# FRONTALK: Benchmarking Front-End Development as Conversational Code Generation with Multi-Modal Feedback

**Xueqing Wu**[†1] **Zihan Xue**[1] **Da Yin**[2] **Shuyan Zhou**[23] **Kai-Wei Chang**[1] **Nanyun Peng**[1] **Yeming Wen**[2]

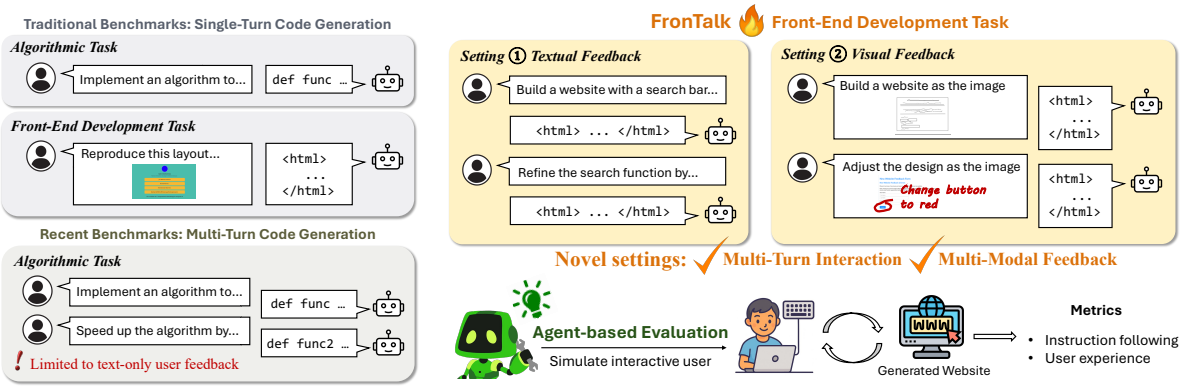



*Figure 1.* Focusing on the front-end development task, **FRONTALK** benchmark explores a novel setting with multi-turn interactions and multi-modal user feedback, along with an agent-based framework for evaluating model-generated websites.

## Abstract

Multi-turn, multi-modal interaction is a common interaction pattern in human–AI collaborative coding, where users iteratively refine implementations using both language and visual feedback. For example, in front-end development, users often combine textual instructions with visual artifacts such as sketches, mockups, and annotated screenshots. Despite its prevalence, such workflow remains largely overlooked: existing research primarily studied simplified single-turn or text-only settings. To address this gap, we introduce **FRONTALK**, a benchmark for **multi-turn, multi-modal code generation** in front-end development. FRONTALK has 100 multi-turn dialogues from diverse real-world websites. Each turn includes a textual instruction and an equivalent visual instruction. We further propose a novel *agent-based evaluation framework* that leverages a web agent to simulate user interactions and measure both implementation correctness and user experience of the generated websites. Evaluation of 20 models reveals two fundamental challenges: (1) a *forgetting issue*, where models often overwrite previously implemented features, and (2) a *visual interpretation gap*, where models struggle to translate visual inputs into functional requirements. To address these challenges, we propose ACECODER, which employs a web agent to autonomously critique implementations and verify compliance with prior instructions. This reduces the forgetting rate from 8.0–28.2% to **nearly zero**. In terms of overall performance, ACECODER improves the textual feedback setting by up to **9.3%** (56.0%→65.3%) and also mitigates the visual interpretation challenge, boosting performance by up to **5.2%** (55.0%→60.2%) with visual feedback.

## 1. Introduction

The success of large language models (LLMs) in code generation has driven a surge in conversational AI-assisted coding workflows (sometimes referred to as "vibe coding" (Karpathy, 2025)) and commercial coding assistants, where **multi-turn interaction** is central. Reflecting this trend, recent benchmarking efforts have moved beyond single-turn code generation (Austin et al., 2021; Chen et al., 2021) to better measure how effectively LLMs respond to feedback across multiple turns (Wang et al.; Laban et al., 2025; Han et al.). However, these benchmarks predominantly focus on algo-

[†]Work done at Meta. [1]University of California, Los Angeles [2]Meta Superintelligence Labs [3]Duke University.**AUTHORERR: Missing \icmlcorrespondingauthor.**

*Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026.* Copyright 2026 by the author(s).

rithmic or API-driven tasks, where multi-modal feedback is less central, and thus limit their evaluation to text-only interactions. This results in under-exploration of domains such as data visualization, application design, and front-end development where **multi-modal feedback** is a critical communication channel. For example, in front-end development, sketching a UI layout or annotating a screenshot is a natural and efficient method for communicating design changes. Therefore, this unique dynamic of multi-turn, multi-modal coding remains a significantly underexplored challenge.

To advance this frontier, we introduce FRONTALK, a benchmark for front-end development that integrates multi-turn code generation with multi-modal feedback. Our benchmark begins with a diverse and realistic collection of *user intents*, derived from real-world websites in the C4 dataset (Raffel et al., 2020) and spanning diverse domains such as e-commerce platforms, financial service sites, and digital art portfolios. To capture realistic conversational dynamics, we employ an LLM-based *user simulator* that generates context-aware instructions conditioned on prior dialogue. Crucially, to enable our simulated users to generate visual feedback, we equip them with a suite of *drawing tools* to communicate intents through sketching and annotating. The resulting FRONTALK dataset comprises 1,000 conversational turns across 100 dialogues, paired with 3,676 manually refined test cases, offering a robust foundation for evaluating multi-modal, multi-turn coding systems.

In terms of evaluation, we propose an **agent-based evaluation framework** to evaluate both *instruction following* and *user experience*. For instruction following, we measure *pass rate* against human-annotated test cases using an automated web agent to interact with the website and verify task completion. This agent is uniquely equipped with *image manipulation tools* (e.g., cropping, image comparison), enhancing model's visual perception capabilities and improving alignment by 1.7% with human annotators on design tasks. For the nuanced dimension of user experience, we employ a *pairwise comparison* protocol. In this setup, one LLM simulates a first-time user, exploring the website to learn its interface while attempting tasks it proposes itself. Then a secondary LLM compares the resulting trajectories and judges which interface is more *usable*, with criteria grounded in learning efficiency and satisfaction in task completion. Our evaluation protocol achieves moderate to substantial agreement with human annotators, with higher alignment on implementation correctness than on the inherently more subjective usability dimension.

We evaluated a range of open- and closed-source models on our FRONTALK benchmark, including eight text-only LLMs and 12 VLMs. Our experiments reveal several key findings:

1. **Open-source vs. proprietary gap**: Proprietary models, such as Gemini-2.5-Pro, maintain a substantial lead over open-source models. This disparity is particularly pronounced with visual feedback, where the performance gap is 24.1%, compared to the 12.5% gap with textual feedback. Since FRONTALK is the first to evaluate multi-turn multi-modal code generation, we attribute this gap to the superior generalization capabilities of proprietary models, an area where contemporary open-source VLMs still fall short.

2. **Forgetting issue:** The multi-turn setting introduces a significant forgetting issue, causing previously implemented functions or design elements to be contradicted or overwritten by later code. This issue consistently occurs across all leading models and results in a performance degradation of up to 46%, as measured by the *forgetting rate* metric.

3. **Visual interpretation gap**: Visual feedback proves consistently more difficult than textual feedback, resulting in significantly lower pass rate. Common failure modes include (1) the literal replication of layout sketches without implementing the underlying functionality, and (2) overlooking textual annotations in densely annotated images.

4. **Key factors affecting usability**: The determinants of usability evolve with model capability. For weaker models, poor usability primarily stems from *broken functionality*, while for stronger models, it is driven by *design flaws* such as inefficient navigation or missing interactive feedback. This highlights that achieving high usability requires implementing implicit, common-sense design principles that are often omitted from user instructions.

While most of these issues stem from fundamental model capabilities, the forgetting issue is uniquely tied to the multi-turn setting. Similar phenomena have been observed in multi-turn chatting benchmarks, where simply reiterating prior instructions can offer a partial remedy (Laban et al., 2025). However, code generation requires not only memorizing prior instructions but also resolving implementation-level conflicts between prior and later requirements, a challenge that cannot be addressed by mere repetition. To address this challenge, we propose ACECODER (**a**gent **c**ritique for instruction **e**nhancement), a simple yet effective method tailored for multi-turn code generation. At each turn, after an initial code solution is generated, an LLM-powered web agent autonomously explores the implemented website to verify alignment with both current and past instructions. The identified coding errors or oversights are then used to rewrite the code and to explicitly prevent these pitfalls. Experiments demonstrate that ACECODER reduces the forgetting rate to **nearly zero**, improves the final performance by up to **9.3%** with textual instructions, and also benefits the visual instructions setting by up to **5.2%**.

In summary, our contributions are as follows: (1) We propose FRONTALK, a realistic benchmark for *multi-turn* front-end coding with both *textual* and *visual* feedback; (2) We identify two critical challenges in current models: a prevalent *forgetting issue* and a significant difficulty in *interpreting visual feedback*; (3) We propose ACECODER, an agent-based critique method that effectively reduces forgetting and also proves effective with visual feedback.

## 2. FRONTALK

### 2.1. Task Formulation

**Data composition.** Each data point in our dataset is a multi-turn dialogue, in which a user iteratively refines a website over multiple turns. It comprises a sequence of user intents $\{\mathbf{i}_t\}_{t=1}^{T}$, where $\mathbf{i}_t$ denotes the **underlying user goal** at turn $t$. To enable fine-grained evaluation, we pair each intent with a set of test cases $\mathcal{C}(\mathbf{i}_t) = \{c_m\}_{m=1}^{M}$ that decompose the intent into a collection of atomic, verifiable statements. For example, an intent *"implement visual badges next to thread titles*$_{(1)}$ *to indicate their status with distinct colors and shapes*$_{(2)}$ is decomposed into two test cases: (1) a badge is displayed next to each thread title, and (2) badges for different statuses are visually distinct. Without loss of generality, we fix the number of turns to $T = 10$ in this work.

**Algorithm 1** Inference pipeline.

**Input:** Model $\mathcal{M}$; user intents $\{\mathbf{i}_t\}_{t=1}^{T}$; user simulator SIMUSER
$\mathbf{H}_1 \leftarrow [\,]$
**for** $t \leftarrow 1$ **to** $T$ **do**
  $\tilde{\mathbf{i}}_t \leftarrow \text{SIMUSER}(\mathbf{i}_t, \mathbf{H}_t)$
  $o_t \leftarrow \mathcal{M}(\tilde{\mathbf{i}}_t, \mathbf{H}_t)$
  $\mathbf{H}_{t+1} \leftarrow \mathbf{H}_t \parallel \langle \tilde{\mathbf{i}}_t, o_t \rangle$
**end for**
$w = \text{RENDER}(o_T)$
**Return** $w$

**Inference pipeline.** During inference, we do not directly use the user intent $\mathbf{i}_t$ as model input, as it is static and may not reflect the evolving conversational context. Instead, we employ a user simulator SIMUSER$(\cdot)$ to transform each intent $\mathbf{i}_t$ into a context-aware instruction $\tilde{\mathbf{i}}_t = \text{SIMUSER}(\mathbf{i}_t, \mathbf{H}_t)$, detailed in §2.3. The instruction $\tilde{\mathbf{i}}_t$ then serves as the actual model input, as shown in Alg. 1. After the final turn, we render the model output $o_T$ into the final website $w$ for evaluation. For analytical purposes, we also compare against a **single-turn baseline** that concatenates the raw textual user intents into a single input, denoted as $o = \mathcal{M}([\mathbf{i}_0, \ldots, \mathbf{i}_T])$.

**Evaluation.** To comprehensively evaluate the quality of the generated websites, we assess two key dimensions: *instruction following* and *user experience*.

We first measure how well the model satisfies the given instructions. Using the curated set of test cases $\mathcal{C}(\mathbf{i})$, we define the **pass rate** (PR) metric by first computing the pass count (PC) – the number of test cases satisfied for a given intent $\mathbf{i}$ – and then aggregating across all intents for the output $o$:

$$\text{PC}(o|\mathbf{i}) = \sum_{c \in \mathcal{C}(\mathbf{i})} \mathbf{I}\left(\text{RENDER}(o) \text{ passes } c\right); \quad (1)$$

$$\text{PR}(o) = \frac{\sum_{\mathbf{i}} \text{PC}(o|\mathbf{i})}{|\sum_{\mathbf{i}} \mathcal{C}(\mathbf{i})|} \quad (2)$$

In practice, the pass count is measured by a web agent that interacts with the website and verifies each test case, as in §2.4. In our experiments, we observe that features implemented in earlier turns are often overwritten in later turns. Thus, we introduce an **forgetting rate** (FR) metric to measure how much functionality correctly implemented in prior turns ($t < T$) is lost in the final output:

$$\text{FR} = 1 - \frac{\sum_{t=1}^{T-1} \text{PC}(o_T|\mathbf{i}_t)}{\sum_{t=1}^{T-1} \text{PC}(o_t|\mathbf{i}_t)} \quad (3)$$

While these metrics assess implementation correctness when following user instructions, they may not fully capture the overall user experience. To address this, we also evaluate **usability**, defined as how easily a new user can learn to navigate the website and accomplish tasks. To this end, we deploy an LLM agent that simulates a first-time user with no prior knowledge of the website. The agent autonomously explores the interface, formulates tasks based on its observations, and attempts to complete them, yielding a behavioral trajectory. We then employ a *pairwise comparison protocol*, where an LLM judge compares the trajectories collected from two websites and determines which provides a superior user experience. The final metric is the win rate of model-generated websites against a set of reference websites, ensuring a robust and standardized evaluation.

### 2.2. Data Curation

We construct the FRONTALK dataset by collecting diverse real-world websites and automatically generating data grounded in these websites, followed by rigorous human refinement and validation to ensure quality.

**Source website selection.** To curate a diverse and challenging set of user intents reflective of real-world scenarios, we begin by sampling 10,000 webpages from the C4 dataset (Raffel et al., 2020). For each site, we generate summaries of its content and key features using GPT-4o. We then apply BERTopic (Grootendorst, 2022) to these summaries to identify thematic clusters. From the most prominent clusters,

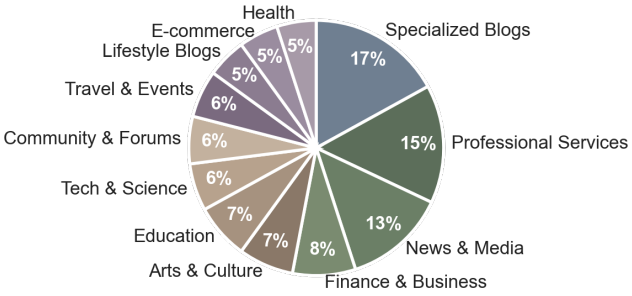



*Figure 2.* Website summary topics distribution.

we manually select 100 websites – one representative site per cluster – while excluding low-information pages (e.g., placeholders). This process yields a realistic and diverse set of seed websites spanning a wide range of domains, as summarized in Figure 2.

**Automatic data generation.** Using the curated website summaries, we prompt GPT-4o to generate user intents $\mathbf{i}_t$ and their paired test cases $\mathcal{C}(\mathbf{i}_t)$ for each conversational turn. These intents guide the subsequent user simulation (details in §2.3). We categorize user intents into two types: *functionality* (e.g., navigation, form submission, search) and *design* (e.g., layout, colors, images, visual effects). For each turn, we randomly select an intent type and prompt GPT-4o to generate a corresponding user intent along with a set of verifiable test cases. Figure 6 shows the distribution of words in the generated intents for each type, illustrating their diversity.

**Manual refinement and validation.** To ensure the reliability of our evaluation, we manually refined all automatically generated test cases to ensure that they are hallucination-free, unambiguous, and verifiable. We removed entirely unsatisfactory test cases and edited those that were partially satisfactory. This process led to the removal of 66 test cases (1.7%) and the editing of 2,783 test cases (73.4%). To validate the improvement, a secondary independent annotator reviewed a sample of 224 test cases both before and after editing. The manual refinement increased the validity rate of test cases from 90.1% to **94.7%**. Statistics of the final dataset are in Table 1.

### 2.3. User Simulation

As described in §2.1, the user simulator rewrites each static user intent into a context-aware instruction that matches the current state of the website. It preserves the core semantics while grounding the instruction in the dynamically evolving website state during the multi-turn conversation. Concretely, user simulators perform two critical functions: (1) **Resolving ambiguity**: When an intent refers to an element that is ambiguous or absent on the current website, the simulator refines the reference (e.g., "the submit button" → "the submit button on the right") or requests the creation of the missing element. (2) **Omitting redundancy**: If an intent requests a feature that has already been implemented, the simulator adapts the instruction to avoid redundant requests (e.g., "implement a navigation bar" → "modify the existing navigation bar").

*Table 1.* Statistics of curated user intents and test cases in FRONTALK.

| | |
|---|---|
| *Dataset Size* | |
| # Dialogues | 100 |
| # Turns | 1,000 |
| # Test Cases | 3,676 |
| *User Intent Length* | |
| # Words per Turn | 93.7 |
| # Words per Dialogue | 936.8 |

Following these principles, we implement two types of user simulators as in Figure 3:

1. **Textual instruction simulator**: An LLM is provided with the current code state and the original user intent, and is instructed to rewrite the intent into a textual user instruction.

2. **Visual instruction simulator**: A VLM is equipped with *drawing tools* and provided with screenshots of the current website. It produces sketches, shapes, and text annotations to anchor instructions directly onto the visual interface. This simulator is permitted multiple turns of drawing to fully articulate the user intent.

Implementation details and simulation examples are provided in Appendix B.3, B.4. We further conduct human evaluation on 114 simulated instructions, both visual and textual, and their associated 411 test cases. As reported in Appendix C.4, textual instruction simulation is highly reliable, achieving near-perfect plausibility, faithfulness to model-generated websites, and fidelity to pre-curated user intents. Visual simulation shows slightly lower reliability due to GPT-4o's limited tool-calling capabilities, as shown in Figure 16. We note that this is less a limitation than a strength of our approach: as more capable models become available, the reliability of visual user simulation will improve accordingly within our framework.

### 2.4. Agent-Based Evaluation

Traditional evaluation approaches in front-end development largely rely on static artifacts such as rendered screenshots or source code (Si et al.; Sun et al., 2025). However, both approaches fall short in dynamic test cases, such as "*selecting an option from a dropdown menu dynamically updates the page*". Screenshot-based metrics capture only a single visual state and cannot verify such interactions. Code-based analysis, while theoretically comprehensive, may overlook subtle implementation errors that only become apparent

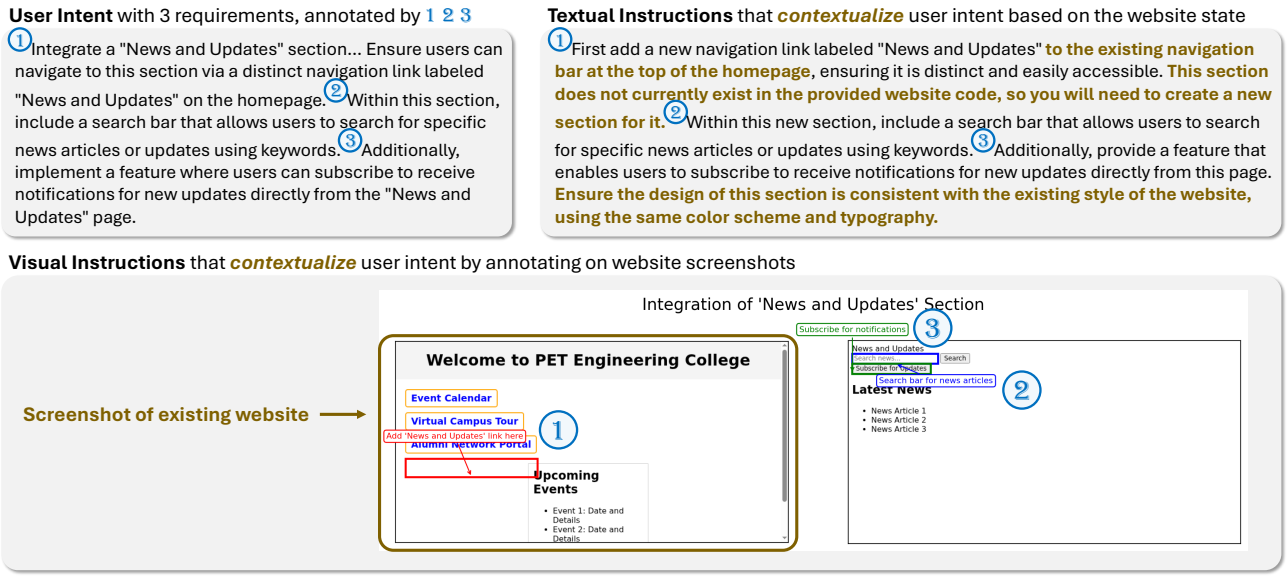



*Figure 3.* **Illustration of the user simulators**. The textual simulator rewrites the user intent based on the current code state, while the visual simulator expresses the intent by drawing sketches or annotating screenshots of the website.

during real user interaction.

To overcome these shortcomings and better emulate authentic user assessment, we employ an **interactive web agent** that evaluates the website through direct interaction. Our agent-based evaluator operates in two modes for the two different metrics: for *pass rate*, it acts as an informed expert, using the full context to verify a test case, while for *usability*, it simulates a naive user, exploring the site and attempting self-proposed tasks. We adapt the agent framework from WebVoyager (He et al., 2024) but introduce a novel suite of *image manipulation tools* to enhance the agent's visual perception, as detailed in Appendix B.5. By enabling the VLM to perform operations like cropping UI elements and comparing visual states, these tools improve alignment with human judgments, particularly on design-centric tasks involving fine-grained visual details, such as verifying a subtle hover effect in a local region.

## 3. Experimental Results on FRONTALK

We evaluate 20 models from nine model families, spanning proprietary and open-source LLMs and VLMs, on our proposed FRONTALK benchmark. Our analysis is organized around three questions: (1) How far are current models from solving FRONTALK? (2) What unique challenges arise in the multi-turn setting? (3) How does multi-modal user feedback affect model performance? The main results are shown in Table 2, with implementation details in Appendix B.

**Main results.** As shown in Table 2, **all models perform substantially below perfect accuracy** on our challenging FRONTALK dataset. Even the state-of-the-art model, Gemini-2.5-Pro, falls short of perfect performance by 25.0% with textual feedback and 31.3% with visual feedback. The gap is more significant for open-source models, where the top models fall short by 37.5% with textual instructions and 55.4% with visual instructions.

**Forgetting issue.** We measure the forgetting issue with the forgetting rate (FR) metric. Table 2 reveals a significant forgetting issue across all models, with their forgetting rate ranging from 4.3% to 44.6%. This issue typically arises when instructions across multiple turns request modifications to the same component of code (e.g., adding different buttons to the same section). Although these instructions are not mutually exclusive, models often overwrite the previous implementation rather than integrating the new feature, as illustrated in Figure 17. Importantly, this issue differs from the commonly discussed *long-context degradation* (Hong et al., 2025). As discussed in Appendix C.1, long-context degradation only affects certain models such as Claude-4-Sonnet, while others, including GPT-4o and Gemini-2.5-Pro, remain relatively robust to long contexts. In contrast, the forgetting issue consistently occurs across all models. Mitigating this issue requires models to dynamically navigate the codebase to preserve existing functionalities while implementing new ones – a capability that proves challenging for current models.

**Visual interpretation gap.** Comparing VLM performance under visual and textual instructions in Table 2, we observe a significant performance drop with visual feedback, though two modalities are designed to convey the same set of user intents. This degradation is particularly pronounced for open-source VLMs, resulting in **a larger gap between closed-source and open-source models** with visual feedback. To investigate the causes of these failures, we

*Table 2.* **Main evaluation results.** We report three metrics, pass rate (PR↑), usability (UX↑), and forgetting rate (FR↓). The best performance in each block is **bolded**. We report the single-turn performance without user simulation, and the multi-turn performance with textual (T) or visual (V) feedback. For visual feedback, we additionally report the difference (diff) relative to the textual setting.

| | | Single-Turn | | Multi-Turn (T) | | | Multi-Turn (V) | | |
|---|---|---|---|---|---|---|---|---|---|
| | | PR↑ | UX↑ | PR↑ | UX↑ | FR↓ | PR↑ (diff) | UX↑ (diff) | FR↓ |
| *Proprietary VLMs* | | | | | | | | | |
| GPT-4o | - | 51.4 | 50.0 | 56.0 | 55.0 | 21.4 | 55.0 (-1.0) | 52.5 (-2.5) | 8.0 |
| Claude-4-Sonnet | - | **77.5** | **71.8** | 45.5 | 60.8 | 22.8 | 59.3 (+13.8) | 64.8 (+4.0) | 10.6 |
| Gemini-2.5-Pro | - | 57.2 | 56.8 | **75.0** | **71.0** | **4.5** | **68.7** (-6.3) | **73.8** (+2.8) | **6.2** |
| *Open-Source LLMs* | | | | | | | | | |
| Qwen3 | 8B | 54.3 | 45.8 | 59.9 | 57.8 | 13.9 | n/a | n/a | n/a |
| GPT-OSS | 20B | 64.3 | 68.5 | 61.6 | **71.5** | 15.8 | n/a | n/a | n/a |
| Qwen3-Coder | 30B | 54.1 | 49.8 | 61.0 | 54.3 | **8.6** | n/a | n/a | n/a |
| Llama-3.3 | 70B | 38.4 | 35.8 | 28.9 | 40.8 | 46.3 | n/a | n/a | n/a |
| GPT-OSS | 120B | **68.0** | **73.5** | 57.9 | 66.8 | 9.2 | n/a | n/a | n/a |
| Qwen3 | 235B | 62.9 | 57.8 | 59.0 | 61.3 | 20.6 | n/a | n/a | n/a |
| Qwen3-Coder | 480B | 67.0 | 64.0 | **62.5** | 62.3 | 15.4 | n/a | n/a | n/a |
| DeepSeek-R1 | 685B | 60.0 | 53.5 | 24.7 | 40.0 | 34.5 | n/a | n/a | n/a |
| *Open-Source VLMs* | | | | | | | | | |
| Qwen2.5-VL | 7B | 24.5 | 19.5 | 23.7 | 26.8 | 28.2 | 16.1 (-7.6) | 17.0 (-9.8) | 39.3 |
| GLM-4.1V-Thinking | 9B | 36.6 | 32.3 | 18.2 | 41.5 | 5.8 | 15.3 (-2.9) | 32.0 (-9.5) | **4.3** |
| Ovis2.5 | 9B | 51.5 | 49.3 | 35.3 | 34.3 | 12.2 | 27.0 (-8.3) | 39.5 (+5.2) | 10.5 |
| Gemma3 | 12B | 47.1 | 43.5 | 43.8 | 43.0 | 30.3 | 30.4 (-13.4) | 38.5 (-4.5) | 22.1 |
| Gemma3 | 27B | 36.7 | 37.2 | 37.2 | 39.5 | 42.1 | 35.5 (-1.2) | **41.0** (+2.5) | 16.7 |
| Qwen3-VL | 30B | **54.6** | **50.0** | **54.1** | **50.3** | **4.7** | 39.1 (-15.0) | 35.8 (-14.5) | 12.3 |
| Qwen2.5-VL | 32B | 44.5 | 38.0 | 33.2 | 37.5 | 46.9 | 36.9 (+3.7) | 36.5 (-1.0) | 25.7 |
| Qwen2.5-VL | 72B | 43.0 | 33.8 | 52.8 | 46.5 | 17.8 | **44.6** (-8.2) | 37.3 (-9.2) | 15.4 |
| GLM-4.5V | 108B | 24.4 | 23.5 | 35.2 | 40.5 | 9.8 | 7.9 (-27.3) | 7.8 (-32.7) | 27.7 |

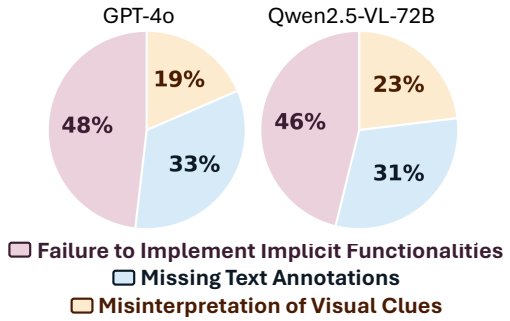



*Figure 4.* Error breakdown in interpreting visual instructions.

manually analyze 608 test cases where VLMs fail the visual instruction but succeed on its textual equivalent. Examining the outputs from two representative models, GPT-4o and Qwen2.5-VL-72B, we categorizes the errors into three primary types, as shown in Figure 4: (1) **Failure to implement implicit functionalities**, the most common error, where VLMs replicate the UI layout but fail to implement the underlying functionalities (e.g., failing to implement the page-update behavior for navigation arrows, as in Figure 18); (2) **Missing text annotations**, where models miss some text annotations in the image, especially in densely annotated images (Figure 19); and (3) **Misinterpretation of visual clues**, the least common error, which occurs when visual clues are slightly ambiguous (Figure 20).

**Comparison against the single-turn baseline.** As discussed in §2.1, the single-turn baseline concatenates all user intents into a single prompt, removing the complexity of multi-turn interaction. This is an analytical reference rather than a realistic setup because, in practice, user requests naturally emerge as the conversation evolves and cannot be reliably drafted up front. Comparing multi-turn and single-turn performance reveals a mixed and model-specific picture: some models (e.g., GPT-4o) perform better in multi-turn settings, while some others (e.g., Qwen3-Coder-480B) achieve stronger results in single-turn settings. This relates to a broader research question: Is it easier for models to process all requirements at once, or incrementally with multiple reasoning steps? We observe different failure patterns in each paradigm: the single-turn setting eliminates forgetting and long-context degradation, while the multi-turn setting offers compensating advantages through step-by-step task decomposition and reduced per-step complexity.

**Results with ambiguous users.** Prior sections assume that users express their intents through detailed, explicit instructions. In practice, however, users often provide indirect and ambiguous instructions – either because they have not

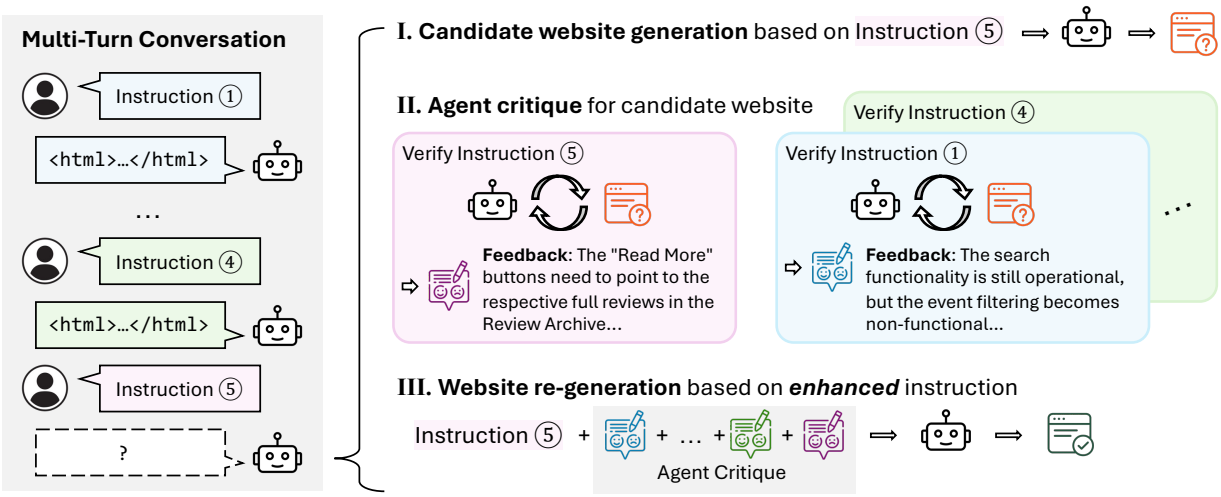



*Figure 5.* **Overview of AceCoder.** AceCoder first generates a candidate website, collects agent-based critique by interacting with the website and verifying the implementation of ***all prior instructions***, and then use the critique to enhance user instructions.

*Table 3.* Performance of GPT-4o with users of different interaction types.

| Turn | User | PR↑ | UX↑ |
|---|---|---|---|
| Single | - | 51.4 | 50.0 |
| Multi | Detailed (Standard) | **56.0** | **55.0** |
| Multi | Clarification-Only | 23.9 | 31.0 |
| Multi | Preference-Only | 16.4 | 21.0 |

yet fully clarified their own goals, or because they find it difficult to articulate them without guidance. Consequently, LLMs must proactively elicit or infer the user intents before executing tasks. To evaluate model performance in such ambiguous interactions, we simulate two user types: (1) **clarification-only user** that responds to direct questions but provides no unsolicited details, and (2) **preference-only user** that verbalizes only minimal requirements but instead reveals intent through choices among model-generated options. These simulated users have full access to the true intent but are constrained to communicate in the specified style. All settings share the same interaction budget ($N$ turns) to ensure fairness. As shown in Table 3, both user types significantly degrade performance compared to standard multi-turn or single-turn settings. This highlights models limited capability to actively elicit and infer user intent, directly affecting the task completion pass rate and the usability of generated websites.

**Human analysis for evaluation metrics.** To validate the reliability of our automatic evaluation, we recruit five annotators with undergraduate-level computer science training to assess model outputs. Agreement between LLM evaluation and human judgments yields an accuracy of 80.2 (Cohen's $\kappa$ of 58.9) for pass rate and 69.3 (Cohen's $\kappa$ of 56.3) for usability, indicating moderate to substantial agreement. A critical challenge lies in assessing fine-grained visual details, which our proposed image manipulation tools help mitigate, as shown in Figure 13. The slightly lower agreement for usability reflects its subjective nature. Further analysis reveals that the determinants of usability shift with model proficiency: for weaker models, usability is mostly affected by broken functionality, while for stronger models, usability hinges on nuanced design choices. Additional quantitative analysis is shown in Appendix C.3.

## 4. AceCoder: An Improved Baseline

As shown in §3, even state-of-the-art models suffer from the forgetting issue. To address this unique challenge arising from the multi-turn setting, we propose AceCoder as a stronger baseline. AceCoder builds on a simple insight: forgetting can be detected and thus prevented by re-verifying the built website against all prior instructions. Concretely, as shown in Figure 5 and Alg. 2, after the model generates an initial output, AceCoder employs a web agent to interact with the rendered website and verify whether both current and past instructions are correctly implemented. The resulting critiques are used to augment the original instructions to *regenerate* an improved website. Implementation details are in Appendix B.6.

As shown in Table 4, **AceCoder consistently improves performance across all evaluated models for both textual and visual instructions**. The gains are especially significant for textual instructions, reducing the forgetting rate to nearly zero. AceCoder also brings benefits to visual instructions, though the gains are more modest as the critique agent itself is limited by the challenge of interpreting visual instructions.

Table 5 further shows that **AceCoder outperforms all baselines**. Notably, the Replay baseline that replays all

*Table 4.* **Main results of AceCoder** with both textual (T) and visual (V) instructions.

| | Multi-Turn (T) | | | Multi-Turn (V) | | |
|---|---|---|---|---|---|---|
| | PR↑ | FR↓ | UX↑ | PR↑ | FR↓ | UX↑ |
| GPT-4o | 56.0 | 21.4 | 55.0 | 55.0 | 8.0 | 52.5 |
| +AceCoder | **65.3** | **0.4** | **76.3** | **60.2** | **7.2** | **65.8** |
| | +9.3 | -21.0 | +21.3 | +5.2 | -0.8 | +13.3 |
| Qwen2.5-VL-7B | 23.7 | 28.2 | 26.8 | 16.1 | 39.3 | 17.0 |
| +AceCoder | **25.4** | **-1.2** | **28.3** | **19.3** | **29.0** | **23.0** |
| | +1.7 | -29.4 | +1.5 | +3.2 | -10.3 | +6.0 |
| Qwen2.5-VL-72B | 52.8 | 17.8 | 46.5 | 44.6 | **15.4** | 37.3 |
| +AceCoder | **58.7** | **1.5** | **51.8** | **45.4** | 17.0 | **49.0** |
| | +5.9 | -16.3 | +5.3 | +0.8 | -1.6 | +11.7 |

*Table 5.* **Ablation study of AceCoder**, evaluated on GPT-4o with textual instructions.

| | PR↑ | FR↓ | UX↑ |
|---|---|---|---|
| *Baselines* | | | |
| Single-Turn | 51.4 | - | 50.0 |
| Multi-Turn | 56.0 | 21.4 | 55.0 |
| Replay | 62.3 | 6.3 | 63.8 |
| *Ablations* | | | |
| **AceCoder** | **65.3** | 0.4 | **76.3** |
| - Critique Past Turns | 61.2 | 16.1 | 70.0 |
| - Agent Critique | 58.8 | **-0.8** | 66.3 |
| + Reflection | 63.4 | -0.2 | 76.0 |

prior instructions has proven effective for multi-turn chatting (Laban et al., 2025). However, it yields smaller benefits in FronTalk, as mere replay can fail to reveal subtle implementation errors or complex code conflicts. We further conduct three ablations: *(1) -Critique Past Turns*: A variant that only critiques the features from the current turn improves the pass rate but fails to meaningfully reduce the forgetting rate. This confirms that **proactively verifying past features is critical to mitigating forgetting**. *(2) -Agent Critique*: A variant that replaces the agent critique with critique produced by a static LLM with access to the full conversation history, but not the built website. This results in a substantial performance degradation, highlighting **the importance of interaction-based agent critique**. *(3) +Reflection*: A variant that uses agent critiques to perform *reflection* rather than *regenerate* the entire website. Although this variant also performs well, it is slightly inferior to our full regeneration-based approach, suggesting that complete regeneration offers greater flexibility for incorporating agent feedback.

## 5. Related Work

**Front-end development.** Early research in front-end development focused on generating static webpages from images or textual descriptions (Gui et al., 2024; Laurençon et al., 2024; Si et al.; Xiao et al., 2025; Lin et al., 2025). Recent work has begun to address interactivity by generating individual interactive components (Xiao et al., 2024) or entire navigable sites (Zhu et al., 2025; Lu et al., 2025b). However, they have predominantly considered a single-turn setting, while our work extends them by introducing a conversational setting. The closest prior work is Li et al. (2025), which studies multi-turn code generation with visual inputs. However, it uses visual input only in the first turn and focuses on low-fidelity, low-information sketches. Another related effort is the concurrent work of Lu et al. (2025a), which employs multi-turn front-end coding as a *method* to improve performance on a single-turn benchmark rather than formalizing it as a distinct task.

**Multi-turn code generation.** Recent work in code generation has started to move beyond traditional single-turn benchmarks (Austin et al., 2021; Chen et al., 2021) toward multi-turn settings. These work simulates diverse conversational dynamics, such as users providing iterative feedback to help model complete coding tasks across multiple turns (Wang et al.; Han et al.), or users progressively clarifying ambiguous instructions (Laban et al., 2025). A common limitation, however, is their exclusive focus on textual interaction, overlooking the unique dynamics of visual feedback in multi-turn interactions.

## 6. Conclusion

This paper introduces FronTalk, a novel benchmark for front-end development that models a multi-turn code generation process with multi-modal user feedback. Our benchmark employs LLM-based user simulators to dynamically adapt pre-defined intents into either textual or visual instructions. The resulting websites are evaluated based on both instruction-following correctness and overall user experience. Evaluation of 20 models reveals that current models are far from perfect and struggle with two primary challenges: (1) a significant forgetting issue, where previously implemented features are overwritten in multi-turn interactions, and (2) a persistent difficulty in interpreting visual instructions. To address the former, we propose AceCoder, a strong baseline method that mitigates forgetting by leveraging a web agent to critique the implementation against past and present instructions. This work provides a solid foundation and a challenging testbed for future research in building more capable and reliable code generation approaches.

## References

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. arXiv preprint arXiv:2108.07732, 2021.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O.,

Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

Grootendorst, M. Bertopic: Neural topic modeling with a class-based tf-idf procedure. arXiv preprint arXiv:2203.05794, 2022.

Gui, Y., Li, Z., Wan, Y., Shi, Y., Zhang, H., Su, Y., Dong, S., Zhou, X., and Jiang, W. Vision2ui: A real-world dataset with layout for code generation from ui designs. CoRR, 2024.

Han, H., Samdani, R., He, Y., et al. Convcodeworld: Benchmarking conversational code generation in reproducible feedback environments. In The Thirteenth International Conference on Learning Representations.

He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., and Yu, D. Webvoyager: Building an end-to-end web agent with large multimodal models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 6864–6890, 2024.

Hong, K., Troynikov, A., and Huber, J. Context rot: How increasing input tokens impacts llm performance. URL https://research. trychroma. com/context-rot, retrieved October, 20:2025, 2025.

Karpathy, A. Discussion about "vibe coding". https://x.com/karpathy/status/1886192184808149383, February 2025. Post on X (formerly Twitter).

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

Laban, P., Hayashi, H., Zhou, Y., and Neville, J. Llms get lost in multi-turn conversation. arXiv preprint arXiv:2505.06120, 2025.

Laurençon, H., Tronchon, L., and Sanh, V. Unlocking the conversion of web screenshots into html code with the websight dataset. arXiv preprint arXiv:2403.09029, 2024.

Li, R., Zhang, Y., and Yang, D. Sketch2Code: Evaluating vision-language models for interactive web design prototyping. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pp. 3921–3955, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.198. URL https://aclanthology.org/2025.naacl-long.198/.

Lin, Z., Zhou, Z., Zhao, Z., Wan, T., Ma, Y., Gao, J., and Li, X. Webuibench: A comprehensive benchmark for evaluating multimodal large language models in webui-to-code. arXiv preprint arXiv:2506.07818, 2025.

Lu, Z., Ren, H., Yang, Y., Wang, K., Zong, Z., Pan, J., Zhan, M., and Li, H. Webgen-agent: Enhancing interactive website generation with multi-level feedback and step-level reinforcement learning. arXiv preprint arXiv:2509.22644, 2025a.

Lu, Z., Yang, Y., Ren, H., Hou, H., Xiao, H., Wang, K., Shi, W., Zhou, A., Zhan, M., and Li, H. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch. arXiv preprint arXiv:2505.03733, 2025b.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research, 21 (140):1–67, 2020.

Si, C., Zhang, Y., Li, R., Yang, Z., Liu, R., and Yang, D. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, volume 1, pp. 3956–3974.

Sun, H., Wang, H. W., Gu, J., Li, L., and Cheng, Y. Fullfront: Benchmarking mllms across the full front-end engineering workflow. arXiv preprint arXiv:2505.17399, 2025.

Wang, X., Wang, Z., Liu, J., Chen, Y., Yuan, L., Peng, H., and Ji, H. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In The Twelfth International Conference on Learning Representations.

Xiao, J., Wan, Y., Huo, Y., Wang, Z., Xu, X., Wang, W., Xu, Z., Wang, Y., and Lyu, M. R. Interaction2code: Benchmarking mllm-based interactive webpage code generation from interactive prototyping. arXiv preprint arXiv:2411.03292, 2024.

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

Zhu, H., Zhang, Y., Zhao, B., Ding, J., Liu, S., Liu, T., Wang, D., Liu, Y., and Li, Z. Frontendbench: A benchmark for evaluating llms on front-end development via automatic evaluation. arXiv preprint arXiv:2506.13832, 2025.

## A. Dataset Details

Wordcloud plots demonstrating distribution of user intents are in Figure 6.

*Figure 6.* Distributions of user intents for functionality and design types respectively.

A data example is as follows:

- **Website goal:** The website "Shifting the Balance" appears to be a personal or family blog that shares thoughts and reflections on life, family, and societal issues.
- **Intent (t=1):** `[type=functionality]` Build a website for "Shifting the Balance" that serves as a personal blog sharing thoughts on life, family, and societal issues. Implement a comment section accessible at the bottom of each blog post page, allowing users to leave comments and replies. Include an email subscription feature on the homepage and each blog post page, enabling visitors to subscribe with their email addresses to receive notifications about new posts. Provide social media sharing options for Twitter, Facebook, and LinkedIn on each blog post page to facilitate easy sharing of content.
  - ***Test cases:***
  - Navigate to the bottom of any blog post page to find the comment section. `[pass]` Users can see a text input field to write comment and an option to submit it. `[fail]` Users cannot write comment, or the comment cannot be submitted.
  - On the homepage and any blog post page, locate the email subscription feature. `[pass]` Users can see an input field to input email address, and an option to confirm subscription. `[fail]` The input field for email address is missing, or the email address cannot be submitted.
  - On any blog post page, find the social media sharing options. `[pass]` Users can see buttons or links for sharing on Twitter, Facebook, and LinkedIn. `[fail]` One or more of the social media sharing options is missing.
- **Intent (t=2):** `[type=functionality]` Enhance the comment section by adding a "Like" feature to each comment and reply, allowing users to express appreciation for comments. Ensure that each comment and reply has a visible "Like" button next to it, which increments a visible like counter when clicked. Users should be able to navigate to the comment section by scrolling to the bottom of any blog post page from the homepage.
  - ***Test cases:***
  - Each comment and reply in the comment section of any blog post page has a visible "Like" button next to it. `[pass]` Every comment and reply displays a "Like" button adjacent to it. `[fail]` One or more comments or replies do not have a "Like" button next to them.
  - Clicking the "Like" button next to each comment or reply in the comment section at the bottom of any blog post page increases its like counter by one. `[pass]` The like counter for a comment or reply increases by one when its "Like" button is clicked. `[fail]` The like counter does not change, or changes incorrectly, when the "Like" button is clicked.
  - The like counter is displayed for each comment and reply in the comment section at the bottom of each blog post page. `[pass]` A like counter is visibly displayed for every comment and reply. `[fail]` The like counter is missing for one or more comments or replies.
- **Intent (t=3):** `[type=functionality]` Introduce a "Featured Posts" section on the homepage, showcasing a rotating selection of three highlighted blog posts to draw visitor attention. Ensure that users can navigate to the detailed blog post page by clicking on any featured post title or image. Include a brief excerpt and a placeholder image for each featured

post to provide a preview of the content. The rotation should automatically change the featured posts every 10 seconds, but users should also have the option to manually navigate through them using forward and backward arrows.

  - ***Test cases:***
  - The homepage displays a "Featured Posts" section with three distinct blog post previews. `[pass]` The homepage shows a section labeled "Featured Posts" with three previews of different blog posts. `[fail]` The "Featured Posts" section is missing, or it does not contain exactly three blog post previews.
  - Each featured post in the "Featured Posts" section on the homepage has a visible title, excerpt, and placeholder image. `[pass]` All three featured posts show a title, an excerpt, and a placeholder image. `[fail]` Any of the featured posts is missing a title, an excerpt, or a placeholder image.
  - Clicking on a featured post's title or image on the homepage navigates the user to the detailed blog post page. `[pass]` Clicking on the title or image of a featured post redirects the user to its detailed blog post page. `[fail]` Clicking on the title or image does not navigate to the detailed blog post page.
  - The featured posts on the homepage automatically rotate every 10 seconds. `[pass]` The set of featured posts changes automatically every 10 seconds. `[fail]` The featured posts do not change automatically, or the timing is incorrect.
  - Users can manually navigate through the featured posts on the homepage using forward and backward arrows. `[pass]` The user can click forward and backward arrows to manually navigate through the featured posts. `[fail]` The arrows are missing, or clicking them does not navigate through the featured posts.

- **Intent (t=4):** `[type=design]` Refine the visual design of the blog post pages by introducing a cohesive color scheme and typography style that enhances readability and aesthetic appeal. Use a serif font for blog titles and a sans-serif font for body text to create a clear hierarchy and visual contrast. Apply a soft background color to the entire page to reduce eye strain and highlight the content areas. Ensure that interactive elements such as social media sharing buttons, comment section, and "Like" buttons have a distinct hover effect to visually indicate interactivity.

  - ***Test cases:***
  - Blog titles on each blog post page are displayed in a serif font, while body text uses a sans-serif font. `[pass]` Blog titles appear in a serif font, and body text is in a sans-serif font. `[fail]` Both blog titles and body text appear in the same font style.
  - The background color of each blog post page is soft and distinct from the content areas. `[pass]` The page background is a soft color, differing from the content sections. `[fail]` The page background color is harsh or indistinct from content areas.
  - Interactive elements such as social media sharing buttons, the comment section, and "Like" buttons on each blog post page exhibit a distinct visual change on hover. `[pass]` Interactive elements change visually (e.g., color, size) when hovered over. `[fail]` Interactive elements show no visual change when hovered over.

- **Intent (t=5):** `[type=design]` Enhance the homepage by designing an inviting hero section at the top, featuring a large, captivating background image that represents the blog's themes. Overlay this image with a semi-transparent color layer to ensure text readability. Include a welcoming headline in a bold serif font, accompanied by a concise tagline in a smaller sans-serif font. Add a call-to-action button, styled with a distinct hover effect, encouraging visitors to explore the blog further.

  - ***Test cases:***
  - The hero section on the homepage features a large background image. `[pass]` A large background image is prominently displayed at the top of the homepage. `[fail]` There is no background image, or it is not prominently displayed.
  - The homepage's hero section should have a background image with a semi-transparent color overlay. `[pass]` The background image has a visible semi-transparent color layer over it. `[fail]` The background image does not have any overlay, or the overlay is not semi-transparent.
  - The headline is in a bold serif font, and the tagline is in a smaller sans-serif font. `[pass]` The headline uses a bold serif font, and the tagline uses a smaller sans-serif font. `[fail]` The headline and tagline do not follow the specified typography styles.
  - The call-to-action button on the homepage hero section is present with a distinct hover effect. `[pass]` A call-to-action button is visible and exhibits a distinct visual change when hovered over. `[fail]` There is no call-to-action button, or it lacks a hover effect.

- **Intent (t=6):** `[type=design]` Enhance the visual appeal of the "Featured Posts" section by adding smooth transition animations when rotating between posts, creating a more engaging user experience. Introduce a subtle zoom-in effect on

the placeholder images of each featured post when hovered over, emphasizing interactivity and drawing attention. Ensure the navigation arrows have a bold, contrasting color to make them easily identifiable, and implement a gentle bounce animation on hover to indicate their functionality.

  - ***Test cases:***
  - When the "Featured Posts" section on the homepage rotates between posts, there is a smooth transition animation observable. `[pass]` The transition between posts is smooth and visually fluid. `[fail]` The transition appears abrupt or jerky.
  - Hovering over a placeholder image in the "Featured Posts" section on the homepage results in a noticeable zoom-in effect. `[pass]` The image enlarges slightly when hovered over. `[fail]` The image remains static with no change on hover.
  - The navigation arrows in the "Featured Posts" section on the homepage are in a bold, contrasting color compared to the background. `[pass]` The arrows stand out distinctly against the background. `[fail]` The arrows blend into the background or are difficult to discern.
  - Hovering over the navigation arrows in the "Featured Posts" section on the homepage triggers a gentle bounce animation. `[pass]` The arrows exhibit a bounce effect when hovered over. `[fail]` The arrows show no movement or animation on hover.

- **Intent (t=7):** `[type=functionality]` Introduce a search functionality on the homepage, allowing users to easily find specific blog posts by entering keywords. Ensure the search bar is prominently positioned near the top of the homepage and includes a placeholder text prompting users to "Search blog posts...". Upon entering a search query and hitting enter, users should be directed to a results page displaying a list of blog posts that match the search terms. Each result should include the blog post title, a brief excerpt, and a link to the full post for easy navigation.

  - ***Test cases:***
  - Verify the presence of a search bar near the top of the homepage with placeholder text "Search blog posts...". `[pass]` The search bar is visible near the top of the homepage with the specified placeholder text. `[fail]` The search bar is not visible, or the placeholder text is incorrect or missing.
  - Test the search functionality on the homepage by entering a keyword in the search bar and pressing enter. `[pass]` The user is directed to a results page displaying a list of blog posts matching the keyword. `[fail]` The user is not directed to a results page, or no relevant posts are displayed despite matching content.
  - Check that each search result on the results page includes the blog post title, a brief excerpt, and a link to the full post. `[pass]` Each result displays the title, excerpt, and a clickable link to the full post. `[fail]` Any of the required elements (title, excerpt, link) are missing from the search results.

- **Intent (t=8):** `[type=functionality]` Introduce a "Related Posts" section at the end of each blog post page, featuring three blog posts with similar themes to engage readers further. Ensure that each related post displays a title, a brief excerpt, and a placeholder image, with links to navigate to the full blog post. Allow users to access this section by scrolling to the bottom of a blog post page, just above the comment section. Incorporate a feature that dynamically selects related posts based on shared tags or categories with the current post.

  - ***Test cases:***
  - Navigate to the bottom of each blog post page and verify the presence of a "Related Posts" section. `[pass]` The "Related Posts" section is visible just above the comment section. `[fail]` The "Related Posts" section is not visible or appears below the comment section.
  - Check that each related post in the "Related Posts" section at the end of each blog post page displays a title, a brief excerpt, and a placeholder image. `[pass]` Each related post shows a title, an excerpt, and an image. `[fail]` One or more related posts are missing a title, an excerpt, or an image.
  - For each post in the "Related Posts" section at the end of each blog post page, users can easily navigate to their respective full blog post. `[pass]` Users can navigate to the full blog post, such as by clicking the title, the image, or a clickable link. `[fail]` Clicking a related post title or image does not redirect to the corresponding full post page, and there are no visible navigation links or buttons.
  - Verify that the related posts on each blog post page are selected based on shared tags or categories with the current post. `[pass]` The related posts share at least one tag or category with the current post. `[fail]` None of the related posts share any tags or categories with the current post.

- **Intent (t=9):** `[type=design]` Enhance the "Related Posts" section by incorporating a visually appealing card layout that neatly organizes each related post with a subtle drop shadow and rounded corners. Use a consistent background color

across all cards to unify the section visually. Implement an interactive hover effect on each card, slightly elevating it to indicate interactivity and engagement. Ensure that the title of each related post is in a bold serif font for emphasis, while excerpts remain in a smaller sans-serif font for readability.

  - ***Test cases:***
  - Each related post is displayed within a card with rounded corners and a subtle drop shadow on the blog post page. `[pass]` Each related post appears within a bordered card with rounded corners and a shadow effect. `[fail]` Related posts lack rounded corners or a shadow effect, or do not appear within a card layout.
  - All cards in the "Related Posts" section on each blog post page have a consistent background color. `[pass]` The background color is uniform across all related post cards. `[fail]` The background color varies between related post cards.
  - A hover effect elevates each card in the "Related Posts" section at the end of each blog post page slightly when interacted with. `[pass]` Hovering over a related post card causes it to slightly elevate from the rest. `[fail]` Hovering over a related post card does not change its position or appearance.
  - The title of each related post in the "Related Posts" section at the end of each blog post page is in a bold serif font, while excerpts are in a smaller sans-serif font. `[pass]` Titles are distinctly bold and serif, and excerpts are visibly smaller and sans-serif. `[fail]` Titles and excerpts do not follow the specified font styling, lacking contrast in weight or style.

- **Intent (t=10):** `[type=design]` Enhance the visual design of the email subscription feature across the homepage and each blog post page by introducing a stylish subscription form. Use a rounded input field with a placeholder text that invites users to "Enter your email...". Add a vibrant call-to-action button next to the input field, styled with a hover effect that changes its background color. Ensure the form is positioned prominently within the layout, and add a subtle animation that draws attention to the form upon page load.

  - ***Test cases:***
  - The email subscription form on the homepage and each blog post page should feature a rounded input field with placeholder text "Enter your email...". `[pass]` The input field is rounded and displays the specified placeholder text. `[fail]` The input field is not rounded or does not display the specified placeholder text.
  - The call-to-action button next to the input field in the email subscription form should have a vibrant background color and change color on hover. `[pass]` The button has a vibrant background color and visibly changes color when hovered over. `[fail]` The button does not have a vibrant background color or does not change color on hover.
  - The email subscription form should be prominently positioned within the layout of the homepage and each blog post page. `[pass]` The form is easily noticeable and positioned in a prominent area of the page. `[fail]` The form is not easily noticeable or is positioned in a less prominent area of the page.
  - The email subscription form on the homepage and each blog post page should feature a subtle animation upon page load to draw attention. `[pass]` A subtle animation is visible when the page loads, drawing attention to the form. `[fail]` No animation is present upon page load, or the animation does not draw attention to the form.

## B. Implementation Details

### B.1. Hyperparameters

For LLM inference, we use the same configuration across every component, involving base inference on the benchmark, LLM-based evaluation, and the AceCoder pipeline. For open-source LLMs, we employ (Kwon et al., 2023) for model serving and make requests using the `openai` package; for proprietary LLMs, we directly call their official APIs.

We set the maximum output-token limit empirically for each model family to ensure that the model can complete its generation for more than 95% of input instructions. The limits are configured as follows:

- 10,000 tokens for GPT-4o, Llama-3.3, Qwen2.5-VL, GPT-OSS and Gemma3 models;
- 20,000 tokens for Gemini-2.5-Pro, Qwen3, Qwen3-Coder, DeepSeek-R1, GLM-4, Ovis2.5, and Qwen3-VL models;
- 30,000 tokens for Claude-4-Sonnet.

If the sum of input and output tokens exceeds the model's context window, we truncate the dialogue history by removing model responses from the earliest turns while always preserving the user instructions. Except for the maximum token limit, we do not modify any additional hyperparameters. Each LLM is run using its default configuration – typically optimized for each model. For open-source models, we use the settings specified in their `generation_config.json`; for proprietary

models, we use their default generation parameters. The `huggingface` identifiers for all open-source models are provided in Table 6.

For agent-based evaluation, we employ the web agent powered by GPT-4o. We report *pass rate* (PR) and *usability* (UX) as main metrics and *forgetting rate* (FR) as a supplementary metric to measure the forgetting issue. As discussed in the main content, the usability metric is reported as the win rate against a curated set of reference websites, where 50% represents tie. We use the single-turn outputs from GPT-4o as reference websites, and release it along with our evaluation code to allow standardized evaluation.

### B.2. Cost Analysis

To measure the evaluation pipeline cost and facilitate wider community adoption, we measured the cost for each major component of our pipeline using GPT-4o on the full dataset. As shown in Table 7, the evaluation components based on web agents account for the largest share of cost, followed by the visual user simulation that relies on a tool-use drawing agent.

We additionally quantify the additional computational overhead brought by our proposed ACECODER. We measure detailed breakdown of the inference cost using GPT-4o on a subset of 10 dialogues. As in Table 8, the majority of the overhead comes from agent-based critique.

**The analysis above largely reflect the broader efficiency limitations of current LLM agents**, including web agents and tool-using agents, which involve multi-turn interactions with the environment, high-complexity observation spaces, and long multi-modal histories. These challenges are well known in existing systems and are not specific to our implementation. Potential future directions for improving efficiency include more efficient agent designs, better history-compression mechanisms, and improved planning algorithms.

*Table 6.* Detailed `huggingface` ID for open-source models we evaluate.

| Model | Size | HF ID |
|---|---|---|
| Qwen3 | 8B | Qwen/Qwen3-8B |
| GPT-OSS | 20B | openai/gpt-oss-20b |
| Qwen3-Coder | 30B | Qwen/Qwen3-Coder-30B-A3B-Instruct |
| Llama-3.3 | 70B | meta-llama/Llama-3.3-70B-Instruct |
| GPT-OSS | 120B | openai/gpt-oss-120b |
| Qwen3 | 235B | Qwen/Qwen3-235B-A22B |
| Qwen3-Coder | 480B | Qwen/Qwen3-Coder-480B-A35B-Instruct-FP8 |
| DeepSeek-R1 | 685B | deepseek-ai/DeepSeek-R1-0528 |
| Qwen2.5-VL | 7B | Qwen/Qwen2.5-VL-7B-Instruct |
| GLM-4.1V-Thinking | 9B | zai-org/GLM-4.1V-9B-Thinking |
| Ovis2.5 | 9B | AIDC-AI/Ovis2.5-9B |
| Gemma3 | 12B | google/gemma-3-12b-it |
| Gemma3 | 27B | google/gemma-3-27b-it |
| Qwen3-VL | 30B | Qwen/Qwen3-VL-30B-A3B-Instruct |
| Qwen2.5-VL | 32B | Qwen/Qwen2.5-VL-32B-Instruct |
| Qwen2.5-VL | 72B | Qwen/Qwen2.5-VL-72B-Instruct |
| GLM-4.5V | 108B | zai-org/GLM-4.5V |

| | # LLM requests | # Input tokens | # Cached input tokens | # Output tokens | Est. cost |
|---|---|---|---|---|---|
| Inference (textual) | 1.0k | 147k | 7.2M | 2.0M | $29.43 |
| Inference (visual) | 1.0k | 1.2M | 9.9M | 1.3M | $27.95 |
| User simulation (textual) | 900 | 3.6M | 0 | 115k | $10.11 |
| User simulation (visual) | 2.0k | 16.1M | 12.7M | 1.0M | $66.38 |
| Evaluation of pass rate | 13.3k | 16.2M | 36.4M | 893k | $95.00 |
| Evaluation of usability | 1.6k | 2.2M | 3.8M | 258k | $12.75 |

*Table 7.* Cost breakdown for each component in the inference and evaluation pipeline.

| | # LLM requests | # Input tokens | # Cached input tokens | # Output tokens | Est. cost |
|---|---|---|---|---|---|
| Baseline inference | 100 | 14.4k | 728k | 205k | $3.00 |
| ACECODER | 4.5k | 3.1M | 15.1M | 967k | $36.22 |
| *Cost breakdown of* ACECODER *components* | | | | | |
| Initial website generation | 100 | 14.7k | 928k | 303k | $4.22 |
| Agent-based critique | 4.3k | 2.9M | 13.0M | 380k | $27.29 |
| Refined website generation | 90 | 148k | 1.2M | 284k | $4.7 |

*Table 8.* Cost analysis of ACECODER.

### B.3. Textual Instruction Simulator

As described in §2.3, the textual instruction simulator is an LLM prompted with the current code state and asked to rewrite the user intent to (1) resolve ambiguity and (2) remove redundancy. The prompts for rewriting functionality- and design-type user intents are provided in Listing 7 and Listing 8, respectively. We prompt GPT-4o to simulate textual instructions. Example outputs for the user intents in §A are in Table 9.

### B.4. Visual Instruction Simulator

For the visual instruction simulator, we equip it with drawing tools based on the `matplotlib` package. In the first turn, the model is provided with screenshots of every page on the website, as well as the coordinates of all visible elements. These coordinates can be used in later turns to guide annotations. At each turn, we provide a VLM (GPT-4o in this work) with the following tools:

1. Subplot organization: Tools to create and manage multiple subplots using `matplotlib`'s subplot functionality.
2. Layout visualization: Renders a layout sketch, represented by minimal HTML code, into an image. This process generates additional artifacts: the coordinates of every element on the new layout page. These coordinates are provided to the model in the next turn to assist with drawing.
3. Shape drawing: Tools for drawing various shapes using `matplotlib`.
4. Text annotation: A custom text annotation tool. We constrain the VLM to call our tool instead of `matplotlib`'s native text function. Our tool automatically organizes text annotations to minimize overlapping.

The VLM will generate python scripts to draw a visual instruction image using these tools. If the generated code has errors during execution, we allow models to re-generate for up to 5 times per turn. At the end of each tool-calling turn, the results, including the drawn images and other artifacts (e.g., coordinates), are fed back to the model. The model is then asked if the visualization is finished, i.e., if the visual instruction fully represents the user's intent. If not, it is allowed to perform another round of tool calling. We allow a maximum of three turns. The full prompt is in Listing 9.

Example outputs for the user intents from §A are shown in Figure 8-11. To demonstrate the multi-turn drawing process, we show the code snippets in two turns of instruction drawing in Listing 1 and Listing 2, with their corresponding output images in Figure 7 and 8.

*Listing 1.* **Code in the *first* step in the visual instruction simulation process.** The resulting image is in Figure 7. Our custom tools are highlighted in gold.

```
import matplotlib.pyplot as plt
from PIL import Image
import matplotlib.patches as patches

# Define the number of subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# First Subplot: Blog Post Page
axes[0].imshow(Image.open("screenshot_blog-post.png"))
scroll_shape = patches.Rectangle((8, 160.9), 809, 21, facecolor='none', edgecolor='#FF0000')
axes[0].add_patch(scroll_shape)
(*@\textcolor{darkgold}{text\_annotation}@*)(axes[0], "Scroll down to comment section", scroll_shape,
color='#FF0000')
axes[0].set_title("Blog Post Page")
```

*Table 9.* Example outputs for simulated textual instructions. Content in the instructions that are distinct from original user intents are highlighted in gold.

| t | User Intents | Simulated Instructions |
|---|---|---|
| 2 | Enhance the comment section by adding a "Like" feature to each comment and reply, allowing users to express appreciation for comments. Ensure that each comment and reply has a visible "Like" button next to it, which increments a visible like counter when clicked. Users should be able to navigate to the comment section by scrolling to the bottom of any blog post page from the homepage. | To enhance user engagement in the comment section of each blog post, add a "Like" feature to both comments and replies. Implement a "Like" button next to each comment and reply, which should visibly increase the like counter upon being clicked. Ensure this functionality is available within the existing comment section located at the bottom of each blog post page. Users can access the comment section by scrolling down to the end of any blog post, which they can navigate to from the homepage by selecting a blog post title under "Latest Blog Posts". |
| 4 | Refine the visual design of the blog post pages by introducing a cohesive color scheme and typography style that enhances readability and aesthetic appeal. Use a serif font for blog titles and a sans-serif font for body text to create a clear hierarchy and visual contrast. Apply a soft background color to the entire page to reduce eye strain and highlight the content areas. Ensure that interactive elements such as social media sharing buttons, comment section, and "Like" buttons have a distinct hover effect to visually indicate interactivity. | Refine the visual design of the blog post pages by implementing a cohesive color scheme and typography style to enhance readability and aesthetic appeal. Introduce a serif font for blog titles and a sans-serif font for body text to establish a clear hierarchy and visual contrast. Apply a soft background color to the entire page to reduce eye strain and highlight the content areas. Ensure that interactive elements, such as the social media sharing buttons within each blog post, the comment section, and the "Like" buttons adjacent to comments, have a distinct hover effect to visually indicate interactivity. Note that if the distinct hover effect is not currently visible, it must be implemented to enhance user experience. |
| 8 | Introduce a "Related Posts" section at the end of each blog post page, featuring three blog posts with similar themes to engage readers further. Ensure that each related post displays a title, a brief excerpt, and a placeholder image, with links to navigate to the full blog post. Allow users to access this section by scrolling to the bottom of a blog post page, just above the comment section. Incorporate a feature that dynamically selects related posts based on shared tags or categories with the current post. | To enhance user engagement on the blog post pages, introduce a "Related Posts" section directly above the existing comments section on each blog post page. This section should display three posts that share similar themes with the current post, determined by matching tags or categories. Each related post should include a title, a short excerpt, and a placeholder image, all of which should be clickable links that take users to the full blog post. This feature will help keep readers interested and encourage them to explore more content on the site. Ensure this section is seamlessly integrated so that users can easily find it by scrolling to the end of the blog post content. |
| 10 | Enhance the visual design of the email subscription feature across the homepage and each blog post page by introducing a stylish subscription form. Use a rounded input field with a placeholder text that invites users to "Enter your email...". Add a vibrant call-to-action button next to the input field, styled with a hover effect that changes its background color. Ensure the form is positioned prominently within the layout, and add a subtle animation that draws attention to the form upon page load. | Refine the email subscription feature on the homepage and each blog post page by enhancing the subscription form's design. Implement a rounded input field with the placeholder text "Enter your email..." and position a vibrant call-to-action button next to it. This button should have a hover effect that changes its background color to make it more engaging. Ensure the form is prominently placed within the layout to capture attention and add a subtle animation that highlights the form when the page loads. Since the current form is basic, these enhancements are not yet implemented and must be addressed. |

```
# Second Subplot: Comment Section with Like Feature
html_code_comments = """
<html lang="en"><head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Comments</title>
    <style>
        body { font-family: Arial, sans-serif; }
        .comment-section { margin-top: 20px; }
        .comment { margin-top: 10px; }
    </style>
</head>
<body>
    <h1>Comments</h1>

    <div class="comment-section">
        <textarea id="comment" placeholder="Add a comment..." rows="4" cols="50"></textarea><br>
        <button onclick="postComment()">Post Comment</button>

        <div id="comments-list" class="comments-list">
            <div class="comment">
                <p>Sample Comment</p>
                <button onclick="likeComment(this)">Like</button> <span>0 Likes</span>
                <button onclick="replyToComment(this)">Reply</button>
            </div>
        </div>
    </div>
</body></html>
"""
(*@\textcolor{darkgold}{layout\_visualization}@*)(html_code_comments, axes[1])
axes[1].set_title("Comment Section with Like Feature")

plt.tight_layout()
```

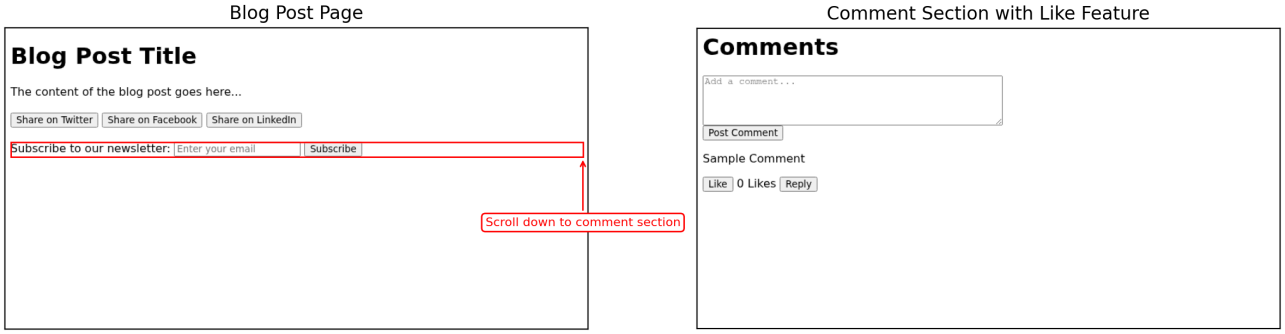



*Figure 7.* Intermediate drawing results after executing code in Listing 1.

*Listing 2.* **Code in the *second* step in the visual instruction simulation process.** The resulting image is in Figure 8. Our custom tools are highlighted in gold.

```
# Adding annotations for the "Like" feature in the comment section

# Coordinates for the "Like" button and "0 Likes" counter
like_button_coords = (8, 209.4, 42.6, 21)
like_counter_coords = (55.7, 209.4, 55.5, 19)

# Drawing a rectangle around the "Like" button
like_button_shape = patches.Rectangle(
    (like_button_coords[0], like_button_coords[1]),
    like_button_coords[2],
    like_button_coords[3],
    facecolor='none',
    edgecolor='#00FF00'
)
axes[1].add_patch(like_button_shape)
(*@\textcolor{darkgold}{text\_annotation}@*)(axes[1], "Add 'Like' button", like_button_shape, color='#00FF00')

# Drawing a rectangle around the "0 Likes" counter
like_counter_shape = patches.Rectangle(
    (like_counter_coords[0], like_counter_coords[1]),
    like_counter_coords[2],
```

```
    like_counter_coords[3],
    facecolor='none',
    edgecolor='#00FF00'
)
axes[1].add_patch(like_counter_shape)
(*@\textcolor{darkgold}{text\_annotation}@*)(axes[1], "Show like count", like_counter_shape, color='#00FF00')
```

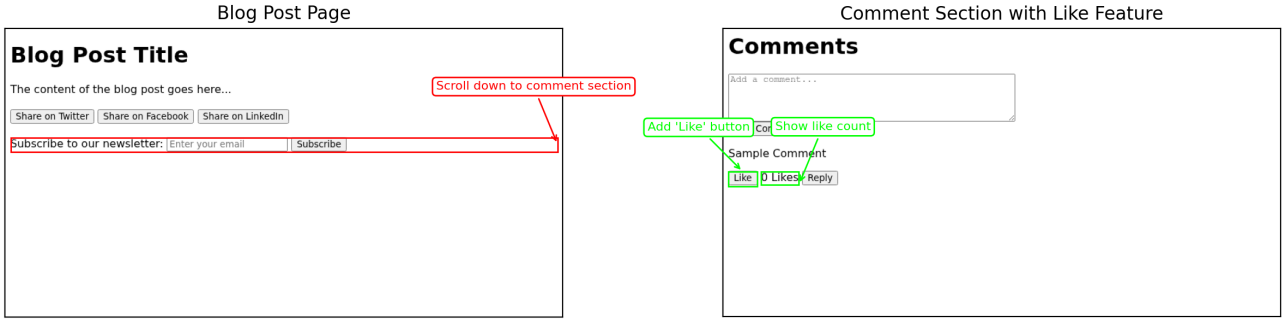



*Figure 8.* Simulated visual instructions for `t=2`. This is obtained after executing code in Listing 2 on top of Figure 7. User intent: Enhance the comment section by adding a "Like" feature to each comment and reply, allowing users to express appreciation for comments. Ensure that each comment and reply has a visible "Like" button next to it, which increments a visible like counter when clicked. Users should be able to navigate to the comment section by scrolling to the bottom of any blog post page from the homepage.

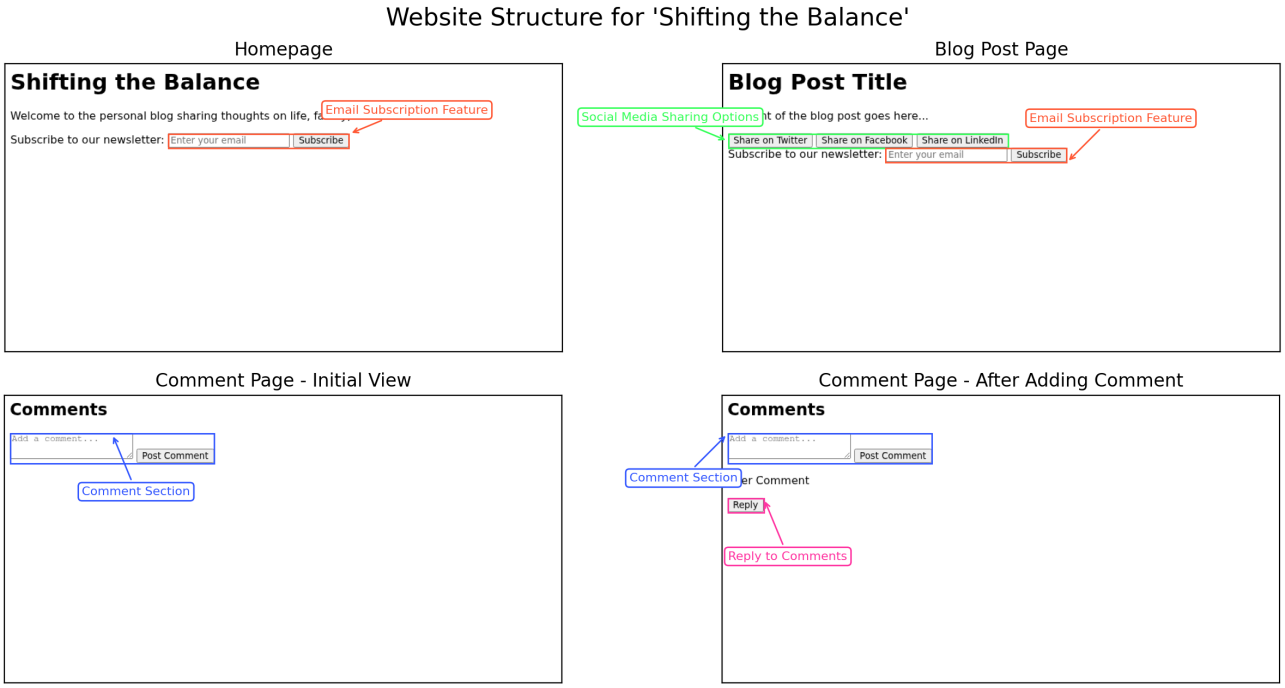



*Figure 9.* Simulated visual instructions for `t=1`. User intent: Build a website for "Shifting the Balance" that serves as a personal blog sharing thoughts on life, family, and societal issues. Implement a comment section accessible at the bottom of each blog post page, allowing users to leave comments and replies. Include an email subscription feature on the homepage and each blog post page, enabling visitors to subscribe with their email addresses to receive notifications about new posts. Provide social media sharing options for Twitter, Facebook, and LinkedIn on each blog post page to facilitate easy sharing of content.

### B.5. Agent-Based Evaluation

Following the WebVoyager framework (He et al., 2024), our evaluation agent interacts with a web environment through multiple steps. At each step, the agent receives two forms of input observations, as shown in Figure 12: (1) A screenshot of the web page, annotated with numerical labels on all interactive elements, and (2) A corresponding text representation

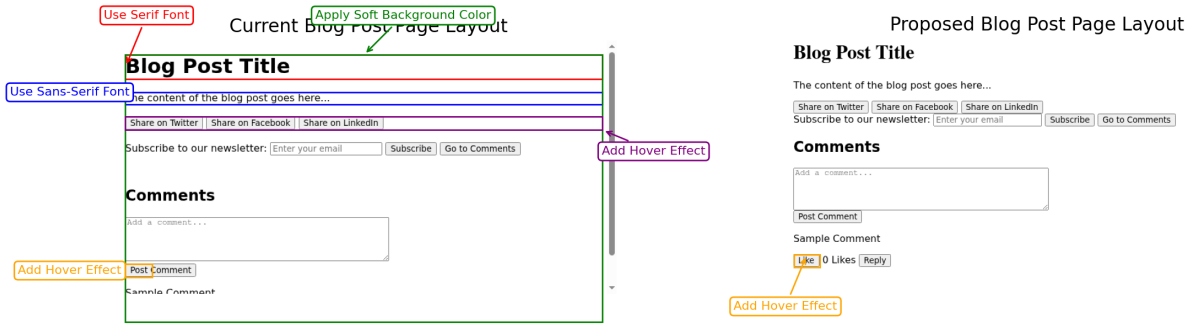



*Figure 10.* Simulated visual instructions for `t=4`. User intent: Refine the visual design of the blog post pages by introducing a cohesive color scheme and typography style that enhances readability and aesthetic appeal. Use a serif font for blog titles and a sans-serif font for body text to create a clear hierarchy and visual contrast. Apply a soft background color to the entire page to reduce eye strain and highlight the content areas. Ensure that interactive elements such as social media sharing buttons, comment section, and "Like" buttons have a distinct hover effect to visually indicate interactivity.

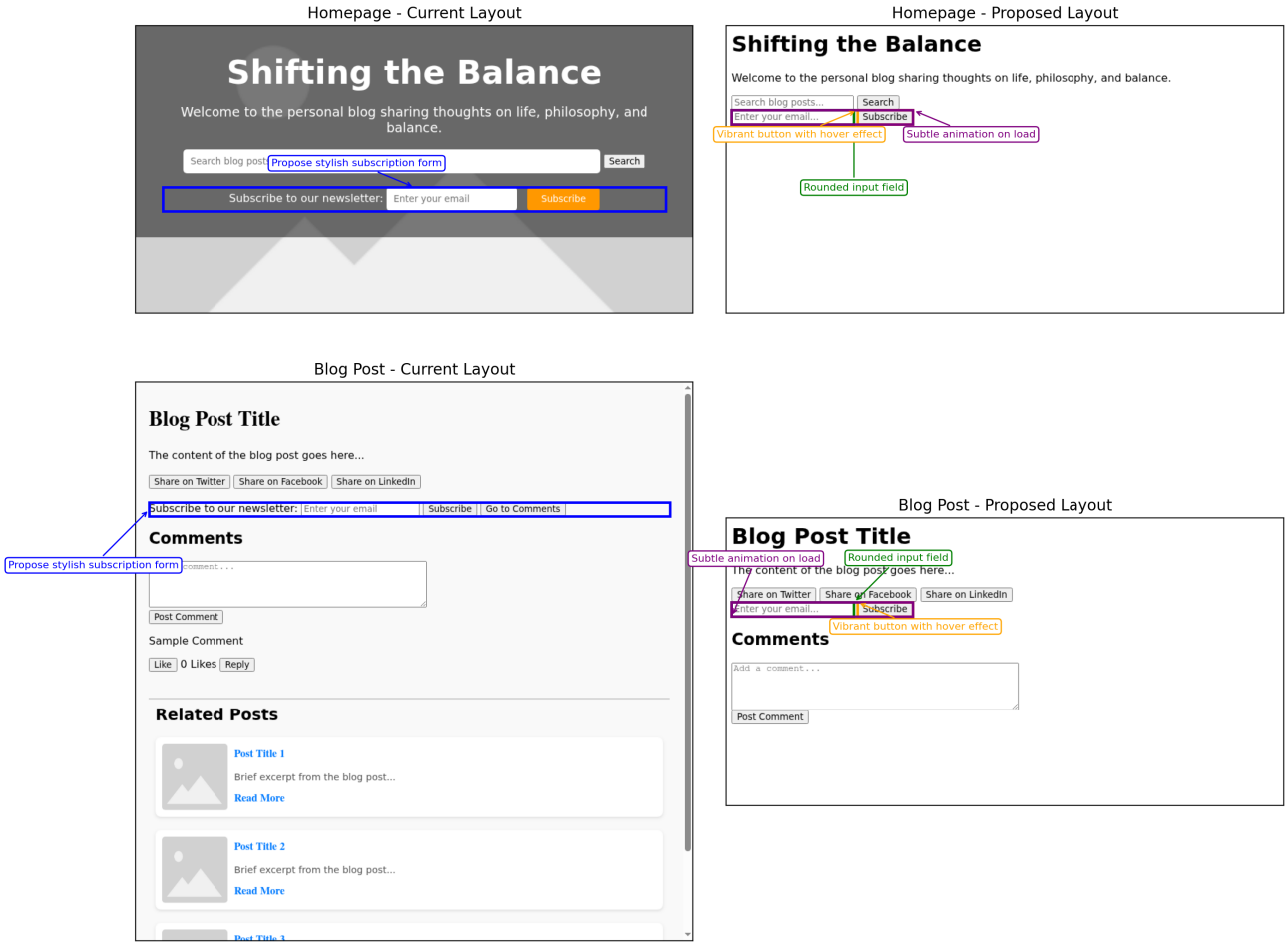



*Figure 11.* Simulated visual instructions for `t=10`. User intent: Enhance the visual design of the email subscription feature across the homepage and each blog post page by introducing a stylish subscription form. Use a rounded input field with a placeholder text that invites users to "Enter your email...". Add a vibrant call-to-action button next to the input field, styled with a hover effect that changes its background color. Ensure the form is positioned prominently within the layout, and add a subtle animation that draws attention to the form upon page load.

listing each labeled element and its properties. Based on these observations, the agent selects an action from a predefined

action space: `Click`, `Hover`, `Type`, `Select`, `Scroll`, `GoBack`, and `Upload`. Most actions require a *target* argument, typically a numbered interactive element or the entire screen. Actions TYPE, SELECT, and `Upload` require an additional content argument, while GOBACK requires no arguments.

We additionally augment the agent with **image manipulation tools**. These tools do not interact directly with the web environment, but instead operate on screenshots from prior turns. They are designed to sharpen the agent's perception and assist in judging test cases that require fine-grained visual perception, such as verifying a subtle hover effect in a local region. The available tools are:

1. `ViewRaw screenshot_x`: view the high-fidelity screenshot of a specific step, without annotated set-of-marks that may potentially hinder evaluation;
2. `Compare [Numerical_Label]; screenshot_x, screenshot_y`: Compare a specific element across two different steps. This tool will crop the local regions for specified element [Numerical_Label] and concatenate the two crops across two screenshots into a single image for comparison.
3. `ViewAnimation [Numerical_Label or WINDOW]; screenshot_x`: View the sequence of screenshots during the transition to `screenshot_x` (from `screenshot_x-1`). Similar to Compare, this tool concatenate the images into a single large image for comparison.

Note that we use the *raw and high-resolution* screenshots in these tools, rather than the annotated and cluttered ones used for input observation. The full prompt for our augmented agent is in Listing 4. To show the benefits of our proposed image manipulation tools, Figure 13 shows an example where evaluator without image manipulation tools fails, but evaluator with image manipulation tools succeeds.

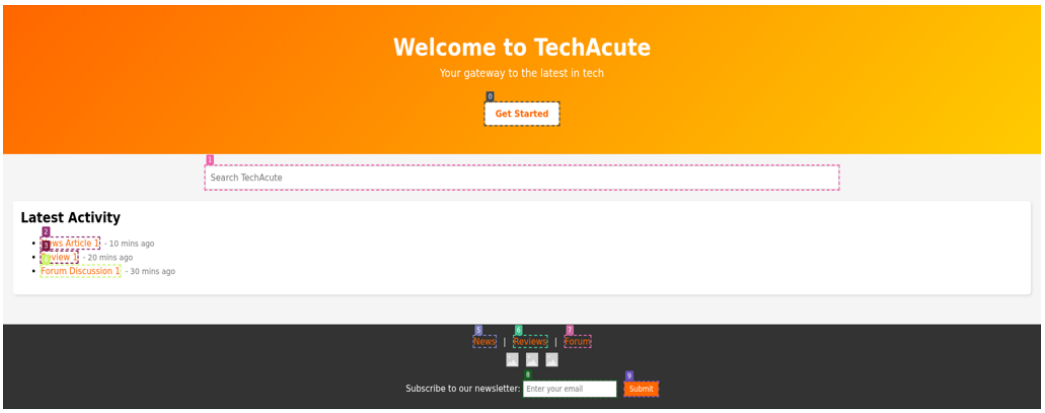



*Figure 12.* An example input observation for our evaluation agent. The textual representation is as follows: [0]: "Get Started"; [1]: <input> ""; [2]: "News Article 1"; [3]: "Review 1"; [4]: "Forum Discussion 1"; [5]: "News"; [6]: "Reviews"; [7]: "Forum"; [8]: <input> ""

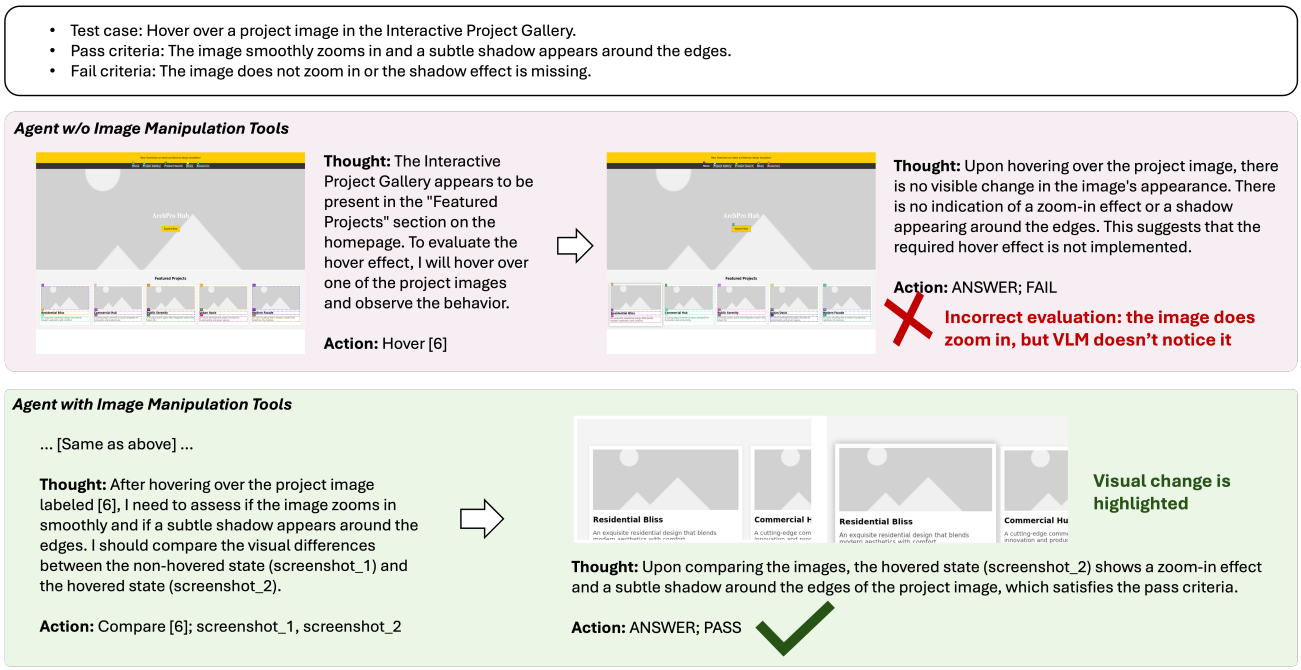



*Figure 13.* Example where evaluator without image manipulation tools fails, but evaluator with image manipulation tools succeeds.

### B.6. AceCoder

Detailed pseudo-code for AceCoder is presented in Alg. 2. In the algorithm, the feedback function $\text{AgentFB}(\mathcal{M}, \tilde{\mathbf{i}}, o) \rightarrow pass, expl$ queries a web agent powered by model $\mathcal{M}$ to verify whether instruction $\tilde{\mathbf{i}}$ is well implemented in output $o$, and provides a binary feedback $pass$ along with a text explanation $expl$.

**Algorithm 2 AceCoder algorithm at each turn.**

**Input:** Model $\mathcal{M}$; current-turn instruction $\tilde{\mathbf{i}}_t$; history $\mathbf{H}_t$
Feedback function $\text{AgentFB}(\mathcal{M}, \tilde{\mathbf{i}}, o) \rightarrow pass, expl$
$o_t \leftarrow \mathcal{M}(\tilde{\mathbf{i}}_t, \mathbf{H}_t)$ ▷ Draft output for turn $t$
$F \leftarrow \emptyset$ ▷ Initialize feedback set
**for** $j \leftarrow 1$ **to** $t$ **do**
  $pass_j, expl_j \leftarrow \text{AgentFB}\left(\mathcal{M}, \tilde{\mathbf{i}}_j, o_t\right)$
  **if** $pass_j =$ **false then**
    $F \leftarrow F \cup \{expl_j\}$ ▷ Collect textual feedback
  **end if**
**end for**
**if** $F \neq \emptyset$ **then**
  $o_t \leftarrow \mathcal{M}(\tilde{\mathbf{i}}_t, \mathbf{H}_t, F)$ ▷ Re-generate with feedback
**end if**
$w \leftarrow \text{Render}(o_t)$
**Return** $w$

To implement the web agent responsible for critique, we adopt a design based on the WebVoyager pipeline, using a prompt similar to Listing 4. Although this agent shares a similar structure with the evaluation agent, we mitigate the risk of label leakage through two key measures: (1) the agent does not rely on the evaluation LLM, but instead uses the baseline LLM to which AceCoder is applied; and (2) the agent is granted access only to the user instructions and the current code, and is never exposed to hidden test cases.

An example from our test set demonstrating AceCoder is as follows. The simulated user instruction is – Enhance the

"Local Attractions" page by adding a feature for users to create a personalized itinerary. This functionality will allow users to select attractions directly from the detailed view of each attraction and add them to their itinerary. Users should be able to include personal notes or preferences for each attraction they add. The itinerary should be accessible from the homepage, with options to view and print it. Ensure smooth navigation between the homepage and the hotel comparison page, allowing users to return without losing their itinerary details. Note that the itinerary feature does not currently exist, and thus must be implemented before these instructions can be fully applied.

A common implementation pitfall for this instruction is that an "Add to Itinerary" button is added to the interface, but is not functional. For the first iteration of model output $\mathcal{M}(H_{t-1}, x_t)$ exhibiting this pitfall, the agent feedback $expl_t$ generated by ACECODER is as follows:

```
Summary:
1. Started on the homepage and navigated to the "Local Attractions" page.
2. Attempted to add attractions to a personalized itinerary using the "Add to Itinerary" buttons, but these actions
were not executable.
3. Accessed the "My Itinerary" page, which displayed an empty itinerary with only a "Print Itinerary" button, and
no options to add notes or preferences.
4. Returned to the homepage to verify if itinerary details persist across navigation, but there was no indication
of any preserved itinerary details.

Analysis:
- The website does not currently support the full functionality of creating a personalized itinerary as instructed.
- The "Add to Itinerary" buttons do not function, and there is no way to add personal notes or preferences.
- The itinerary is not populated with selected attractions, and there is no evidence of itinerary persistence when
navigating between pages.

Missing Elements:
- A functional "Add to Itinerary" feature that allows for adding attractions with personal notes.
- Persistent itinerary details that remain accessible across different pages.
- A confirmation or feedback mechanism to indicate successful addition of attractions to the itinerary.
```

With the user instruction enhanced by this agent feedback, the model successfully addresses this issue and generates functional "Add to Itinerary" buttons. Detailed screenshots illustrating the website before and after using the agent feedback is shown in Figure 14.

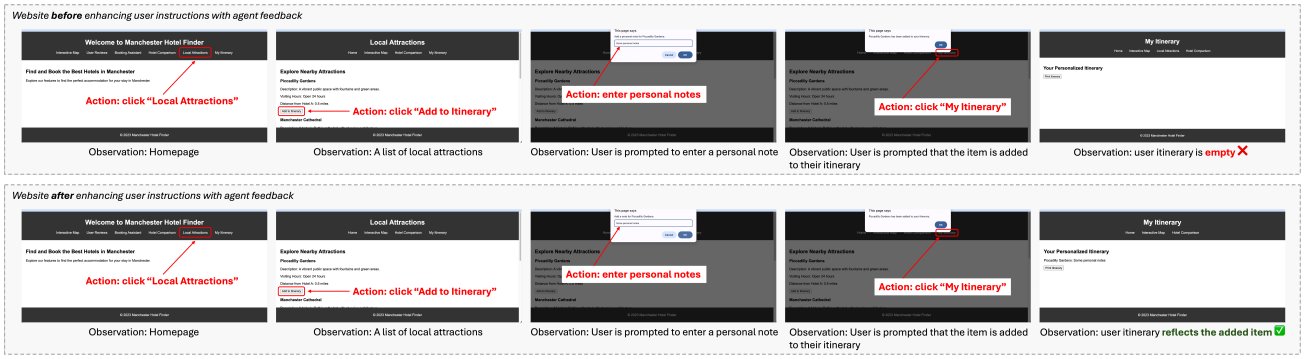



*Figure 14.* Screenshots of an example where ACECODER successfully addresses a pitfall in the implementation.

## C. Additional Quantitative Analysis

### C.1. Long-Context Degradation

In addition to the forgetting issue, **long-context degradation** – where model performance declines as the context length increases – is another challenge that emerges in our multi-turn setting. In the FRONTALK benchmark, the context length can easily exceed 100k tokens, posing a significant challenge for many LLMs. However, this issue is **model-dependent**. While models such as GPT-4o and Gemini-2.5-Pro handle long contexts relatively robustly, others exhibit substantial performance degradation. In contrast, the forgetting issue affects most models, including strong ones like GPT-4o and Gemini-2.5-Pro, even though these models are relatively resilient to long-context degradation.

As illustrated in Figure 15, models like Claude-4-Sonnet and DeepSeek-R1 achieve high pass rates in early turns, but their

performance drops sharply in later turns as the context grows. To further investigate this behavior, Table 10 presents an analysis on a subset of 12 dialogues (426 test cases). The results show that reducing the context window can improve multi-turn performance for Claude-4-Sonnet, even under a strict token limit of 10k. Nevertheless, all multi-turn configurations still significantly underperform compared to their single-turn baselines.

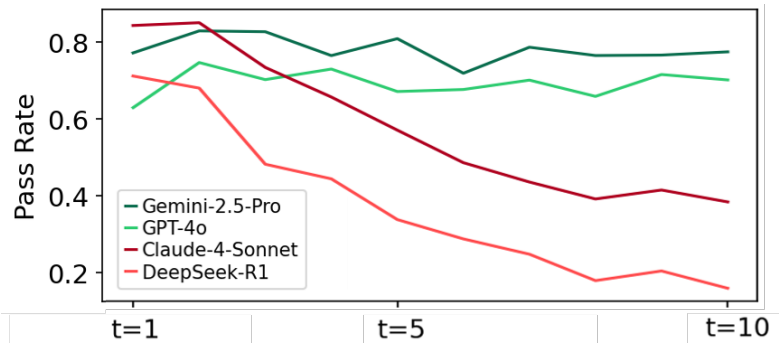



*Figure 15.* **Pass rate across different turns.** We show the pass rate of intermediate outputs, i.e. PR($o_t$|$\mathbf{i}_t$).

*Table 10.* **Claude-4-Sonnet performance when evaluated with different token limits for generation outputs.** For each setting, we report the ratio of truncated output (%Trunc.), pass rate (PR↑) and usability (UX↑).

| | %Trunc. | PR↑ | UX↑ |
|---|---|---|---|
| Multi-turn, token limit = 30k | **5** | 42.0 | 52.1 |
| Multi-turn, token limit = 20k | 23 | 44.6 | 50.0 |
| Multi-turn, token limit = 10k | 88 | 47.8 | 54.2 |
| Single-Turn, token limit = 30k | 6 | **76.1** | **73.5** |

### C.2. Confidence Intervals of Evaluation

To assess the statistical reliability of our results, we conduct multiple evaluation runs for key experiments: GPT-4o and Qwen2.5-VL-72B, each tested both with and without AceCoder. The evaluation is performed on 382 test cases; Each configuration has been evaluated five times. The mean performance and corresponding 95% confidence intervals are reported in Table 11. The narrow confidence intervals indicate that the measurements are stable. Moreover, the confidence intervals for the ACECODER-enhanced models do not overlap with those of their respective baselines, demonstrating that the improvements introduced by ACECODER are statistically reliable despite evaluation variance.

*Table 11.* Confidence interval calculated by repeating evaluation for five times.

| | Pass Rate |
|---|---|
| GPT-4o | 65.1 ± 3.7 |
| +ACECODER | 71.2 ± 2.9 |
| Qwen2.5-VL-72B | 58.8 ± 1.4 |
| +ACECODER | 62.3 ± 1.2 |

*Table 12.* Human alignment for the automatic evaluation of pass rate, measured by agreement accuracy ($acc$) and Cohen's Kappa ($\kappa$). We compare our evaluator against image-based evaluator, code-based evaluator, and an ablated version without image manipulation tools (w/o IM tools).

| | All Types | | Design | | Functionality | |
|---|---|---|---|---|---|---|
| | $acc$ | $\kappa$ | $acc$ | $\kappa$ | $acc$ | $\kappa$ |
| Random evaluator | 50.4 | 0.9 | 52.1 | 4.2 | 48.4 | 0.0 |
| Image evaluator | 70.4 | 39.1 | 70.0 | 37.9 | 72.1 | 37.9 |
| Code evaluator | 72.5 | 42.1 | 68.9 | 37.5 | 76.7 | 46.0 |
| **Agent evaluator** | **80.6** | **59.2** | **78.4** | **56.4** | **83.0** | **62.0** |
| w/o IM tools | 80.2 | 58.9 | 77.8 | 55.3 | 82.9 | 61.9 |
| | -0.4 | -0.3 | -0.6 | -1.1 | -0.1 | -0.1 |

### C.3. Agreement Analysis for Automatic Evaluation

We present the details about assessing the alignment between human evaluators and LLM judges as discussed in §3.

**Pass rate.** We tasked three annotators to independently evaluate 218 test cases. As in Table 12, our agent-based evaluator exhibits strong alignment with human judgments, achieving an accuracy of 80.6 and a Cohen's $\kappa$ of 59.2. In contrast, the

image-based evaluator cannot handle dynamic test cases. The code-based evaluator, while theoretically capable of judging all cases, tends to be overly optimistic and misses subtle errors, leading to weaker alignment and a higher false positive rate (19.0%, compared to 12.2% for our agent-based evaluator). We conduct an ablation study by removing our proposed image manipulation tools from the agent-based evaluator (w/o IM) and measure their resulted alignment scores. Results show that our full evaluator consistently achieves higher alignment, with the benefit being more pronounced for design-category test cases. Figure 13 illustrates an example where these tools are critical for evaluating fine-grained visual details.

**Usability.** We evaluate usability based on interaction trajectories generated by a web agent simulating a first-time user. These trajectories exhibit plausible, human-like interaction behaviors, as illustrated by the example in Table 15. To validate the reliability of automatic evaluation, we tasked three annotators to independently compare a total of 150 trajectory pairs. Analysis of human-LLM alignment yielded an accuracy of 69.3 and a Cohen's $\kappa$ of 56.3 (using quadratic weighting to handle tied votes), indicating moderate to substantial agreement. While the alignment is lower than the main metric of pass rate, reflecting the inherently subjective nature of user experience, it serves as an important complementary metric, filling a critical gap in user-centric evaluation for front-end development. Further analysis reveals that the determinants of usability shift with model proficiency: *For lower-performing models, usability is mostly affected by basic functionality*, where non-functional features like broken navigation links can severely degrade the user experience. *For higher-performing models, usability hinges on nuanced design choices* not always explicitly specified in user instructions, such as intuitive navigation (e.g., an easily accessible navigation bar) and clear system feedback (e.g., a confirmation message after a form submission). These findings highlight front-end development as a multi-dimensional challenge: Boosting user experience requires moving beyond mere implementation correctness to perform thoughtful, user-centric design – a critical direction for future research.

**Agreement between different LLMs as evaluators.** We further assess the reliability of the evaluation pipeline by measuring the agreement across different LLMs as evaluators. We employ three LLMs including GPT-4o, Gemini-2.5-Pro and Qwen2.5-VL-72B, to evaluate a outputs from four representative models. For usability, we evaluate a subset of 355 test cases, while for usability we evaluate the entire test set. Detailed numbers are in Table 13. While the absolute scores varied slightly across judges, **the relative ranking of all models remains consistent across all three LLMs**, demonstrating robust agreement in model ranking and evaluation. We additionally calculated the IAA metrics for pass rate evaluation, achieving an accuracy of 81.4 (Cohen's $\kappa$ of 62.3), indicating **substantial agreement**. These results further demonstrate the reliability of our LLM-based evaluation framework.

*Table 13.* We report evaluation results obtained by three different LLM evaluators, GPT-4o, Gemini-2.5-Pro, and Qwen2.5-VL-72B. We report the pass rate (PR) and usability (UX) for four representative set of model outputs, GPT-4o, Gemini-2.5-Pro, Qwen2.5-VL-72B, and Qwen2.5-VL-7B.

| | GPT-4o | Gemini-2.5-Pro | Qwen2.5-VL-72B | Qwen2.5-VL-7B |
|---|---|---|---|---|
| *GPT-4o Evaluator* | | | | |
| PR↑ | 52.7 **(rank=2)** | 70.1 **(rank=1)** | 45.1 **(rank=3)** | 19.7 **(rank=4)** |
| UX↑ | 55.0 **(rank=2)** | 71.0 **(rank=1)** | 46.5 **(rank=3)** | 26.8 **(rank=4)** |
| *Gemini-2.5-Pro* | | | | |
| PR↑ | 42.8 **(rank=2)** | 61.7 **(rank=1)** | 34.5 **(rank=3)** | 16.6 **(rank=4)** |
| UX↑ | 45.5 **(rank=2)** | 75.0 **(rank=1)** | 35.3 **(rank=3)** | 13.3 **(rank=4)** |
| *Qwen2.5-VL-72B Evaluator* | | | | |
| PR↑ | 46.5 **(rank=2)** | 56.1 **(rank=1)** | 38.0 **(rank=3)** | 20.6 **(rank=4)** |
| UX↑ | 61.0 **(rank=2)** | 61.3 **(rank=1)** | 49.5 **(rank=3)** | 38.5 **(rank=4)** |

### C.4. Human Analysis for User Simulation

To quantitatively assess the reliability of our proposed user simulation, we perform human analysis over 114 simulated instructions, both textual and visual, and their associated 411 test cases. We assess the simulator quality in terms of three rubrics: (1) *plausibility*, representing whether the simulated instruction resembles a realistic human request; (2) *consistency*, indicating whether the simulated instruction is consistent with the multi-turn conversational context, especially the actual web state produced by the model being evaluated; and (3) *fidelity*, measuring whether the simulated instruction faithfully convey the intended user intents, evaluated by the ratio of test cases well represented by the user instructions. We report the evaluation results in Table 14. Textual instruction simulation consistency achieves near perfect scoring across metrics. Visual simulators also exhibit high plausibility (94%) and consistency (97%) because it directly grounds user intents in

rendered screenshots. However, it shows much lower fidelity (76%) due to limited tool-use effectiveness in GPT-4o. An example of failed visual instruction simulation caused by GPT-4o's insufficient understanding of tools is shown in Figure 16, and its corresponding erroneous code snippet is in Listing 3. Despite this limitation, we view visual simulation as a valuable proof of concept: with our benchmark and more capable future models, the reliability of visual user simulation can be further improved.

*Table 14.* User simulator quality evaluation assessed by human evaluators.

| | Plausibility | Consistency | Fidelity |
|---|---|---|---|
| Textual | 98 | 96 | 98 |
| Visual | 94 | 97 | 76 |

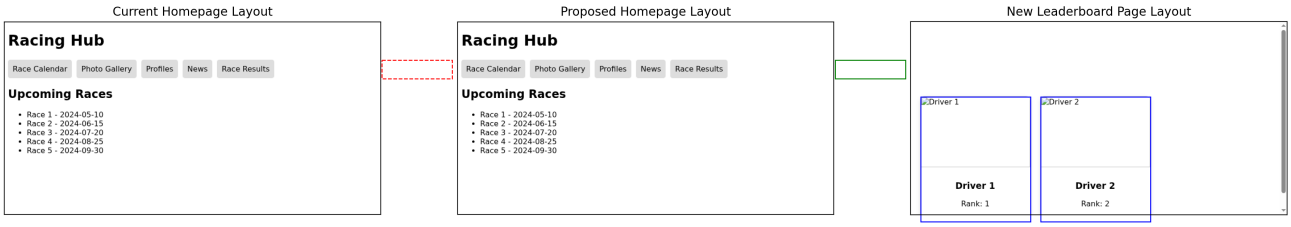



*Figure 16.* Example of a visual simulation failure caused by incorrect tool usage. The user intent that was not shown in the image is: *Introduce a "Leaderboard" page accessible from the homepage via a "Leaderboard" button.* As in Listing 3, the LLM simulator attempted to draw a rectangle to represent an added leaderboard button; however, it incorrectly specified the coordinates, resulting in the rectangle being placed outside the subplot boundaries. Furthermore, the input to `text_annotation` was incorrectly set to `ax` rather than the shape object, so the annotation does not appear in the rendered image.

*Listing 3.* **Code in the *second* step in the visual instruction simulation process.** The resulting image is in Figure 16. Our custom tools are highlighted in gold, and tool use errors are highlighted in red.

```
import matplotlib.pyplot as plt
from matplotlib import patches
from PIL import Image

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Existing Homepage Layout
ax = axes[0]
ax.set_title("Current Homepage Layout")
ax.imshow(Image.open("screenshot_index.png"))

# Highlight where the "Leaderboard" button will be added
(*@\textcolor{red}{nav\_bbox = (8, 80.9, 786, 39)}@*)
leaderboard_button_shape = patches.Rectangle((*@\textcolor{red}{(nav\_bbox[0] + nav\_bbox[2] + 10,
nav\_bbox[1])}@*), 150, nav_bbox[3],
                                        facecolor='none', edgecolor='red', linestyle='--')
ax.add_patch(leaderboard_button_shape)
(*@\textcolor{darkgold}{text\_annotation}@*)((*@\textcolor{red}{ax}@*), "Add 'Leaderboard' button here",
leaderboard_button_shape, color='red')

# Proposed Layout for Homepage
ax = axes[1]
ax.set_title("Proposed Homepage Layout")
ax.imshow(Image.open("screenshot_index.png"))

# Show "Leaderboard" button in the proposed layout
leaderboard_button_shape = patches.Rectangle((*@\textcolor{red}{(nav\_bbox[0] + nav\_bbox[2] + 10,
nav\_bbox[1])}@*), 150, nav_bbox[3],
                                        facecolor='none', edgecolor='green')
ax.add_patch(leaderboard_button_shape)
(*@\textcolor{darkgold}{text\_annotation}@*)((*@\textcolor{red}{ax}@*), "'Leaderboard' button",
leaderboard_button_shape, color='green')
```

# D. Additional Qualitative Analysis

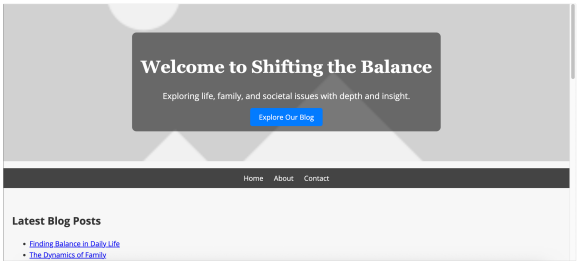
## D.1. Error Cases

An example for forgetting issue is shown in Figure 17. Additionally, we show three error cases for the three types of visual instruction interpretation errors in Figure 18, 19 and 20 respectively.

**t = 5**

**Instruction**: Design an inviting hero section with a large background image...

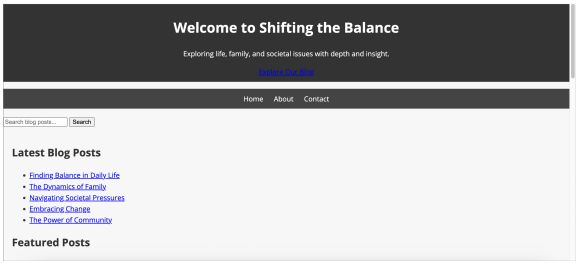
✓ Instructions are well met




...

**t = 8**

✗ Instructions from t = 5 are forgotten




**Reason**: `styles.css` constantly gets rewritten

```
-header {
-    background: #333;
-    color: #fff;
-    padding: 10px 0;
+.hero {
+    background:
url('placeholder/placeholder.png')
no-repeat center center/cover;
+    position: relative;
+    height: 60vh;
+    display: flex;
+    align-items: center;
+    justify-content: center;
+    text-align: center;
+    color: white;
+}

...

+.cta-button:hover {
+    background-color: #0056b3;
+}
```

t = 5: **Hero section is implemented** to replace the old header

```
-.slide img {
+.slide img.zoom-effect {
+    transition: transform 0.3s
ease;
+}
+.slide img.zoom-effect:hover {
+    transform: scale(1.05);
 }

 .prev, .next { ...
-    color: white;
+    color: #007BFF;
-    transition: background-color
0.3s ease;
+    transition: background-color
0.3s ease, transform 0.3s ease;
... }

 .prev:hover, .next:hover {
     background-color:
rgba(0,0,0,0.8);
+    transform: scale(1.1);
 }
```

t = 6: Minor updates

```
+.search-form {
+    display: flex;
+}
+
+.search-form input[type="text"] {
+    padding: 5px;
+    border-radius: 5px 0 0 5px;
+    border: none;
+    outline: none;
+}
+
+.search-form button {
+    padding: 5px 10px;
+    border-radius: 0 5px 5px 0;
+    border: none;
+    background-color: #007BFF;
+    color: white;
+    cursor: pointer;
+    transition: background-color
0.3s ease;
+}
+
...
```

t = 7: Styles for search form is added

```
-.hero {
-    background:
url('placeholder/placeholder.png')
no-repeat center center/cover;
+header {
+    background: #333;
+    color: #fff;
+    padding: 10px 0;
     text-align: center;
-    color: white;
-}
-
-.hero-overlay {

...

+#related-posts {
+    margin-top: 40px;
+}
+
+#related-posts h3 {

...
```

t = 8: Styles for related posts are newly added, but **the hero section is lost and reverted to old header implementation**

*Figure 17.* An example for forgetting issue. The hero section is correctly implemented in `t=5`, but gets forgotten at `t=8`. This is because the `styles.css` file constantly gets rewritten; at `t=8`, the rewriting lost the implementation of hero section, but reverts back to the old header implementation.

***User Intent:***

Enhance the homepage by introducing a visually engaging carousel that showcases featured articles, positioned prominently beneath the search bar. Use large, eye-catching images for each article, accompanied by overlay text that includes the article's title and author. Incorporate smooth transition animations between slides and **include navigation arrows on either side of the carousel for easy manual control**. Ensure that the carousel maintains a consistent aesthetic with the rest of the homepage, using complementary colors and typography.

***Implementation error:*** Navigation arrows are implemented (marked by ), but clicking does not update carousel

***Simulated Visual Instruction:***

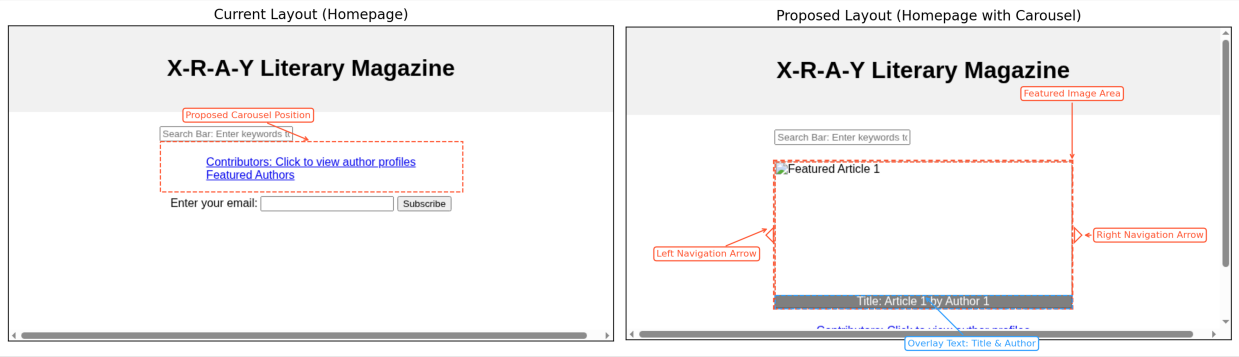
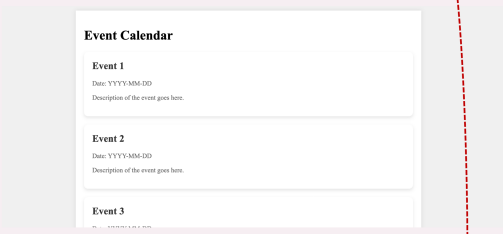



*Figure 18.* Error case for failure to implement implicit functionalities.

***User Intent:***

Refine the Event Calendar page by introducing a visually appealing layout that organizes events in a card format. Each card should have a soft shadow and rounded corners to create a sense of depth and separation. Use a consistent color palette for the event titles and dates, ensuring they stand out against the card background. Incorporate a hover effect that subtly enlarges the card and **brightens the background color** to indicate interactivity. Ensure that the event descriptions are displayed in a legible serif font beneath the event titles.

***Implementation error:*** The hover effect enlarges the card but doesn't brighten the card. Text annotations marked by ○ was missing.




***Simulated Visual Instruction:***

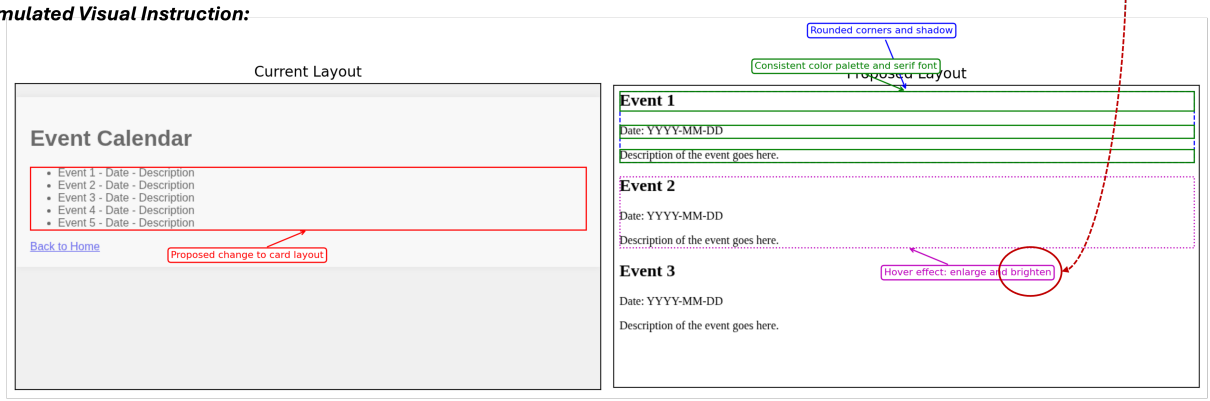



*Figure 19.* Error case for missing text annotations.

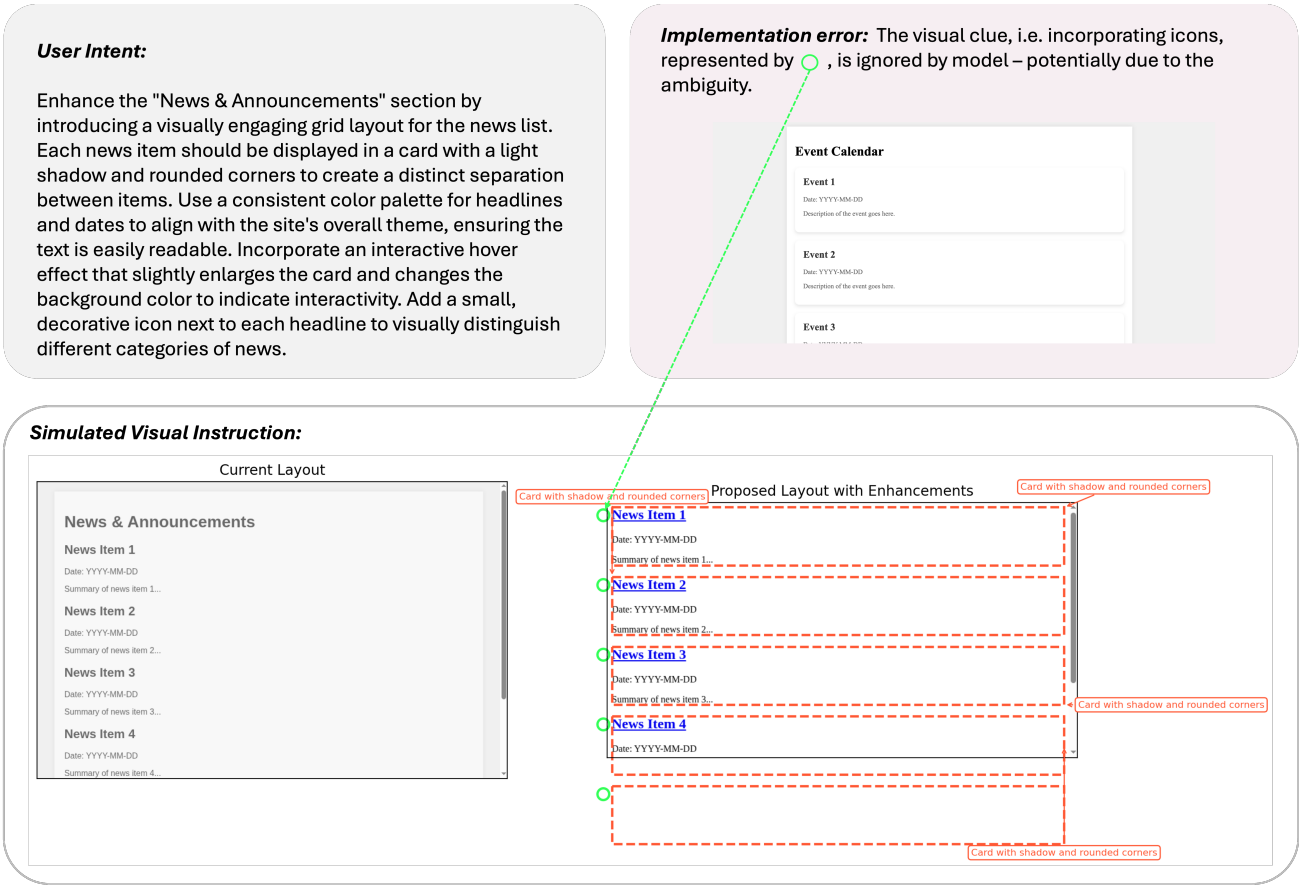
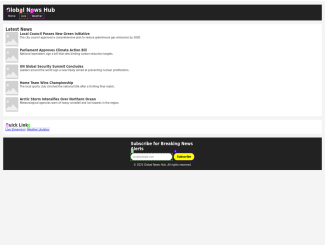
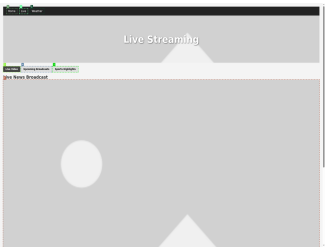
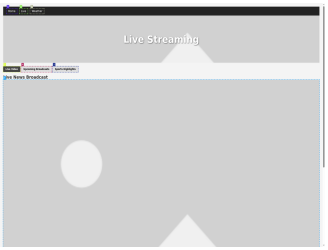
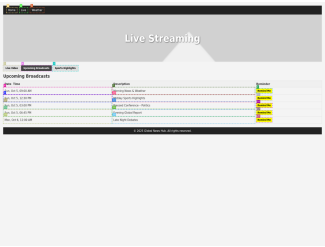
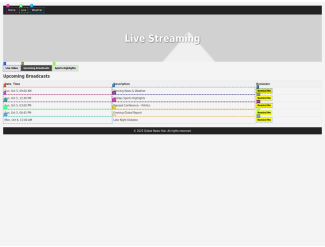



*Figure 20.* Error case for misinterpretation of visual clues.

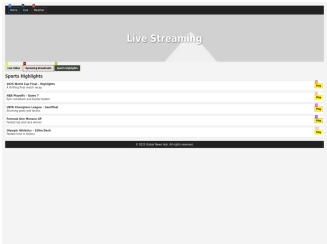
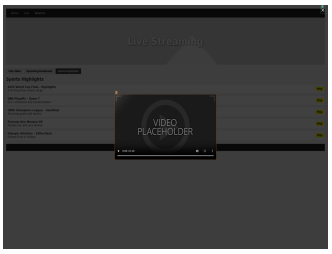
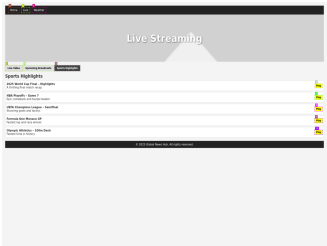
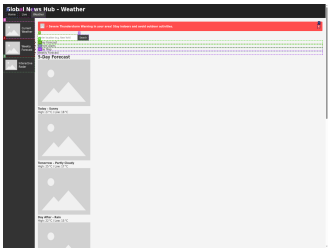
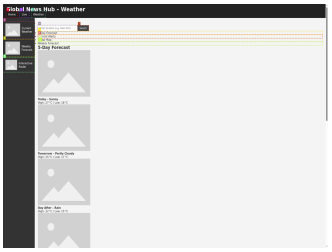
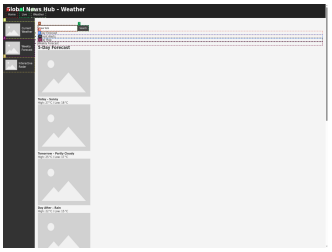
### D.2. Example Trajectory from Usability Evaluation

*Table 15.* Example trajectory from usability evaluation.

| t | Screenshot | Thought and Action |
|---|---|---|
| 1 |  | **Thought:** The website appears to provide news articles with headlines related to local and international events. There are navigation options at the top for "Home," "Live," and "Weather," which likely lead to different sections of the site. There is also a subscription option for breaking news alerts. To explore more about the news coverage, I will click on "Live" to see if it provides real-time updates or live streaming news. **Action:** Click [1] |
| 2 |  | **Thought:** The "Live" section of the website offers options for live video, upcoming broadcasts, and sports highlights. This suggests that users can watch live news broadcasts or check schedules for future events. Since the goal is to understand how to access live news, I will click on "Live Video" to see if it provides access to a live news feed. **Action:** Click [3] |
| 3 |  | **Thought:** Clicking "Live Video" did not change the page content, indicating the video feed may not be functional or requires additional support from the browser. To further explore live news options, I will check "Upcoming Broadcasts" to see if there are scheduled events or programs related to news coverage. **Action:** Click [4] |
| 4 |  | **Thought:** The "Upcoming Broadcasts" section provides a schedule of events, including news and weather updates, sports highlights, press conferences, global reports, and debates. Users can set reminders for these broadcasts, which is useful for keeping track of important events. To test the reminder functionality, I will set a reminder for the "Morning News & Weather" broadcast. **Action:** Click [8] |
| 5 |  | **Thought:** The pop-up message indicates that a reminder was successfully set for the "Morning News & Weather" broadcast. This feature enhances user engagement by allowing them to keep track of specific events. To further explore the website's offerings, I will check the "Sports Highlights" section to see if it provides specific content related to sports news. **Action:** Click [5] |

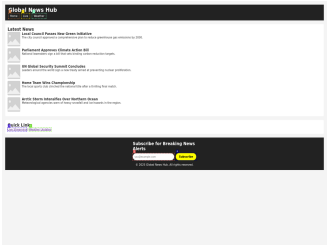
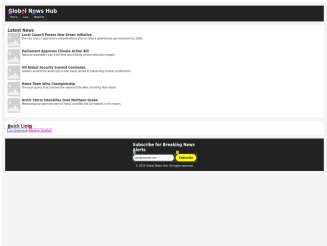
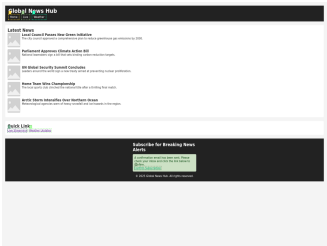
*Continued on next page*

Table 15 – *Continued from previous page*

| t | Screenshot | Thought and Action |
|---|---|---|
| 6 |  | **Thought:** The "Sports Highlights" section offers a variety of sports event recaps, including the World Cup Final, NBA Playoffs, UEFA Champions League, Formula One, and Olympic Athletics. Each event has a "Play" button, indicating video highlights are available. To test the functionality, I will click "Play" for the "2025 World Cup Final - Highlights" to view the content. **Action:** Click [6] |
| 7 |  | **Thought:** Clicking "Play" for the "2025 World Cup Final - Highlights" successfully opened a video player with a placeholder video. This confirms the website's functionality for viewing sports highlights. To return to exploring other sections of the site, I will close the video player. **Action:** Click [0] |
| 8 |  | **Thought:** The video player has been successfully closed, returning me to the "Sports Highlights" section. To explore other areas of news coverage, I will navigate to the "Weather" section to check if it provides current weather updates or forecasts. **Action:** Click [2] |
| 9 |  | **Thought:** The "Weather" section provides a severe thunderstorm warning, a 5-day forecast, current alerts, and an interactive radar. Users can also search for weather updates by location. To explore the current weather conditions, I will dismiss the severe weather warning message to have a clearer view of the page. **Action:** Click [6] |
| 10 |  | **Thought:** The severe weather warning has been dismissed, providing a clearer view of the weather section. There's a search bar available to enter a location for specific weather updates. To test this feature, I will input a location into the search bar to see detailed weather information for that area. **Action:** Type [6]; "New York" |
| 11 |  | **Thought:** I have entered "New York" into the search bar. To view the weather updates for New York, I need to click the "Search" button to apply the location filter and retrieve the relevant information. **Action:** Click [7] |

*Continued on next page*

Table 15 – *Continued from previous page*

| t | Screenshot | Thought and Action |
|---|---|---|
| 12 | | **Thought:** After clicking the "Search" button, the weather section now displays detailed weather information for New York, including high and low temperatures, precipitation, and wind speed for the next five days. This confirms the site's ability to provide location-specific weather forecasts. To continue exploring the website's offerings, I will return to the "Home" section to see what general news coverage is available. **Action:** Click [0] |
| 13 |  | **Thought:** Returning to the "Home" section, I see a list of the latest news articles covering various topics such as environmental initiatives, political actions, sports achievements, and weather events. There is also an option to subscribe for breaking news alerts, which is useful for users who want to stay updated. To test the subscription feature, I will enter an email address into the subscription textbox. **Action:** Type [5]; "user@example.com" |
| 14 |  | **Thought:** I have entered the email address "user@example.com" into the subscription textbox. To complete the subscription process for breaking news alerts, I need to click the "Subscribe" button. **Action:** Click [6] |
| 15 |  | `Step limit reached` |

At the end of the trajectory, we prompt the model to summarize the user experience throught the trajectory. The summary for the trajectory in Table 15 is as follows:

```
The website provides a comprehensive platform for accessing news content across various domains, including local
and international news, weather updates, and sports highlights. Here’s a summary of my experience with the website:

1. **Navigation**: The main navigation options ("Home," "Live," "Weather") are clear and intuitive, allowing users
to easily switch between different sections of the website.

2. **Live Section**: The live section offers video content and schedules for upcoming broadcasts, enhancing user
engagement with live updates and reminders.

3. **Sports Highlights**: This section provides accessible video highlights for major sporting events, adding value
for sports enthusiasts.

4. **Weather Section**: The weather section includes current conditions, forecasts, and alerts. It also features a
location-based search function, which is practical for users seeking specific weather information.

5. **Subscription Feature**: The option to subscribe for breaking news alerts is straightforward, with clear
feedback via a confirmation message upon successful subscription.

Overall, the website is user-friendly, with a clean layout and logical structure that facilitates easy access to
news content. The inclusion of multimedia content and interactive features like setting reminders and searching for
specific weather conditions further enhances the user experience.
```

## E. Prompts

*Listing 4.* Prompt for evaluating **pass rate**.

```
You are an expert evaluator of built websites. You are given a test condition along with
its success/failure criteria. Your task is to determine whether the website **PASS**es or
**FAIL**s the test condition. You can perform the evaluation either by examining a static
screenshot of the homepage or, if necessary, by navigating and interacting with the
website.

At each step, I will provide you with:
1. An annotated screenshot - with **numerical labels** placed in the **top-left corner**
of each web element. Each screenshot is named as screenshot_x (x=1,2,...)
2. A simplified **textual representation** of the page -- including the tag names and
texts for every element annotated in the screenshot.

Your goal is to generate **ACTIONS** to perform the evaluation. If the information is
sufficient to make a judgment, your ACTION should be: `ANSWER; PASS` or `ANSWER; FAIL`.
Otherwise, you should generate actions to interact with the website before producing the
answer. Choose one of the following valid action formats:

### Valid Actions

Action should **STRICTLY** follow the format:
- Click [Numerical_Label]
- Hover [Numerical_Label]
- Type [Numerical_Label]; [Input_Text]
- Select [Numerical_Label]; [Option_Text]
- Scroll [Numerical_Label or WINDOW]; [up or down]
- GoBack
- Upload [Numerical_Label]; [Filename]
- ANSWER; [content]

Additional actions to inspect visual details, **STRICTLY** following the format:
- Compare [Numerical_Label]; screenshot_x, screenshot_y
- ViewRaw screenshot_x
- ViewAnimation [Numerical_Label or WINDOW]; screenshot_x

### Action Guidelines

1. Execute only one action per iteration.
2. **Avoid repeating** the same action if the page does not change -- you may have chosen
the wrong element or label.
3. To input text, you do **not** need to click the textbox first -- just use the `Type`
action. Pressing `ENTER` is handled automatically. However, you may still need to click a
search button afterward to apply a filter.
4. Clearly distinguish between textboxes and buttons -- do **not** type into a button. If
no textbox is visible, consider clicking a search button to reveal it.
5. To upload a file, you do **not** need to click the upload button first -- just use the
`Upload` action and specify the filename. The filename **MUST** be chosen from:
`placeholder.png`, `placeholder.mp4`, `placeholder.mp3`, or `placeholder.pdf`.
6. Use `Compare` to compare the same element's visual display across two **different**
screenshots. For example, compare the screenshots before (screenshot_x) and after
(screenshot_y) a hover action to test hover effect.
7. Use `ViewRaw` to retrieve the high-fidelity raw screenshot without annotations.
8. Use `ViewAnimation` to view animated behavior when loading screenshot_x. To focus on a
specific element, provide its numerical label.

### Evaluation Guidelines

1. You are also given the instructions for building the website, which **MAY NOT** align
with the actual website structure. Use them **only as context** to help understand the
website, but **always** rely on **real interactions** with the website to make the final
evaluation.
```

```
2. First verify whether the feature to be tested is present on the website. If it is
missing (e.g., the test condition refers to a header, but the website does not have a
header), the evaluation should be FAIL.
3. If the feature to be tested is not immediately visible on the homepage but can be
accessed from it, you must first navigate from the homepage to find it. **Extensively**
explore the website to locate the feature, such as by scrolling down, checking menus, or
following links, until you confirm whether the feature exists.
4. After navigating to the feature, if the test condition is based on **purely static**
visual features such as color and layout, do not use any more actions -- directly output
`ANSWER` based on the screenshot.
5. When a test involves **multiple steps or questions**, use `ANSWER` only **after**
addressing all of them.
6. **Extensively** interact with the website to trigger behaviors relevant to the test
condition. For example, when testing a search bar, test it with multiple inputs including
both reasonable pseudo-data and generic entries like "1", "2", "a", "b".
7. The annotated screenshot is **NOT** what end users actually see. To evaluate the
visual design, **always** rely on additional actions to gather visual details before
making the judgment.
8. The website uses placeholder data. Do **not** judge pass/fail outcomes based on
whether the displayed data reflects real-world data.
9. The website also uses placeholder content for media (images, videos, audio, PDFs). Do
**not** judge pass/fail outcomes based on the **content** of these files. However, you
should evaluate their display and interactive behavior when determining pass/fail.

### Your Reply Format

Thought: {Step-by-step reasoning}
Action: {One properly formatted action}
```

*Listing 5.* Prompt for **simulating first-time users** in the evaluation of **usability**.

```
Your task is to evaluate the **usability** of a website. You will simulate a **first-time
user**: you are given only the website's high-level goal without detailed instructions.
Your task is to extensively explore the website, infer what you want to accomplish as an
end user, attempt the tasks, and judge how easy it is to learn and use the site to
complete them.

At each step, I will provide you with:
1. An annotated screenshot - with **numerical labels** placed in the **top-left corner**
of each web element.
2. A simplified **textual representation** of the page -- including the tag names and
texts for every element annotated in the screenshot.

At each step, you can choose one of the following valid action formats:

### Valid Actions

Action should **STRICTLY** follow the format:
- Click [Numerical_Label]
- Hover [Numerical_Label]
- Type [Numerical_Label]; [Input_Text]
- Select [Numerical_Label]; [Option_Text]
- Scroll [Numerical_Label or WINDOW]; [up or down]
- GoBack
- Upload [Numerical_Label]; [Filename]

### Guidelines

1. Execute only one action per iteration.
2. **Avoid repeating** the same action if the page does not change -- you may have chosen
the wrong element or label.
3. To input text, you do **not** need to click the textbox first -- just use the `Type`
action. Pressing `ENTER` is handled automatically. However, you may still need to click a
search button afterward to apply a filter.
```

```
4. Clearly distinguish between textboxes and buttons -- do **not** type into a button. If
no textbox is visible, consider clicking a search button to reveal it.
5. To upload a file, you do **not** need to click the upload button first -- just use the
`Upload` action and specify the filename. The filename **MUST** be chosen from:
`placeholder.png`, `placeholder.mp4`, `placeholder.mp3`, or `placeholder.pdf`.
6. The website uses placeholder for data and media (images, videos, audio, PDFs).

### Your Reply Format

Thought: {Describe image content, then perform step-by-step reasoning}
Action: {One properly formatted action}
```

*Listing 6.* Prompt for performing **pair-wise comparison** in the evaluation of **usability**.

```
You are an expert UX Researcher specializing in usability analysis. Your analysis must be
objective, evidence-based, and grounded in established usability principles.

You will be given two user interaction trajectories, `Trajectory A` and `Trajectory B`.
Your task is to perform a pairwise comparison to determine which website offers a better
user experience. You will provide a detailed analysis and a final, single-word verdict
for Website A: WIN, LOSE, or TIE.

### Evaluation Criteria

Base your analysis *strictly* on the following usability dimensions:

1. **Journey Success:** Did a goal emerge from the user's browsing? If so, how well did
the website support their journey from discovery to potential action? Did they
successfully navigate to a satisfying endpoint or did they abandon their exploration?
2. **Efficiency & Path:** How much effort (cognitive load, number of actions,
backtracking) was required? Was the user's path logical and fluid, or was it disjointed
and filled with unnecessary steps?
3. **Satisfaction & Friction:** How did the user feel? Identify specific moments of
frustration, confusion, delight, or ease, as indicated by their thoughts.
4. **Discoverability & Clarity:** How easy was it for the user to understand the site's
structure, find what's available, and learn what actions are possible? Is the value
proposition clear from the start? This is especially critical for **Exploratory** tasks.

### Instructions

Follow these steps and structure your response exactly as follows.

1. **Identify Emergent Goal(s) (if any):** Briefly state the primary goal(s) the user
developed during each trajectory.
2. **Analyze Trajectory A:** Summarize the user's experience on Website A, citing
specific evidence for each of the four evaluation criteria.
3. **Analyze Trajectory B:** Do the same for Website B.
4. **Comparative Analysis:** Directly compare Website A and Website B on each of the
criteria. State which website performed better and why.
5. **Final Verdict & Justification:** Conclude with a verdict for Website A and a concise
justification highlighting the most critical factors. Formulate your final verdict as
follows:

VERDICT: [WIN, LOSE, or TIE]
```

*Listing 7.* Prompt for **textual instruction simulator** for functionality-type user intents. The gold text refers to user intent, and the blue text refers to the current code state.

```
You are provided with instructions to **add or refine functionalities** of a website.
Your task is to refine the instructions so they are accurate, specific, and actionable.
Use the code of the existing website as context.
1. **Verify component or functionality references.** These references typically involve
**where** to implement the new feature, or **how** to navigate to the new feature from
the homepage.
```

```
   * If a referenced component/functionality exists, refine vague references to be more
precise.
     * Example: Change "add a button to the main menu" to "add a button to the dark gray
navigation bar at the top".
   * If it does **not** exist, clearly note that it must be implemented before the
instruction can be applied.
2. **Clarify vague instructions.** If an instruction is not specific enough, make it more
concrete based on the actual website.
3. **Avoid code-level details.** Do not reference class names, HTML attributes, color hex
codes, or other implementation-specific identifiers.

Your refined instructions should be a short paragraph (3-6 sentences). Start your refined
instructions with
**Response:**

# Instructions to Refine

(*@\textcolor{darkgold}{USER INTENT}@*)

# Code for Existing Website

(*@\textcolor{LinkBlue}{\#\# filename1}@*)

(*@\textcolor{LinkBlue}{file content 1}@*)

(*@\textcolor{LinkBlue}{\#\# ...}@*)
```

*Listing 8.* Prompt for **textual instruction simulator** for design-type user intents. The gold text refers to user intent, and the blue text refers to the current code state.

```
You are provided with instructions to **refine the visual design** of a website. Your
task is to refine the instructions so they are accurate, specific, and actionable. Use
the code of the existing website as context.
1. **Check for already implemented instructions.** Compare each instruction to the
current state of the website. If an instruction is already implemented, remove it from
the refined list.
   * Example: If the instruction says "Use a two-column layout" but the site already has
a two-column layout, remove that instruction.
   * Keep all unfulfilled instructions exactly as they are - **do not omit or change
them** unless covered by steps 2-3 below.
2. **Verify component or functionality references.**
   * If a referenced component/functionality exists, refine vague references to be more
precise.
     * Example: Change "the button" to "the green submit button in the bottom left".
   * If it does **not** exist, clearly note that it must be implemented before the
instruction can be applied.
3. **Clarify vague instructions.** If an instruction is not specific enough, make it more
concrete based on the actual website.
4. **Avoid code-level details.** Do not reference class names, HTML attributes, color hex
codes, or other implementation-specific identifiers.

Your refined instructions should be a short paragraph (~5 sentences). Start your refined
instructions with
**Response:**

# Instructions to Refine

(*@\textcolor{darkgold}{USER INTENT}@*)

# Code for Existing Website

(*@\textcolor{LinkBlue}{\#\# filename1}@*)

(*@\textcolor{LinkBlue}{file content 1}@*)
```

```
(*@\textcolor{LinkBlue}{\#\# ...}@*)
```

*Listing 9.* Prompt for **visual instruction simulator**. The gold text refers to user intent. The screenshots for every rendered page along with their filenames (in the format of `screenshot_name.png`) are appended in the following messages.

```
You are an expert drawing agent with a series of drawing tools based on `matplotlib`.
Your task is to visualize a series of user instructions about implementing or refining
the {{functionality|visual design}} of a website.

For each page in the website, you are given:
* A screenshot (with its filename),
* The HTML code, and
* The coordinates of elements in the page.

You can load and display a screenshot with:
```python
from PIL import Image
ax.imshow(Image.open("screenshot_name.png"))
```

# User Instructions

(*@\textcolor{darkgold}{USER INTENT}@*)

# Tool Documentation

You have the following four tools:
1. Subplot organization
2. Layout visualization
3. Shape drawing
4. Text annotation

You'll generate python code to call these tools - I'll explain the tools one by one. Your
code should be wrapped within ``python ```. Make sure to import necessary tools and
libraries you want to use.

## Subplot Organization

When visualizing the instructions, you may need to work with multiple subplots:
1. **First subplot - always the screenshot of relevant page**. Use shapes and text
annotations to link each instruction to its corresponding component. For example:
   * If the instruction states, "Use a dark background color with white text for the menu
items", add annotations to the menu bar.
   * If it states, "Display the discount price in a bright color", annotate the product
card.
   * If it states, "Use a circular profile photo with a thin border", annotate the
profile picture accordingly.
2. **Revised layout subplot(s)** - for layout changes, if necessary.
   1. In the first subplot, highlight the components to be moved and indicate their
intended movement.
   2. In the later subplot(s), show the components in their **new positions** with shapes
and text annotations - using the **same annotation color** for each component in both the
original screenshot and the updated layout views for clarity.
   3. To show the updated layout: (1) If the UI changes are minor, reuse the screenshot
and mark it with shapes showing the new positions. (2) For major changes, use the HTML
layout visualization tool to create a simplified updated layout.
3. **State transition subplot(s)** - for interaction-driven changes, if necessary
   1. In the first subplot, highlight the interactive component(s) that trigger the
transition and annotate interaction type.
   2. In the new subplot, show the resulting end state and clearly indicate what changed.
4. **Titles**: Each subplot should have a **clear and descriptive title** to clearly
indicate the relationship and transitioning between subplots (e.g., "Current Layout",
"Proposed Layout", "After Clicking `Submit`").
5. **Multiple pages**: If the instructions involve multiple pages, you can allocate one
or multiple subplots for each page following the instructions above. Annotations for each
```

```
page will be based on its own screenshot.

You may create subplots **only in your first drawing turn**. Before starting, decide
exactly how many subplots you need and what **specific page state** each will represent.
Once created, each subplot is fixed to a single, distinct view - you cannot add or remove
subplots later.

Use matplotlib to create multiple subplots, and use each element of `axes` for further
drawing:
```python
fig, axes = plt.subplots(n_row, n_col)
```

Note: For each subplot, **NEVER** directly draw on top of blank canvas background! You
**must** define a background using one of the following methods:
1. Screenshot background: If you are visualizing the existing page or interaction flow,
use the screenshot: `ax.imshow(Image.open("current_screenshot.png"))`
2. HTML layout background: If you are visualizing a new layout or interface, use the
layout visualization tool: `layout_visualization(html_code, ax)`. After this call, wait
for coordinate data before adding shapes or annotations.

## Layout Visualization

To visualize the components required in the instructions, you should create a **minimal**
HTML layout **only use boxes and text**. Avoid colors, fonts, and detailed styling.

**Important**: You may receive reference HTML in the input. These are for context only -
**do not** copy or reuse them in your `layout_visualization` call.

Use the following to visualize HTML layout in a given `ax`:
```python
html_code = """<HTML code here>"""
layout_visualization(html_code, ax)
```
Assume the function `layout_visualization` is already implemented - do not redefine it.
After you call it, I will provide coordinates for each element in the next round. You can
then use those to draw shapes or text annotations in future steps. **Do not** perform any
annotations on `ax` before receiving the coordinates.

## Shape Drawing

Use `matplotlib.patches` to draw rectangles, circles, ellipses, or polygons. Shapes are
typically used for: (1) Highlighting specific regions for which you have instructions; or
(2) Indicating the addition of new components, such as a circle for an icon or a
rectangle for a new block.

Examples:
```python
new_shape = patches.Rectangle((x, y), width, height, facecolor='none',
edgecolor='#??????')  # draw rectangle
new_shape = patches.Circle((x_center, y_center), radius, facecolor='none',
edgecolor='#??????')  # draw circle
new_shape = patches.Ellipse((x_center, y_center), width, height, facecolor='none',
edgecolor='#??????')  # draw ellipse
new_shape = patches.Polygon([[x1, y1], [x2, y2], [x3, y3]], closed=True,
facecolor='none', edgecolor='#??????')  # draw polygon
```

and then draw the shape:
```python
ax.add_patch(new_shape)
```

Shape drawing is usually paired with text annotation - you first draw a shape to
highlight a specific component, then annotate it with text. When doing so, follow these
```

```
color guidelines for clarity and visual coherence:
1. **Consistent color for pairing**: Use the same `edgecolor` for the shape and `color`
for the text annotation to visually link them as a pair
2. **Shared annotation across components**: If a single annotation (e.g., "add shadow
effect") applies to multiple components, avoid repeating the same annotation for each
one. Instead, draw shapes around **all relevant components** using the same distinctive
color, and annotate **just one** of them with a note explaining that the annotation
applies to all shapes of that color.
3. **Distinct colors for distinct meanings**: If you are applying **multiple different
annotations**, use **different colors** for each group of shape/text pairs to clearly
distinguish between them.

## Text Annotation

To annotate text, **NEVER use `ax.text` or `ax.annotate` directly**! **Always** use the
following function:
```python
text_annotation(ax, text, new_shape, color='#??????')
```
* `new_shape` should be a Rectangle, Circle, Ellipse, or Polygon (as described in the
Shape Drawing section) you want to annotate.
* This function draws an arrow pointing to the shape, with a labeled box at the arrow's
origin. The box and arrow will both use the specified `color`, creating a visually
consistent annotation.
* Placement and overlap are automatically managed to ensure clarity and avoid visual
clutter.
* The `text_annotation` function is already implemented - do not redefine it.

## Multi-Turn Tool Calling

You can complete the drawing in a single turn or across multiple turns. After each turn,
I will provide the current figure. If you've called `layout_visualization` in the turn, I
will also provide the coordinates for every element in the HTML - so if necessary, you
can draw annotations over the HTML visualization.

Most tasks can be completed in one turn. However, if you need to annotate elements based
on coordinates of HTML layout, it is acceptable to use multiple turns.
```