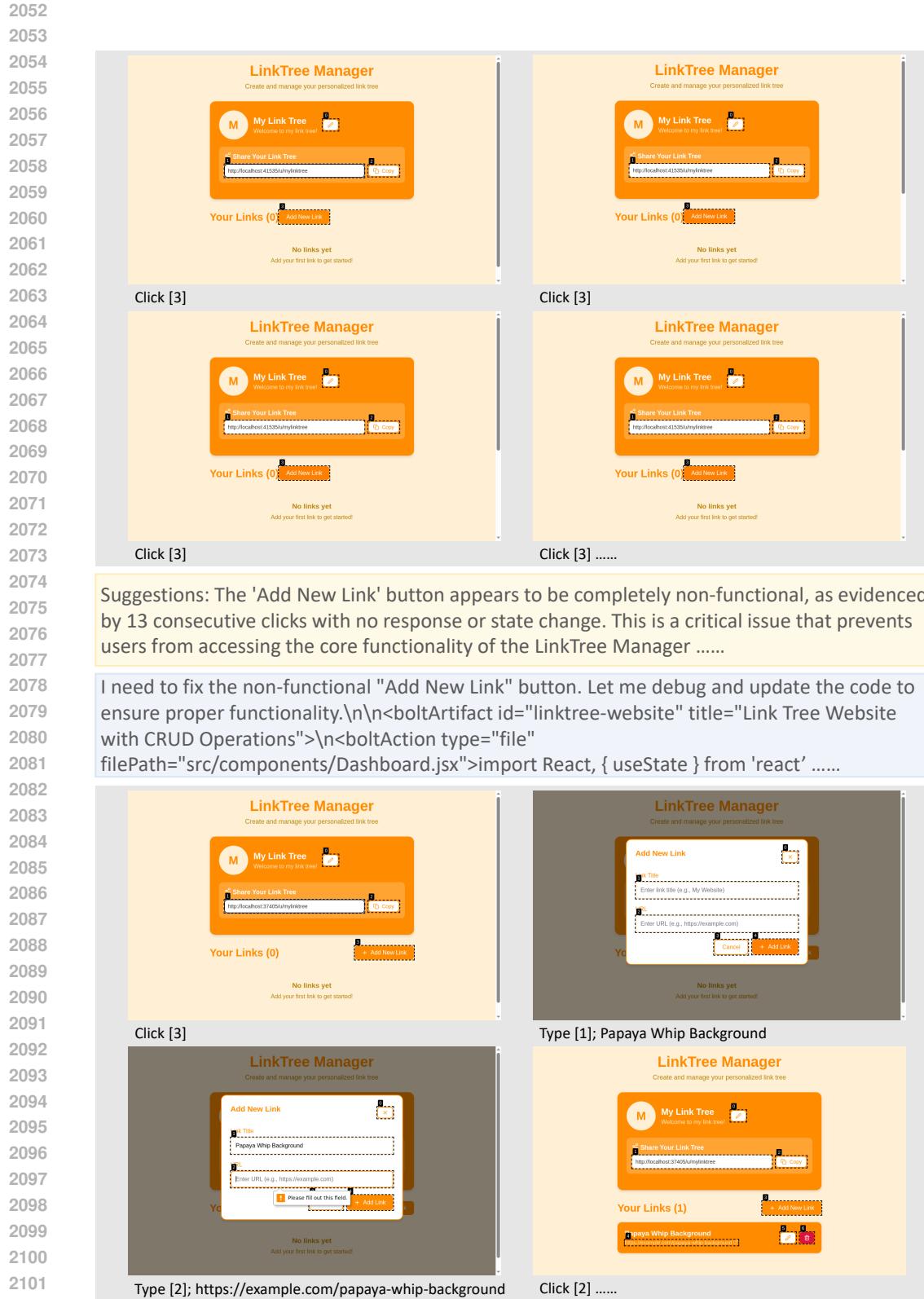


Figure 25: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on GUI-agent testing feedback. The step is simplified due to space constraints.



Figure 26: Example of a step in a WebGen-Agent trajectory where the agent improves the website's appearance based on GUI-agent testing feedback. The step is simplified due to space constraints.