

000 001 002 003 004 005 COMPUTER-USE AGENTS AS JUDGES FOR AUTOMATIC 006 GUI DESIGN 007 008 009

010 **Anonymous authors**
011 Paper under double-blind review
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031

ABSTRACT

032 Computer-Use Agents (CUA) are becoming increasingly capable of au-
033 tonomously operating digital environments through Graphical User Interfaces
034 (GUI). Yet, most GUI remain designed primarily for humans—prioritizing aes-
035 thetics and usability—forcing agents to adopt human-oriented behaviors that are
036 unnecessary for efficient task execution. At the same time, rapid advances in
037 coding-oriented language models (Coder) have transformed automatic GUI de-
038 sign. This raises a fundamental question: *Can CUA as judges to assist Coder for*
039 *automatic GUI design?* To investigate, we introduce **AUI-Gym**, a benchmark for
040 **Automatic GUI** development spanning 52 applications across diverse domains.
041 Using language models, we synthesis 1560 tasks that simulate real-world scenar-
042 ios. To ensure task reliability, we further develop a verifier that programmati-
043 cally checks whether each task is executable within its environment. Building
044 on this, we propose a **Coder-CUA in Collaboration** framework: the Coder acts
045 as Designer, generating and revising websites, while the CUA serves as Judge,
046 evaluating functionality and refining designs. Success is measured not by visual
047 appearance, but by task solvability and CUA navigation success rate. To turn
048 CUA feedback into usable guidance, we design an **CUA Dashboard** that com-
049 presses multi-step navigation histories into concise visual summaries, offering in-
050 terpretable guidance for iterative redesign. By positioning agents as both designers
051 and judges, our framework shifts interface design toward agent-native efficiency
052 and reliability. Our work takes a step toward shifting agents from passive use
053 toward active participation in digital environments.

1 INTRODUCTION

032 Recent advances in language agents have shown that **Computer-Use Agents** [openai \(2025\)](#); [An-](#)
033 [thropic \(2025\)](#) can autonomously operate within GUIs—performing tasks such as online shopping
034 by sequentially clicking through multiple buttons [Zhou et al. \(2024\)](#). However, today’s environ-
035 ments remain fundamentally human-centric, optimized for aesthetics and usability through features
036 like dynamic animations or colorful layouts. To adapt to these settings, researchers typically train
037 CUA on large-scale human demonstration trajectories, click logs, or static screenshots [Xu et al.](#)
038 [\(2024\)](#); [Lin et al. \(2024b\)](#); [Seed \(2025\)](#), effectively forcing agents to imitate human behavior. This
039 approach binds automation to human-oriented design choices, where stylistic details crucial for hu-
040 mans are redundant for agents whose primary objective is efficient task completion. In parallel,
041 coding-oriented language models—**Coders**—have already demonstrated strong capabilities, capa-
042 ble of generating functional HTML pages or even entire websites from a single instruction [Si et al.](#)
043 [\(2024\)](#). Yet these outputs remain confined to human-facing loops: even when generated by agents,
044 interfaces are still optimized for human use rather than agent-native interaction.

045 Both CUA and Coders thus exhibit remarkable potential for automation and design. This motivates
046 a fundamental question: Can CUA assists Coders redesign UIs in an automatic manner—where
047 environments are created for, and evaluated by, agents themselves, with CUA acting as judges? In
048 this work, we reconceptualize the UI as a tunable environment. The core idea is to employ the
049 **Coder as Designer**—responsible for initializing and revising UIs—while the **CUA acts as Judges**,
050 navigating through tasks and collecting interaction trajectories as feedback.

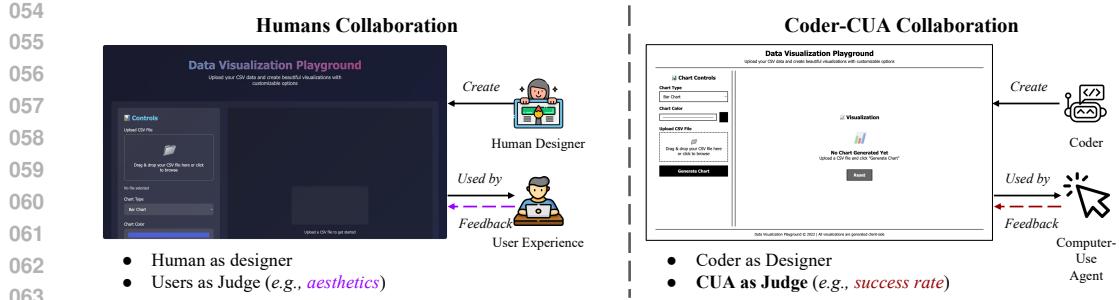


Figure 1: Illustration of Humans Collaboration vs. our Coder-CUA Collaboration in term of UI designs. Left: Most GUIs are designed by humans and optimized for user experience (e.g., aesthetics), forcing trained agents to adapt to human-oriented behaviors. **Right:** Our Coder-CUA Collaboration framework leverages Coder as Designer and CUA as Judge together, enabling more reliable task execution and improved usability for agents.

As no existing testbed aligns with our goal, we introduce AUI-Gym to pioneer evaluation in this setting. AUI-Gym automatically develops websites across 52 applications spanning six domains (apps, landing pages, games, interactive demos, tools, and utilities). Unlike most coders that focus on single-page generation, AUI-Gym requires agents to produce fully automated, executable, application-level designs with an emphasis on functional completeness. Enabling sufficient, scalable, and human-free evaluation is non-trivial. However, is non-trivial. To simulate realistic usage scenarios, we prompt GPT-5 to propose 30 candidate tasks per application, yielding 1560 tasks in total. These tasks are then validated by humans. To ensure that each website can be reliably tested, GPT-5 also generates a customized rule-based functional checker for individual task, determining whether the task is feasible within the given interface. This infrastructure establishes a human-free, reliable foundation for subsequent CUA exploration and feedback-driven UI refinement.

To this end, we develop a **Coder–CUA collaboration** framework. The *Coder acts as Designer*, responsible for UI initialization and refinement, while the *CUA serves as Judge*, supplying feedback. The central challenge is how to transform raw CUA interactions into effective revision signals from an agent perspective. We address this through two complementary dimensions of feedback: **(a) CUA Navigation**, where the agent executes tasks through atomic actions such as clicks or typing and judges success or failure; and **(b) Task Solvability**, where unsolvable tasks are accumulated as functionality failures and returned to the Coder as precise indicators of missing features. CUA navigation produces long, multi-step trajectories interleaved with screenshots, making direct feedback difficult to interpret. To overcome this, we introduce the **CUA Dashboard**, which condenses each task, its outcome, actions, and intermediate states into a single 1920×1080 image. Rather than storing every screenshot, the dashboard highlights only key interactive regions, with region sizes adaptively scaled by the number of steps. This dynamic design reduces redundancy by average 76.2% while preserving essential cues, offering a clear step-by-step view of how the CUA perceives and acts on the interface. As a result, success and failure points become immediately visible, and the dashboard provides concise, interpretable feedback that the Coder to guide iterative UI redesign.

Our empirical results show that while state-of-the-art Coders can generate complete GUIs that appear suitable to humans, they still encounter notable limitations: **(i) Task solvability as a foundation.** Initial UIs often fail to capture many practical scenarios, resulting in low usability. However, by collecting failure cases, the Coder can readily boost performance through language-based functional summarization. **(ii) CUA navigation as a key bottleneck.** Even when UIs achieve high functional completeness, CUAs initially exhibit low success rates due to the complexity of multi-step navigation. Through our Coder–CUA collaboration, we substantially improve navigation success rates, particularly showing that CUA feedback-driven redesigns—such as *de-stylization*, increased contrast, and simplified layouts—significantly enhance CUA execution. Together, these findings highlight the promising potential of agents for automatic UI design and testing, improving both task success and robustness. To summarize, our contributions are threefold:

1. **AUI-Gym: a scalable testbed for automatic GUI development and testing**, covering 52 applications across six domains with 1560 GPT-5–proposed, human-validated tasks and per-task rule-based checkers. This enables human-free development of automatic UI creation and testing.
2. **Coder–CUA framework with CUA Dashboard**. The Coder initializes and refines UIs while the CUA judges via two signals: navigation outcomes and task solvability. A single-image 1K

108 CUA Dashboard compresses task goal, actions, intermediate states, and outcome by highlighting
 109 key interactive regions with adaptive scaling, reducing visual tokens by 76.2% on average while
 110 preserving essential cues for redesign.
 111

112 **3. Evaluation insights.** Task solvability is foundational yet readily improved via failure-driven
 113 functional summarization, whereas CUA navigation is the main bottleneck. Feedback-driven
 114 redesigns (e.g., de-stylization, higher contrast, simplified layouts) substantially raise execution
 115 success and overall robustness.

116 2 RELATED WORKS

117 2.1 COMPUTER-USE AGENTS

120 Recent studies reveal the potential of LLMs beyond language modeling, with advancements in
 121 demonstrating their ability to autonomously complete complex tasks using tool integration [Schick et al. \(2023\)](#) like humans. This has prompted the development of GUI automation agents that learn
 122 to operate digital user interfaces by imitating human trajectories. This learning is primarily achieved
 123 in two ways: (i) by steering general multimodal foundation models [Achiam et al. \(2023\)](#) with in-
 124 context human trajectory examples, and the general models perceive the UI through intermediate
 125 representations like HTML, accessibility trees [Drouin et al. \(2024\)](#); [Gao et al. \(2023\)](#); [Zheng et al. \(2024\)](#), Optical Character Recognition [Lu et al. \(2024\)](#), or Set of Masks [Yang et al. \(2023\)](#). (ii) by
 126 pre-training specialized GUI foundation models through extensive supervised fine-tuning or rein-
 127 forcement learning on large-scale vision-text UI data (e.g., screenshots and instructions) [Xu et al. \(2024\)](#);
 128 [Lin et al. \(2024b\)](#); [Gou et al. \(2024\)](#); [Lin et al. \(2024a\)](#); [Lu et al. \(2025\)](#); [Seed \(2025\)](#).
 129 While foundational, these data-driven approaches suffered from heavy requirements for high-quality
 130 human trajectories to achieve agent performance improvements. Despite their methodological dif-
 131 ferences, these approaches share a common, agent-centric paradigm, focusing on improving the
 132 agent’s capabilities to navigate a static and often complex environment. Notably, we investigate a
 133 complementary approach. Instead of adapting the agent, we explore how to dynamically tune the
 134 environment to enhance the performance of a frozen agent.
 135

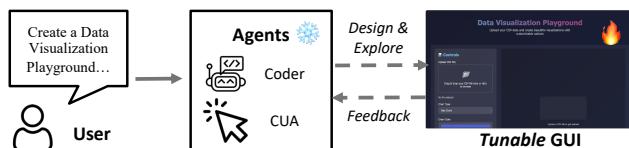
136 2.2 AUTOMATIC SOFTWARE DESIGNS

137 Besides CUAs, there have been extensive research on software automation, automatic interface
 138 design [Lu et al. \(2023\)](#); [Kong et al. \(2008\)](#) and generation [Si et al. \(2024\)](#); [Beltramelli \(2018\)](#);
 139 [Laurençon et al. \(2024\)](#). Programmatic and semantic UI components—such as accessibility lay-
 140 ers, ARIA tags, and declarative interface frameworks (e.g., React Native, Flutter)—illustrate how
 141 environments can be annotated or abstracted for automated processes. Similarly, benchmarks in au-
 142 tomated software interaction, such as WebArena [Zhou et al. \(2024\)](#) and GAIA [Mialon et al. \(2023\)](#),
 143 assume agent operates within fixed, human-oriented systems for task automation. More recently,
 144 embodied AI environments (e.g., ALFRED [Shridhar et al. \(2020\)](#), Habitat [Puig et al. \(2023\)](#), Mine-
 145 Dojo [Fan et al. \(2022\)](#)) show how environments can be crafted to accelerate agent training, though
 146 primarily in physical or simulated domains. These efforts highlight the growing recognition that
 147 environments themselves can be reimaged for machine interaction, yet a systematic framework
 148 for designing agent-centric digital environments in everyday computing remains absent.
 149

150 3 AUI-GYM BENCHMARK

151 3.1 TASK DEFINITIONS

152 We develop AUI-Gym for automatic
 153 GUI development and testing. Given
 154 a language user query Q as input and
 155 several available agents (e.g., Coder
 156 or CUA), the output is a complete
 157 website that serves as a *tunable* en-
 158 vironment \mathcal{E} . We detail the input and
 159 output respectively below.
 160



161 **Figure 2: AUI-Gym task definition.** A user issues a request
 162 (e.g., “Create a Data Visualization Playground”), and agents
 163 (e.g., Coder or CUA) interact with the GUI through design,
 164 exploration, and feedback. In this setup, the GUI serves as a
 165 *tunable* environment.

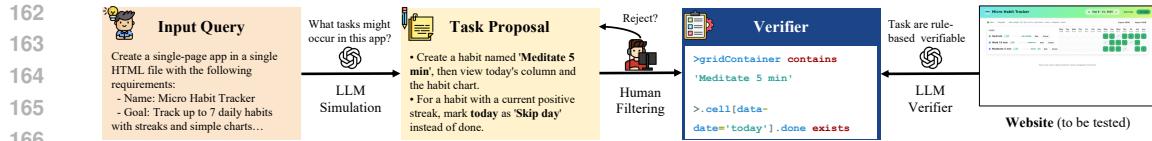


Figure 3: **AUI-Gym construction pipeline.** (i) An input query specifies the app requirements. (ii) GPT-5 proposes candidate tasks with explicit goals. (iii) Humans filter and refine tasks using domain-specific principles. (iv) A test-time Verifier reads the website HTML and generates task-specific, rule-based checkers to validate success on the to-be-tested website.

Input Query \mathcal{Q} . Since the outcome is a website, the user query \mathcal{Q} should be both descriptive and concrete. To this end, we explicitly standardize queries into the structured format illustrated above. This supplements the query with a name, goal, functional features, and UI theme.

Input formulation

Create a single-page app in a single HTML file with the following requirements:
- Name: {Camping Gear Checklist}
- Goal: {Track gear for camping trips}.
- Features: {Checklist items, weight calculator, save lists.}
- Theme: {The UI should be outdoor-themed.}

Output website \mathcal{E} . The website is an application-level deliverable that must be fully functional, going beyond a static page to support navigation, transitions, button interactions, and completion of functional goals, with the objective of maximizing the agent’s success rate. Constructing an effective evaluation framework in this setting is non-trivial and introduces several challenges. We next present our scalable, automatic solutions.

3.2 TASK CREATION

The full curation pipeline is illustrated in Fig.3. To construct the benchmark, we collect 52 task prompts from OpenAI’s playground¹, covering multiple domains.

Synthesize candidate tasks \mathcal{T} . Applications are typically designed to support a variety of relevant tasks, and a key evaluation is whether they can smoothly handle such tasks. We leverage GPT-5 OpenAI (2025) to synthesize diverse user requirements: given an instruction \mathcal{I} , it generates a set of candidate tasks \mathcal{T} that simulate practical usage. As illustrated in Fig. 3, for the application ‘*Micro Habit Tracker*’, an example task is: “*Create a habit named ‘Meditate 5 min’, then view today’s column and the habit chart.*” These tasks serve as fine-grained probes that capture the potential demands of the environment \mathcal{E} .

Manual quality control. As the tasks are automatically generated by GPT-5, human oversight is required to ensure their quality. Different applications demand different characteristics: for example, tasks for game UIs should emphasize interactivity and control, while tasks for utility tools should capture information accessibility and workflow patterns. To this end, humans define domain-specific principles and filter out low-quality tasks (e.g., trivial clicks) or nonsensical ones (e.g., beyond the application scope), ambiguous query (cross-application), ensuring that the proposed tasks remain concrete, meaningful and aligned with each domain’s design philosophy.

Data Statistics. Based on the above strategy, we obtain 30 tasks for each application. The benchmark spans 52 web applications across six domains, yielding a total of 1,560 tasks and enabling comprehensive evaluation across diverse applications. As illustrated in Table 7, the domains include: (i) App, general-purpose applications; (ii) Landing, commercial and promotional interfaces; (iii) Game, puzzle and arcade-style challenges; (iv) Interactive, dynamic user engagement with real-time feedback; (v) Tool, specialized utilities; and (vi) Utility, everyday organizational support. This diverse coverage captures distinct GUI challenges—ensuring robust evaluation across varied interaction paradigms and functional complexities.

¹<https://github.com/openai/gpt-5-coding-examples>

216
 217 **Table 1: Examples of App domains in AUI-Gym.** For each domain, we show a website created
 218 by GPT-5, paired with 30 tasks (blue) simulating real-world usage. Each task is further linked to a
 219 rule-based verifier (green). See full distribution and examples in Tab.7.

220 Domain	221 #Apps	222 Percentage	223 Example Instruction	224 GUI created by GPT-5
225 App	226 11	227 21%	<p>Create a single-page app in a single HTML file with the following requirements:</p> <ul style="list-style-type: none"> - Name: Healthy Meal Tracker - Goal: Log meals and nutrition info. - Features: Ingredient list, calories per meal, daily summary. - The UI should be clean with food icons. <p>Task: Add five meals for today's date (any names/ingredients) so today's meal count reaches at least 5.</p> <p>Rule: <code>#dailyMealCount >= 5</code></p>	

230 3.3 EVALUATION WITH VERIFIERS

231 Even with the proposed tasks, it remains challenging to determine whether a given GUI can truly
 232 satisfy them, as websites are interactive and highly diverse environments. More importantly, since
 233 the *GUIs are generated at test time*, it is difficult to design fixed standards that generalize across
 234 all cases, given the variety of possible implementation approaches. A naive solution is to adopt
 235 a VLM-as-Judge approach, but this inevitably introduces bias and uncertainty. Ideally, the most
 236 reliable solution would be concrete functional checks with manual validation, yet this approach is
 237 prohibitively expensive and labor-intensive.

238 To address this, we define a **Verifier** $\mathcal{V}(\cdot)$ powered by GPT-5 at test time, which takes as input a
 239 candidate GUI together with a specific task. It analyzes the available elements and states, reasoning
 240 over the presence of required UI components, their properties, and potential interaction paths. If the task is deemed solvable, the Verifier produces
 241 a task-specific verification **function checker** $\tilde{\mathcal{V}}(\cdot)$ (by JavaScript) that encodes the success condition
 242 by element status; otherwise, the task is discarded as invalid, preventing noisy or unachievable
 243 goals from disrupting evaluation. Such as in Fig.3, for task “*Create a habit named ‘Meditate 5 min,’*
 244 *then view today’s column and the habit chart.*”, based on the candidate website (right), the verifier
 245 generate the rule `gridContainer contains 'Meditate 5min'` In this way, the Verifier is
 246 customized for each website and each task at test time, ensuring reliable validation.

```
Verifier(input = GUI_HTML, task):
    analyze elements and states
    if task solvable:
        return (Yes, function_checker)
    else:
        return (No, None)
```

247 **Metrics.** With the support of function checkers as reliable verification, we can ensure that a website
 248 is both actionable and workable for the CUA. This further allows us to evaluate whether tasks are
 249 completed after CUA navigation, thereby measuring task success rate within the UI environment. In
 250 this way, we devise the following measure:

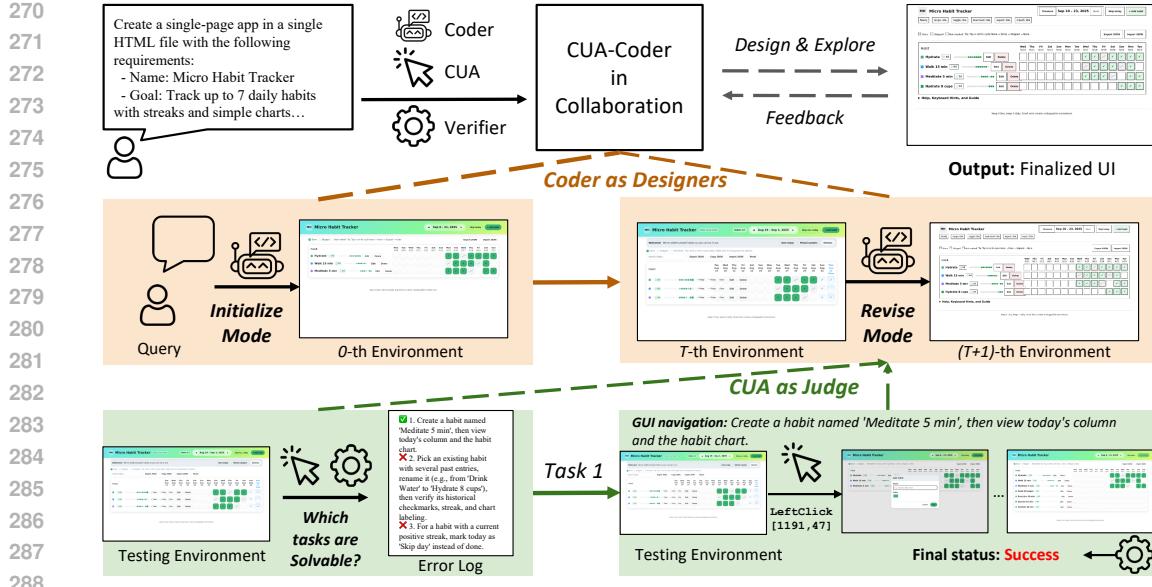
251 **(i) CUA Success Rate (SR).** This measures the average success rate over all tasks executed by CUA.
 252 If CUA successfully completes a task, it is counted as a success; otherwise, it is counted as a failure.
 253 Notably, if the Coder fails to yield a functional checker, the task is counted as a failure.

$$254 \quad 255 \quad 256 \quad 257 \quad 258 \quad 259 \quad 260 \quad 261 \quad 262 \quad 263 \quad 264 \quad 265 \quad 266 \quad 267 \quad 268 \quad 269 \quad SR = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \mathbf{1}(\text{task } t \text{ is successfully completed}), \quad (1)$$

270 where \mathcal{T} denotes the set of all tasks and $\mathbf{1}\{\cdot\}$ is the indicator function.

271 **(ii) Function Completeness (FC).** While CUA performance reflects the ultimate goal, it may be
 272 sparse if most CUAs fail to complete tasks. Therefore, we devise a second metric to evaluate only
 273 whether the Coder-created website functionally supports the task (valid), independent of CUA nav-
 274 igation. This metric reflects task validity and serves as a more basic measure.

$$275 \quad 276 \quad 277 \quad 278 \quad 279 \quad FC = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \mathbf{1}\{\text{a functional checker exists for task } t\}. \quad (2)$$



289 **Figure 4: Overview of the Agent-centric User Interface (AUI) framework.** The process begins
290 with the Coder as Designer, which initializes and iteratively revises the UI based on queries and
291 feedback. In parallel, the CUA as Judge executes task-driven navigation within the testing environ-
292 ment, generating trajectories and error logs to evaluate task solvability. A verifier ensures functional
293 correctness, while feedback from CUA navigation informs subsequent UI revisions. This collabora-
294 tion yields a finalized agent-centric UI optimized for both functionality and execution success.

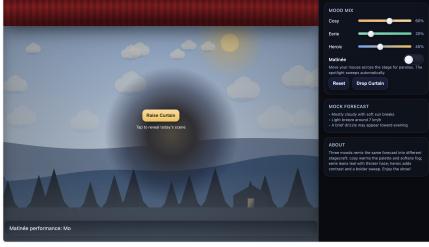
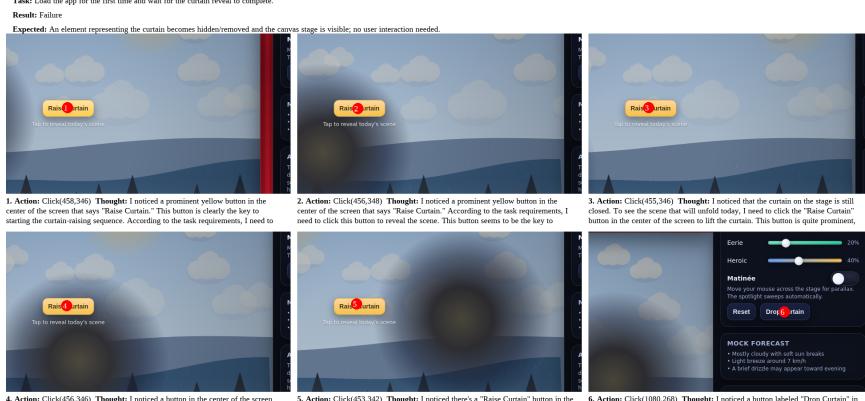
297 4 CUA–CODER IN COLLABORATION

299 **Overview.** We present our framework for enabling collaboration between the CUA and the Coder,
300 consisting of two main components: the Coder as Designer while the CUA as Judge. Given a user
301 instruction Q , AUI generates an initial UI environment \mathcal{E}_0 , which is iteratively revised through inter-
302 action and feedback. The framework involves two central roles: a Coder policy π_{Coder} that proposes
303 and revises UI designs, and a CUA policy π_{CUA} that explores the UI and evaluates its functionality.
304 We formalize this process as a Markov Design Process. The state is the current UI \mathcal{E}_t , the action
305 is a design update proposed by π_{Coder} , and the transition deterministically $\mathcal{E}_{t+1} \leftarrow \pi_{\text{Coder}}(\mathcal{E}_t, \mathcal{R}_t)$.
306 The feedback \mathcal{R}_t is relate to the metrics (i.e., Eq.1 and Eq.2) results achieved by the CUA when
307 interacting with \mathcal{E}_t , i.e., $\mathcal{R}_t \leftarrow S(\mathcal{E}_t, \pi_{\text{CUA}})$. The Coder is optimized to maximize the total reward
308 $\mathbb{E} \left[\sum_t \gamma^t \mathcal{R}_t \right]$. In this formulation, the CUA acts as a user that provides actionable feedback by
309 testing the environment, while the Coder serves as a designer who integrates this feedback into code
310 revisions to iteratively improve the UI. Unlike conventional CUA setups, where the agent adapts to a
311 fixed environment $\pi_{\text{CUA}} \leftarrow \mathcal{E}$, our framework adapts the environment itself based on CUA feedback
312 $\mathcal{E} \leftarrow \pi_{\text{CUA}}$, thereby optimizing UIs for agent-native success. We illustrate the full workflow in Fig. 4
313 and detail each role in the following subsections.

315 4.1 CODER AS DESIGNERS

317 Recent advances in Coder OpenAI (2025); Qwen (2025); anthropic (2025) demonstrate strong capa-
318 bilities in generating UI applications. In our framework, we position Coders as *designers*, responsi-
319 ble not only for creating new environments but also for refining them based on feedback from CUAs.
320 Accordingly, Coders operate in two complementary modes: one dedicated to the initial creation of
321 UIs, and the other focused on their iterative improvement through CUA-guided feedback.

322 **i. Initialization.** Given a user query defined in formulation 3.1 and enriched with multiple details,
323 the Coder progressively generates long-context code to construct a complete HTML-rendered UI \mathcal{E}_0
324 from scratch, which serves as the base environment for subsequent interactions.

324	Task	Load the app for the first time and wait for the curtain reveal to complete.
325		
326		
327		
328		
329	Website	
330		1280 × 720
331		
332		
333		
334		
335		
336		
337		
338		
339		
340		
341	Dashboard (an image)	
342		
343		
344		
345		
346		
347		
348		
349		
350	Result	Failure
351	Comments	The weather-theatre app requires button clicks to trigger curtain reveal, but the task expects automatic curtain opening on first load without user interaction, creating a fundamental mismatch between expected auto-start behavior and actual manual activation requirement.
352		
353		
354		
355	Table 2: Illustration of CUA Dashboard. The dashboard generates one informative image that	
356	clearly demonstrates how the CUA performs each step along with the corresponding observations,	
357	while reducing visual tokens by cropping to the key interactive regions.	
358		
359		
360	ii. Revision from Feedback. After constructing the initial environment \mathcal{E}_0 , the Coder enters an	
361	iterative refinement loop to update the UI: $\mathcal{E}_{t+1} \leftarrow (\mathcal{E}_t, \mathcal{R}_t)$, where \mathcal{R}_t denotes the feedback signal	
362	expressed as a language caption, described in the next section.	
363		
364	4.2 CUA AS JUDGES	
365		
366	We employ Computer-Use Agents (CUAs) as <i>Judges</i> to trial and diagnose the UIs \mathcal{E}_t generated	
367	by the Coder, providing actionable feedback for iterative redesign. Specifically, we define two	
368	complementary forms of reward signals:	
369	(i) Task Solvability Feedback $\mathcal{R}_{\text{task}}$. Before navigation begins, we verify whether a task τ is	
370	implementable on the current UI. Let \mathcal{V} denote the verifier in Sec. 3.3. A task is deemed solvable if	
371	and only if $\mathcal{V}(\mathcal{E}_t, \tau) = 1$; otherwise it is labeled a <i>functional failure</i> . This gate prevents wasted	
372	rollouts on impossible tasks and sharpens the feedback signal. We collect all failed tasks into $\mathcal{T}_{\text{fail}} =$	
373	$\{\tau : \mathcal{V}(\mathcal{E}_t, \tau) = 0\}$ and return them to the Coder as precise indicators of missing features. The	
374	Coder then aggregates and summarizes these failures into a language feedback signal $\mathcal{R}_{\text{task}}$.	
375	(ii) CUA Navigation Feedback \mathcal{R}_{nav}. For solvable tasks $\mathcal{T}_{\text{succ}} = \{\tau : \mathcal{V}(\mathcal{E}_t, \tau) = 1\}$, eval-	
376	uation proceeds as a UI navigation problem. At step k , the CUA receives an observation o_k (a	
377	screenshot of the current state), emits an action $a_k \in \{\text{CLICK}, \text{TYPE}, \text{SCROLL}, \dots\}$ with an optional	
	reasoning trace, and the environment transitions to the next state, yielding o_{k+1} . The trajectory	

378 **Table 3: Main results on AUI-Gym per Coder.** Top: Function Completeness Rate (%). Bottom:
 379 CUA Success Rate (%).
 380

Coder	Feedback Type	landing (%)	game (%)	app (%)	utility (%)	interactive (%)	tool (%)	overall (%)
<i>Function Completeness</i>								
GPT-5	Baseline	53.0	77.8	70.6	63.3	73.0	70.0	67.9
	+ Task Solvability	19.7	100.0	69.4	65.6	55.6	56.2	60.5
	+ CUA Navigation	53.3	87.8	74.2	70.0	70.4	69.5	70.8
	+ Integrated	75.3	92.2	85.2	73.3	82.6	76.7	81.5
Qwen3-Coder-30B	Baseline	16.3	50.4	41.2	43.9	52.2	54.8	42.1
	+ Task Solvability	55.0	79.6	58.5	67.8	56.3	74.3	64.3
	+ CUA Navigation	23.3	50.4	38.8	49.4	39.3	55.2	41.3
	+ Integrated	47.7	72.2	59.7	56.7	57.0	69.5	60.1
GPT-4o	Baseline	9.7	55.2	36.1	38.9	44.8	37.6	36.3
	+ Task Solvability	23.7	55.9	52.1	55.0	58.9	65.2	50.6
	+ CUA Navigation	8.3	55.2	28.2	34.4	26.3	35.7	30.4
	+ Integrated	16.3	68.5	36.4	51.7	51.1	41.4	43.1
<i>CUA Success Rate</i>								
GPT-5	Baseline	34.7	24.8	27.3	14.4	18.1	21.9	24.5
	+ Task Solvability	16.3	39.3	26.7	16.1	20.7	11.9	22.6
	+ CUA Navigation	17.7	43.3	30.0	21.1	21.1	17.6	25.7
	+ Integrated	40.7	27.4	31.5	22.2	14.1	12.9	26.0
Qwen3-Coder-30B	Baseline	5.3	9.3	9.1	11.7	7.0	1.4	7.3
	+ Task Solvability	14.7	42.2	19.1	14.4	11.1	4.3	18.3
	+ CUA Navigation	6.7	20.7	9.1	11.1	12.2	11.4	11.7
	+ Integrated	23.7	30.7	22.4	7.8	9.3	13.8	19.0
GPT-4o	Baseline	4.7	12.6	12.4	6.7	9.3	5.7	8.8
	+ Task Solvability	8.7	18.5	19.1	5.6	8.5	22.9	14.1
	+ CUA Navigation	5.7	31.5	10.0	8.3	10.4	6.7	12.3
	+ Integrated	10.3	27.4	13.9	13.3	15.2	16.7	16.1

402
 403 $\mathcal{H} = (o_0, a_0, \dots, o_K)$ terminates when either (a) the function checker signals success $\tilde{\mathcal{V}}(\mathcal{E}_t, \tau) = 1$,
 404 or (b) a step limit is reached, which we record as a failure. We log full trajectories—observations,
 405 actions, and intermediate rationales—and use them to construct targeted feedback for UI refinement.
 406

407 **CUA Dashboard for Compact Feedback.** Raw trajectories \mathcal{H} are long and interleaved, making
 408 them ill-suited for direct ingestion by the Coder. We therefore distill each rollout into an CUA
 409 Dashboard (Fig. 2): a single, fixed-resolution (1920×1080) canvas that compresses key evidence
 410 from the trial. Rather than storing full frames, we crop and tile only *interactive regions* touched
 411 by the CUA, allocating dynamic region sizes based on step order to preserve temporal structure.
 412 This yields a substantial reduction in redundancy (e.g., a 76.2% drop in visual content) while retain-
 413 ing the cues needed to localize failure modes (missed affordances, hidden state, ambiguous labels)
 414 and success paths at a glance. The dashboard provides a step-by-step visual trace aligned with ac-
 415 tions, making error locations immediately visible. Finally, we convert the dashboard into a concise
 416 language summary \mathcal{R}_{nav} by passing it to the Coder and as the feedback used in the revision rule.
 417

5 EXPERIMENTS

5.1 SETTINGS

421 **Baselines.** For the Coder, we select representative models including GPT-5 [OpenAI \(2025\)](#), GPT-
 422 4o [Hurst et al. \(2024\)](#), and the open-source Qwen3-Coder [Qwen \(2025\)](#). For the CUA, we adopt
 423 UI-TARS-1.5-7B [Seed \(2025\)](#) and Operator [openai \(2025\)](#). UI-TARS-1.5-7B is a lightweight yet
 424 strong performer among open models with high efficiency, and Operator is among the state-of-
 425 the-art close-source API-based CUAs. The experiment results highlight that the performance gain
 426 brought by our proposed method is universal for both lightweight and powerful CUAs.
 427

5.2 MAIN RESULTS

428 [Table 3](#) reports results across six domains for three coders. Several key findings emerge: **(i) Function**
 429 **Completeness.** Revision based on task solvability feedback leads to substantial gains, con-
 430 sistently boosting the overall functionality completeness for all coders. After applying integrated

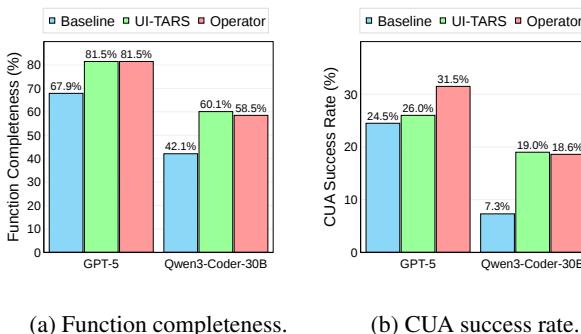
432 revision for GPT-5, the function completeness is increased to 81.5% from 67.9%, reaching the high-
 433 est. Notably, the landing, game and app domains have dramatic improvements, with the maximum
 434 improvements of 38.7%. Interestingly, Revision based on task solvability feedback or CUA nav-
 435 igation feedback alone does not guarantee function completeness improvents, but the integrated
 436 revision combining these two components bring stable improvents on all domains for all coders,
 437 highlighting the strength of our design. Moreover, fixing unresolved functionalities alone also ben-
 438 efits CUA task solving, yielding a 4.8% average improvement on CUA evaluation, highlighting the
 439 mutual reinforcement between task solvability and CUA navigation.

440 **(ii) CUA performance.** Open-source CUAs initially perform poorly, with an average overall CUA
 441 success rate of only 13.5%. However, our framework can consistently improve the CUA success
 442 rate, with an average 6.8% improvements. Interestingly, our framework bring large improvements
 443 to weak coders such as Qwen3-Coder-30B and GPT-4o, with a maximum overall improvement of
 444 11.7%, showcasing that our framework can greatly empower weak models. Overall, these results
 445 demonstrate the effectiveness of our framework: task solvability feedback guides to robust UI de-
 446 sign, while leveraging CUA navigation feedback optimizes interfaces toward agent-centric success.
 447

448 6 KEY ABLATIONS

450 **Effects by different CUAs choices.** In Fig. 5, we compare UI-TARS and Operator as CUA policies
 451 within the integrated revision loop. We evaluate with two coders—GPT-5 (closed-source, stronger)
 452 and Qwen3-Coder-30B (open-source, weaker)—to cover both capability and licensing spectra. Both
 453 CUA policies yield comparable gains in functional completeness, with UI-TARS slightly outper-
 454 forming on Qwen3-Coder-30B. Although the task-solvability signal is identical across CUAs, UI-
 455 TARS tends to fail more tasks, thereby surfacing richer failure cases and driving greater function-
 456 oriented revisions. For CUA success rate (SR), Operator delivers larger gains with the stronger
 457 coder (GPT-5), while improvements are similar across CUAs for the weaker coder. This suggests
 458 Operator’s navigation strengths are best realized on more complex UIs, whereas weaker coders often
 459 produce simpler interfaces. Overall, lightweight open-source CUAs like UI-TARS are an efficient
 460 and effective choice for harvesting navigation feedback in practice.

461 See the Appendix and for more ablations (Sec.A) and quantification examples (Sec.E).



475 **Figure 5: Effect by different Coders**, including open-source UI-TARS and SoTA Operator.

476 7 CONCLUSION

478 We introduced AUI-Gym, a new benchmark for automatic GUI development (52 applications; 1,560
 479 tasks with programmatic checkers), and a Coder–CUA collaboration framework that recasts UI
 480 design as an agent-native loop, with the Coder as Designer and the CUA as Judge. Central to this loop
 481 is the CUA Dashboard, which compresses long navigation trajectories into compact, interpretable
 482 summaries that convert raw interactions into actionable revision signals. Empirically, task solvabil-
 483 ity is foundational—readily improved by failure-driven functional summarization—whereas CUA
 484 navigation remains the primary bottleneck; feedback-driven redesigns (*e.g.*, de-stylization, higher
 485 contrast, simplified layouts) consistently raise execution success and robustness, highlighting the
 value of designing *for* agents rather than merely adapting human-centric interfaces.

486 ETHICS STATEMENT
487488 This work proposes a UI generation benchmark AUI-Gym, a multi-agent framework AUI and does
489 not involve sensitive or private information. Human annotation was conducted with informed
490 consent and fair compensation. We see minimal risk of harm; potential misuse (e.g., generating mis-
491 leading visualizations) is noted, and we release our benchmark strictly for research purposes.
492493 REPRODUCIBILITY STATEMENT
494495 We provide details of dataset construction, evaluation protocols, and model settings in the main text
496 and appendix. All data used are publicly available, and our benchmark, code, and evaluation scripts
497 will be released upon publication to facilitate replication of our results.
498499 REFERENCES
500501 Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Ale-
502 man, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical
503 report. *arXiv preprint arXiv:2303.08774*, 2023.

504 Anthropic. Claude 3.7 sonnet system card. 2025.

505 anthropic. Introducing claude 4, 2025. URL <https://www.anthropic.com/news/claude-4>.506 Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Pro-
507 ceedings of the ACM SIGCHI symposium on engineering interactive computing systems*, pp. 1–6,
508 2018.509 Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom
510 Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena:
511 How capable are web agents at solving common knowledge work tasks? *arXiv preprint
arXiv:2403.07718*, 2024.512 Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew
513 Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended em-
514 bodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information
515 Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=rc8o_j8I8PX.516 Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen
517 Zhang, Peiyi Wang, Xiangwu Guo, et al. Assistgui: Task-oriented desktop graphical user interface
518 automation. *arXiv preprint arXiv:2312.13108*, 2023.519 Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and
520 Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents.
521 *arXiv preprint arXiv:2410.05243*, 2024.522 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-
523 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint
arXiv:2410.21276*, 2024.524 Jun Kong, Keven L Ates, Kang Zhang, and Yan Gu. Adaptive mobile interfaces through grammar
525 induction. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*,
526 volume 1, pp. 133–140. IEEE, 2008.527 Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots
528 into html code with the websight dataset, 2024.529 Kevin Qinghong Lin, Linjie Li, Difei Gao, Qinchen Wu, Mingyi Yan, Zhengyuan Yang, Lijuan
530 Wang, and Mike Zheng Shou. Videogui: A benchmark for gui automation from instructional
531 videos. *arXiv preprint arXiv:2406.10227*, 4, 2024a.

540 Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Zechen Bai, Weixian Lei, Lijuan Wang,
 541 and Mike Zheng Shou. Showui: One vision-language-action model for generalist gui agent. In
 542 *NeurIPS 2024 Workshop on Open-World Agents*, volume 1, 2024b.

543

544 Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: End-to-end policy opti-
 545 mization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*, 2025.

546

547 Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based
 548 gui agent. *arXiv preprint arXiv:2408.00203*, 2024.

549

550 Yuwen Lu, Ziang Tong, Qinyi Zhao, Chengzhi Zhang, and Toby Jia-Jun Li. Ui layout generation
 551 with llms guided by ui grammar. *arXiv preprint arXiv:2310.15455*, 2023.

552

553 Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia:
 554 a benchmark for general ai assistants. In *The Twelfth International Conference on Learning
 555 Representations*, 2023.

556

557 OpenAI. Introducing gpt-5. OpenAI website, 2025. Available at <https://openai.com/index/introducing-gpt-5/>, accessed August 10, 2025.

558

559 openai. Operator, 2025. URL <https://openai.com/index/introducing-operator/>.

560

561 Xavi Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Ruslan Partsey, Jimmy Yang,
 562 Ruta Desai, Alexander William Clegg, Michal Hlavac, Tiffany Min, Theo Gervet, Vladimir Von-
 563 drus, Vincent-Pierre Berges, John Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakr-
 564 ishan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and
 565 Roozbeh Mottaghi. Habitat 3.0: A co-habitat for humans, avatars and robots, 2023.

566

567 Qwen. Qwen-3-coder. <https://qwenlm.github.io/blog/qwen3-coder>, 2025.

568

569 Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro,
 570 Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can
 571 teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–
 572 68551, 2023.

573

574 ByteDance Seed. Ui-tars-1.5. <https://seed-tars.com/1.5>, 2025.

575

576 Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi,
 577 Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions
 578 for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern
 579 recognition*, pp. 10740–10749, 2020.

580

581 Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code:
 582 Benchmarking multimodal code generation for automated front-end engineering. *arXiv preprint
 583 arXiv:2403.03163*, 2024.

584

585 Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu,
 586 and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv
 587 preprint arXiv:2412.04454*, 2024.

588

589 Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark
 590 prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*,
 591 2023.

592

593 Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web
 594 agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.

595

596 Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,
 597 Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic
 598 web environment for building autonomous agents, 2024. URL <https://arxiv.org/abs/2307.13854>.

599

600

594	A Experiment Results	12																																																																																										
595																																																																																												
596																																																																																												
597	B Agent’s Prompts	13																																																																																										
598																																																																																												
599	C Experiment Details	20																																																																																										
600																																																																																												
601	D Full Statistics and Examples	21																																																																																										
602																																																																																												
603																																																																																												
604	E Qualitative Analysis	21																																																																																										
605																																																																																												
606	F The Use of Large Language Models	23																																																																																										
607																																																																																												
608	A EXPERIMENT RESULTS																																																																																											
609																																																																																												
610	Below, we show the experiment results additional to the results showcased in the main paper.																																																																																											
611																																																																																												
612																																																																																												
613	Table 4: Main results per model (Operator as CUA): CUA Success Rate (Function Completeness Rate).																																																																																											
614																																																																																												
615																																																																																												
616	<table border="1"> <thead> <tr> <th>Model</th> <th>Version</th> <th>landing</th> <th>game</th> <th>app</th> <th>utility</th> <th>interactive</th> <th>tool</th> <th>overall</th> </tr> </thead> <tbody> <tr> <td>GPT-5</td> <td>Baseline</td> <td>34.7% (53.0%)</td> <td>24.8% (77.8%)</td> <td>27.3% (70.6%)</td> <td>14.4% (63.3%)</td> <td>18.1% (73.0%)</td> <td>21.9% (70.0%)</td> <td>24.5% (67.9%)</td> </tr> <tr> <td></td> <td>+ Integrated</td> <td>41.3% (75.3%)</td> <td>42.6% (92.2%)</td> <td>38.8% (85.2%)</td> <td>27.8% (73.3%)</td> <td>10.7% (82.6%)</td> <td>21.4% (76.7%)</td> <td>31.5% (81.5%)</td> </tr> <tr> <td>Qwen3-</td> <td>Baseline</td> <td>5.3% (16.3%)</td> <td>9.3% (50.4%)</td> <td>9.1% (41.2%)</td> <td>11.7% (43.9%)</td> <td>7.0% (52.2%)</td> <td>1.4% (54.8%)</td> <td>7.3% (42.1%)</td> </tr> <tr> <td>Coder-30B</td> <td>+ Integrated</td> <td>10.0% (47.0%)</td> <td>27.0% (68.9%)</td> <td>19.1% (60.3%)</td> <td>20.6% (55.6%)</td> <td>13.7% (57.4%)</td> <td>23.8% (62.9%)</td> <td>18.6% (58.5%)</td> </tr> <tr> <td>GPT-4o</td> <td>Baseline</td> <td>4.7% (9.7%)</td> <td>12.6% (55.2%)</td> <td>12.4% (36.1%)</td> <td>6.7% (38.9%)</td> <td>9.3% (44.8%)</td> <td>5.7% (37.6%)</td> <td>8.8% (36.3%)</td> </tr> <tr> <td></td> <td>+ Integrated</td> <td>15.7% (19.0%)</td> <td>35.9% (59.3%)</td> <td>14.5% (44.5%)</td> <td>15.0% (47.8%)</td> <td>5.9% (50.7%)</td> <td>13.8% (46.2%)</td> <td>16.9% (43.8%)</td> </tr> </tbody> </table>	Model	Version	landing	game	app	utility	interactive	tool	overall	GPT-5	Baseline	34.7% (53.0%)	24.8% (77.8%)	27.3% (70.6%)	14.4% (63.3%)	18.1% (73.0%)	21.9% (70.0%)	24.5% (67.9%)		+ Integrated	41.3% (75.3%)	42.6% (92.2%)	38.8% (85.2%)	27.8% (73.3%)	10.7% (82.6%)	21.4% (76.7%)	31.5% (81.5%)	Qwen3-	Baseline	5.3% (16.3%)	9.3% (50.4%)	9.1% (41.2%)	11.7% (43.9%)	7.0% (52.2%)	1.4% (54.8%)	7.3% (42.1%)	Coder-30B	+ Integrated	10.0% (47.0%)	27.0% (68.9%)	19.1% (60.3%)	20.6% (55.6%)	13.7% (57.4%)	23.8% (62.9%)	18.6% (58.5%)	GPT-4o	Baseline	4.7% (9.7%)	12.6% (55.2%)	12.4% (36.1%)	6.7% (38.9%)	9.3% (44.8%)	5.7% (37.6%)	8.8% (36.3%)		+ Integrated	15.7% (19.0%)	35.9% (59.3%)	14.5% (44.5%)	15.0% (47.8%)	5.9% (50.7%)	13.8% (46.2%)	16.9% (43.8%)																												
Model	Version	landing	game	app	utility	interactive	tool	overall																																																																																				
GPT-5	Baseline	34.7% (53.0%)	24.8% (77.8%)	27.3% (70.6%)	14.4% (63.3%)	18.1% (73.0%)	21.9% (70.0%)	24.5% (67.9%)																																																																																				
	+ Integrated	41.3% (75.3%)	42.6% (92.2%)	38.8% (85.2%)	27.8% (73.3%)	10.7% (82.6%)	21.4% (76.7%)	31.5% (81.5%)																																																																																				
Qwen3-	Baseline	5.3% (16.3%)	9.3% (50.4%)	9.1% (41.2%)	11.7% (43.9%)	7.0% (52.2%)	1.4% (54.8%)	7.3% (42.1%)																																																																																				
Coder-30B	+ Integrated	10.0% (47.0%)	27.0% (68.9%)	19.1% (60.3%)	20.6% (55.6%)	13.7% (57.4%)	23.8% (62.9%)	18.6% (58.5%)																																																																																				
GPT-4o	Baseline	4.7% (9.7%)	12.6% (55.2%)	12.4% (36.1%)	6.7% (38.9%)	9.3% (44.8%)	5.7% (37.6%)	8.8% (36.3%)																																																																																				
	+ Integrated	15.7% (19.0%)	35.9% (59.3%)	14.5% (44.5%)	15.0% (47.8%)	5.9% (50.7%)	13.8% (46.2%)	16.9% (43.8%)																																																																																				
617																																																																																												
618																																																																																												
619																																																																																												
620																																																																																												
621																																																																																												
622																																																																																												
623	As shown in the Table 4, when using operator as CUA policy for integrated revision, consistent																																																																																											
624	improvements for both function completeness and CUA success rate can be observed. Moreover,																																																																																											
625	compared to the CUA success rate showcased in Table 3, it can be observed that Operator has higher																																																																																											
626	CUA success rate than UI-TARS in hard domains such as game and app that requires responsive and																																																																																											
627	complex interactions, showcasing its strong navigation capability.																																																																																											
628	Table 5: Commenter Ablations per model: CUA Success Rate (Function Completeness Rate).																																																																																											
629																																																																																												
630	<table border="1"> <thead> <tr> <th>Model</th> <th>Variant</th> <th>landing</th> <th>game</th> <th>app</th> <th>utility</th> <th>interactive</th> <th>tool</th> <th>overall</th> </tr> </thead> <tbody> <tr> <td>GPT-5</td> <td>Text-only</td> <td>24.0% (50.7%)</td> <td>31.1% (87.8%)</td> <td>21.2% (69.4%)</td> <td>16.1% (55.6%)</td> <td>8.9% (59.3%)</td> <td>6.2% (43.3%)</td> <td>18.7% (62.1%)</td> </tr> <tr> <td></td> <td>Screenshot-only</td> <td>17.3% (30.3%)</td> <td>16.7% (65.6%)</td> <td>12.4% (42.7%)</td> <td>15.6% (38.3%)</td> <td>5.2% (27.8%)</td> <td>9.5% (46.7%)</td> <td>12.8% (41.7%)</td> </tr> <tr> <td></td> <td>Dashboard</td> <td>17.7% (53.3%)</td> <td>43.3% (87.8%)</td> <td>30.0% (74.2%)</td> <td>21.1% (70.0%)</td> <td>21.1% (70.4%)</td> <td>17.6% (69.5%)</td> <td>25.7% (70.8%)</td> </tr> <tr> <td>Qwen3-</td> <td>Text-only</td> <td>8.0% (18.3%)</td> <td>20.7% (61.9%)</td> <td>7.3% (42.4%)</td> <td>8.3% (54.4%)</td> <td>10.7% (48.9%)</td> <td>16.2% (57.1%)</td> <td>11.7% (45.6%)</td> </tr> <tr> <td>Coder-30B</td> <td>Screenshot-only</td> <td>9.3% (20.7%)</td> <td>11.9% (63.7%)</td> <td>5.2% (34.5%)</td> <td>10.6% (40.6%)</td> <td>7.4% (55.9%)</td> <td>5.2% (37.6%)</td> <td>8.1% (41.7%)</td> </tr> <tr> <td></td> <td>Dashboard</td> <td>6.7% (23.3%)</td> <td>20.7% (50.4%)</td> <td>9.1% (38.8%)</td> <td>11.1% (49.4%)</td> <td>12.2% (39.3%)</td> <td>11.4% (55.2%)</td> <td>11.7% (41.3%)</td> </tr> <tr> <td>GPT-4o</td> <td>Text-only</td> <td>7.7% (13.0%)</td> <td>14.8% (57.0%)</td> <td>12.7% (34.8%)</td> <td>2.8% (37.8%)</td> <td>15.9% (39.3%)</td> <td>7.6% (29.0%)</td> <td>10.8% (34.8%)</td> </tr> <tr> <td></td> <td>Screenshot-only</td> <td>4.7% (10.3%)</td> <td>15.6% (43.7%)</td> <td>10.6% (31.2%)</td> <td>6.1% (45.6%)</td> <td>5.6% (34.8%)</td> <td>7.1% (37.6%)</td> <td>8.5% (32.5%)</td> </tr> <tr> <td></td> <td>Dashboard</td> <td>5.7% (8.3%)</td> <td>31.5% (55.2%)</td> <td>10.0% (28.2%)</td> <td>8.3% (34.4%)</td> <td>10.4% (26.3%)</td> <td>6.7% (35.7%)</td> <td>12.3% (30.4%)</td> </tr> </tbody> </table>	Model	Variant	landing	game	app	utility	interactive	tool	overall	GPT-5	Text-only	24.0% (50.7%)	31.1% (87.8%)	21.2% (69.4%)	16.1% (55.6%)	8.9% (59.3%)	6.2% (43.3%)	18.7% (62.1%)		Screenshot-only	17.3% (30.3%)	16.7% (65.6%)	12.4% (42.7%)	15.6% (38.3%)	5.2% (27.8%)	9.5% (46.7%)	12.8% (41.7%)		Dashboard	17.7% (53.3%)	43.3% (87.8%)	30.0% (74.2%)	21.1% (70.0%)	21.1% (70.4%)	17.6% (69.5%)	25.7% (70.8%)	Qwen3-	Text-only	8.0% (18.3%)	20.7% (61.9%)	7.3% (42.4%)	8.3% (54.4%)	10.7% (48.9%)	16.2% (57.1%)	11.7% (45.6%)	Coder-30B	Screenshot-only	9.3% (20.7%)	11.9% (63.7%)	5.2% (34.5%)	10.6% (40.6%)	7.4% (55.9%)	5.2% (37.6%)	8.1% (41.7%)		Dashboard	6.7% (23.3%)	20.7% (50.4%)	9.1% (38.8%)	11.1% (49.4%)	12.2% (39.3%)	11.4% (55.2%)	11.7% (41.3%)	GPT-4o	Text-only	7.7% (13.0%)	14.8% (57.0%)	12.7% (34.8%)	2.8% (37.8%)	15.9% (39.3%)	7.6% (29.0%)	10.8% (34.8%)		Screenshot-only	4.7% (10.3%)	15.6% (43.7%)	10.6% (31.2%)	6.1% (45.6%)	5.6% (34.8%)	7.1% (37.6%)	8.5% (32.5%)		Dashboard	5.7% (8.3%)	31.5% (55.2%)	10.0% (28.2%)	8.3% (34.4%)	10.4% (26.3%)	6.7% (35.7%)	12.3% (30.4%)	
Model	Variant	landing	game	app	utility	interactive	tool	overall																																																																																				
GPT-5	Text-only	24.0% (50.7%)	31.1% (87.8%)	21.2% (69.4%)	16.1% (55.6%)	8.9% (59.3%)	6.2% (43.3%)	18.7% (62.1%)																																																																																				
	Screenshot-only	17.3% (30.3%)	16.7% (65.6%)	12.4% (42.7%)	15.6% (38.3%)	5.2% (27.8%)	9.5% (46.7%)	12.8% (41.7%)																																																																																				
	Dashboard	17.7% (53.3%)	43.3% (87.8%)	30.0% (74.2%)	21.1% (70.0%)	21.1% (70.4%)	17.6% (69.5%)	25.7% (70.8%)																																																																																				
Qwen3-	Text-only	8.0% (18.3%)	20.7% (61.9%)	7.3% (42.4%)	8.3% (54.4%)	10.7% (48.9%)	16.2% (57.1%)	11.7% (45.6%)																																																																																				
Coder-30B	Screenshot-only	9.3% (20.7%)	11.9% (63.7%)	5.2% (34.5%)	10.6% (40.6%)	7.4% (55.9%)	5.2% (37.6%)	8.1% (41.7%)																																																																																				
	Dashboard	6.7% (23.3%)	20.7% (50.4%)	9.1% (38.8%)	11.1% (49.4%)	12.2% (39.3%)	11.4% (55.2%)	11.7% (41.3%)																																																																																				
GPT-4o	Text-only	7.7% (13.0%)	14.8% (57.0%)	12.7% (34.8%)	2.8% (37.8%)	15.9% (39.3%)	7.6% (29.0%)	10.8% (34.8%)																																																																																				
	Screenshot-only	4.7% (10.3%)	15.6% (43.7%)	10.6% (31.2%)	6.1% (45.6%)	5.6% (34.8%)	7.1% (37.6%)	8.5% (32.5%)																																																																																				
	Dashboard	5.7% (8.3%)	31.5% (55.2%)	10.0% (28.2%)	8.3% (34.4%)	10.4% (26.3%)	6.7% (35.7%)	12.3% (30.4%)																																																																																				
631																																																																																												
632																																																																																												
633																																																																																												
634																																																																																												
635																																																																																												
636																																																																																												
637																																																																																												
638	Table 5 demonstrates the results when using different types of commenters for revision based on																																																																																											
639	CUA navigation feedback. From the results, it can be inferred that dashboard is capable to provide																																																																																											
640	comprehensive visual and textual cues derived from the CUA policy trajectories, but requiring the																																																																																											
641	commenter to have strong visual perception.																																																																																											
642	Table 6 demonstrates that why VLM evaluation on CUA task trajectory is unreliable. It can be																																																																																											
643	observed that the the compared to rule-based oracle evaluation, the VLM evalation tends to judge the																																																																																											
644	CUA policy outcome as failure, thus having very low balanced accuracy, recall and F1. Moreover,																																																																																											
645	the low Cohen’s κ indicates very weak agreement of VLM evaluation compared to rule-based oracle																																																																																											
646	evaluation. This indicates that VLM evaluation on the final screenshot only is unreliable, and may																																																																																											
647	requires more screenshots along the CUA policy task trajectory for more reliable evaluation, leading																																																																																											
	to much higher computational cost.																																																																																											

648
 649
 650
 651
 652 **Table 6: VLM evaluation on final screenshot only is unreliable.** Given the final screenshot of
 653 CUA trajectory and the expected outcome, the accuracy of VLM evaluation is only slightly above
 654 the naive all-fail baseline; Balanced Accuracy is near 0.55; Recall/F1 and Cohen’s κ are low.
 655
 656

Metric	Overall	GPT-5	Qwen2.5-VL-72B	GPT-4o
Naive all-fail baseline accuracy	0.720	0.720	0.720	0.720
Accuracy vs. oracle	0.735	0.736	0.738	0.732
Balanced accuracy	0.556	0.549	0.568	0.552
Precision (Pass)	0.616	0.660	0.612	0.589
Recall (Pass)	0.147	0.121	0.178	0.142
F1 (Pass)	0.237	0.205	0.276	0.229
Cohen’s κ	0.145	0.128	0.175	0.132

661
 662 **B AGENT’S PROMPTS**
 663
 664

665 **Task Proposer Prompt.** Below, we show the prompt used by the Task Proposer.
 666

667 **Task Proposer Prompt**

668 Propose a comprehensive set of 30 diverse, realistic user tasks for the following {tag-type} application:
 669

670 Application: {app_title}
 671 Description: {app_description}

672 Each task should be:

- 673 - Clear and specific in its description
- 674 - Represent realistic user scenarios
- 675 - Cover different complexity levels and use cases
- 676 - Grounded in an observable outcome: The task’s completion must be marked by a clear and unambiguous
 677 change in the application’s state or interface. The expected outcome description must precisely define
 678 this terminal state.
- 679 - Avoid single element grounding (focus on complete workflows)
- 680 - Test the application’s core functionality effectively

681 {tag_specific_content}

682 Tag Philosophy Template:
 683

684 game:

685 Focus on GAME-SPECIFIC user tasks:

- 686 1. Playing complete game rounds or levels
- 687 2. Achieving high scores and personal bests
- 688 3. Completing specific game objectives or challenges
- 689 4. Using game controls and input methods
- 690 5. Navigating game menus and settings
- 691 6. Restarting games and trying different strategies
- 692 7. Progressing through difficulty levels

693 Additional task requirements:

- 694 - Focus on actual gameplay actions and goals
- 695 - Include winning and losing scenarios
- 696 - Cover different skill levels and strategies
- 697 - Test game restart and replay functionality
- 698 - Emphasize user enjoyment and engagement

699 tool:

700 Focus on TOOL-SPECIFIC user tasks:

- 701 1. Creating or generating content using the tool
- 702 2. Inputting data in various formats and types (typed/pasted text or on-page controls)

702 | 3. Transforming and processing information
703 | 4. Previewing results in-page (no file uploads/downloads)
704 | 5. Using tool-specific features and options
705 | 6. Working with both simple and complex inputs
706 | 7. Completing end-to-end workflows within the page

Additional task requirements:

- Focus on practical use cases and workflows
- Include both basic and advanced tool usage
- Cover different input types and scenarios without external files
- Verify visible in-page outputs or status changes in the DOM
- Emphasize real-world problem solving

- utility:
- Focus on **UTILITY-SPECIFIC** user tasks:
 1. Setting up and configuring the utility for...
 2. Adding, organizing, and managing data...
 3. Tracking progress and monitoring status...
 4. Using timers, reminders, and scheduling...
 5. Customizing settings and preferences...
 6. Completing daily or routine activities...
 7. Accessing and updating information quo...

Additional task requirements:

- Focus on everyday productivity scenarios
- Include setup and personalization tasks
- Cover routine and habitual usage patterns
- Test organization and tracking features
- Emphasize practical daily life applications

interactive:

Focus on INTERACTIVE-SPECIFIC user tasks:

1. Exploring and experimenting with interactive elements
2. Creating and manipulating visual or audio content
3. Adjusting parameters and settings in real-time
4. Playing with creative tools and features
5. Experiencing immersive visual or audio effects
6. Using touch, click, and gesture interactions
7. Customizing appearance and behavior

Additional task requirements:

- Focus on creative and exploratory activities
- Include experimentation and play scenarios
- Cover different interaction methods
- Test customization and personalization
- Emphasize sensory and aesthetic experiences

landing:

Focus on LANDING-SPECIFIC user tasks:

1. Browsing and exploring page content and sections
2. Reading and understanding key information
3. Clicking on call-to-action buttons and links
4. Navigating through different page sections
5. Finding contact information and ways to engage
6. Viewing team, product, or service details
7. Accessing additional resources and links

Additional task requirements:

- Focus on visitor browsing and exploration
- Include information-seeking behaviors
- Cover engagement and conversion actions
- Test navigation and content discovery
- Emphasize typical visitor journey scenarios

```

756
757     app (default/other):
758     Focus on APP-SPECIFIC user tasks:
759     1. Creating, editing, and managing content or data
760     2. Using multiple features in combination
761     3. Setting up and personalizing the application
762     4. Completing complex multi-step workflows
763     5. Organizing and categorizing information
764     6. Accessing and updating saved information
765
766     Additional task requirements:
767     - Focus on practical in-app usage
768     - Include multi-feature workflows and combinations
769     - Cover content creation and management
770     - Test personalization and customization
771     - Verify completion via visible state changes in the DOM (no external integrations)
772
773     Task Categorization Framework:
774     Each task must be categorized into one of the following three archetypes, which provides a structured
775     approach to evaluating different facets of the application's functionality:
776     - "core_function": Tests a single, primary feature in isolation.
777     - "user_workflow": Tests a sequence of features that represent a complete user goal.
778     - "edge_case": Tests non-standard inputs, boundary conditions, or less common interaction patterns.
779
780     Please respond in JSON format:
781
782     {
783         "app_name": "<app_name>",
784         "tags": ["<tag1>", "..."],
785         "tasks": [
786             {
787                 "id": 1,
788                 "description": "Clear, specific task description",
789                 "category": "core_function|user_workflow|edge_case",
790                 "expected_outcome": "What should happen when task completes"
791             }
792         ]
793     }

```

789 **Coder Prompt.** Below, we show the prompt used by the Coder.

791 Coder Prompt

793 [Initial Website Generation]

795 Create a single-page web application based on the following specification:

796 {instruction}

798 Requirements:

- 799 1. Create a complete HTML file with embedded CSS and JavaScript
- 800 2. The app should be fully functional and interactive
- 801 3. Use modern HTML5, CSS3, and vanilla JavaScript (no external libraries)
- 802 4. Include proper semantic HTML structure
- 803 5. Make the UI clean, responsive, and user-friendly
- 804 6. Add unique IDs to interactive elements for easier automation testing
- 805 7. Ensure the app works in a 1280x720 viewport

806 Please generate the complete HTML file:

808 [Revision from CUA Failures — Core Prompt]

```

810
811 You are tasked with improving a web application based on detailed failure analysis from auto-
812 mated testing.
813
814 ## CONTEXT
815 Application: {app_name}
816 Model: {model_name}
817 Total Failed Tasks: {len(failed_tasks)}
818 Failure Categories: {list(failure_categories.keys())}
819 Original HTML Length: {len(initial_html.strip())}
820
821 ## OUTPUT FORMAT
822 Generate a single, complete, and self-contained HTML file. The file must be fully functional, including
823 all necessary CSS and JavaScript, from 'DOCTYPE html' to '/html'. Do not use placeholders or truncate
824 the code.
825
826 ## ORIGINAL INITIAL WEBSITE (FULL)
827
828 {initial_html}
829
830 ## COMMENTER UI ANALYSIS
831 {(failure_analysis or "No visual UI analysis available").strip()}
832
833 {(non_regression_contract_prompt or "").strip()}
834
835 ## IMPROVEMENT REQUIREMENTS
836
837 ### 1. Core Issues to Address
838 Based on the failure analysis, you must:
839 - Identify missing DOM elements that tasks expect to exist
840 - Add missing JavaScript functionality for user interactions
841 - Fix timing issues that prevent task completion
842 - Ensure proper event handling and state management
843 - Add missing visual feedback and UI updates
844
845 ### 2. Specific Fixes Needed
846 For each failed task category:
847 - basic_usage: Ensure fundamental interactions work (clicking, displaying, updating)
848 - workflow: Support complete user workflows and multi-step processes
849 - advanced_feature: Implement sophisticated UI behaviors and animations
850 - edge_case: Handle unusual inputs and boundary conditions properly
851
852 ### 3. Technical Implementation Guidelines
853 - Preserve ALL existing working functionality from initial version
854 - Add missing HTML elements with unique IDs for automation
855 - Implement complete JavaScript event handlers and state updates
856 - Ensure synchronous UI updates for immediate feedback
857 - Do NOT introduce new input constraints that would block task inputs implied by the tasks (e.g., accept
858 plain text or non-HTTP payloads if tasks need them). Validation must be permissive and never reduce
859 what the initial version allowed.
860 - Do NOT auto-trigger flows on page load that would change initial states relied upon by tasks (e.g.,
861 auto-generation, auto-download, auto-navigation). Initial state should be neutral and idle.
862 - Keep critical controls visible within a 1280x720 viewport without scrolling. Avoid multi-panel "hub"
863 layouts; prefer single-view, compact layouts that fit important controls on screen.
864 - Avoid adding non-essential animations/transitions; prioritize high visibility and clarity over decoration.
865 - Make sure timers, counters, and dynamic content work correctly
866
867 ### 4. DOM Structure Requirements
868 - Every interactive element MUST have a unique ID
869 - Form controls must have proper event listeners
870 - Dynamic content areas must update immediately on state changes
871 - Visual feedback must be implemented for all user actions
872
873 ### 5. JavaScript Functionality Requirements

```

864 - All user interactions mentioned in failed tasks must be fully implemented
 865 - State changes must be reflected in the DOM immediately
 866 - Event handlers must properly update all related UI elements
 867 - Any game logic, scoring, timing must be complete and functional
 868

869 **Surgical Revision Policy**
 870 - Preserve existing IDs; do not rename or remove working elements from initial version.
 871 - Avoid large rewrites. Patch only the functions, event handlers, and minimal markup necessary to satisfy the failed/unsupported tasks.
 872 - Preserve working logic from initial version; do not regress features that already work.
 873 - Reuse existing elements/IDs for state wherever possible; only add new IDs if strictly necessary to expose the state of new logic.
 874 - Preserve initial version immediacy semantics. Do NOT introduce extra confirmation steps as prerequisites where initial version achieved completion via immediate interactions. Implement functional logic first, then expose proxies from the same code path; never update proxies without the underlying state change.
 875

876

877

878

879 **Commenter JSON (if provided)**
 880 - If the COMMENTER UI ANALYSIS is a JSON object, prioritize applying entries in ‘actionable_changes’ precisely.
 881 - Keep changes surgical and bounded by those actionable suggestions; do not broaden scope beyond them.
 882

883 **## OUTPUT REQUIREMENTS**
 884 Generate a COMPLETE, FULLY FUNCTIONAL HTML file that:
 885 1. Addresses ALL failure points identified in the analysis
 886 2. Maintains existing successful functionality from initial version
 887 3. Implements missing features causing task failures
 888 4. Provides proper DOM elements for automation testing
 889 5. Ensures immediate UI feedback for all user actions

890 **[Revision — Agent-Centric Design Principles]**

891 While improving functionality, apply the following design principles to optimize the UI for automated agents. The goal is functionality and testability, not human aesthetics.
 892

893

894 **### A. Visual Clarity and Simplicity**
 895 - Use a simple color scheme (e.g., black text on a white background).
 896 - Avoid decorative elements that do not serve a functional purpose, such as animations, gradients, or shadows.
 897 - Establish a clear visual hierarchy using typography and spacing. Logically group related controls.
 898

899 **### B. Robust Agent Interaction**
 900 - All interactive controls must be clearly labeled and sized appropriately to be easily and unambiguously targeted by automation tools.
 901 - Support keyboard-based interaction for all core functionality. Navigable elements should have clear focus indicators.
 902 - Prioritize immediate state updates upon interaction. Avoid complex, multi-step confirmation dialogs for actions where direct manipulation is sufficient.
 903 - All critical functionality should be accessible within a standard 1280x720 viewport without requiring scrolling.
 904

905

906

907 **### C. Transparent State Management**
 908 - The DOM must serve as a reliable, single source of truth for the application’s state.
 909 - Ensure that any significant state change (e.g., a result is generated, a calculation is complete) is clearly and synchronously reflected in the DOM. This can be achieved by updating element attributes, text content, or values.
 910 - Interactive elements and state indicators must have unique and stable IDs to facilitate reliable testing and interaction.
 911

912

913

914 **### D. Versatile Input Handling**
 915 - For continuous inputs (like sliders), provide alternative discrete control mechanisms (e.g., step buttons, direct text input). No interaction should rely solely on pointer-dragging.
 916 - Input validation should be permissive and should not block inputs that an automated task might

918 reasonably provide.
 919 - Distinguish between actions that cause immediate, reversible state changes (e.g., selecting an option)
 920 and those that trigger irreversible, multi-step processes (e.g., submitting a form).
 921
 922 **### E. Behavior Preservation**
 923 - Simplifying the visual design must not alter the core interaction logic.
 924 - Any user action that was immediate in initial version must remain immediate in the revised version.
 925
 926 Please generate the complete improved HTML file:
 927
 928 [Revision from Unsupported Tasks]
 929 You are tasked with improving a web application to support additional tasks that are currently un-
 930 supported.
 931
 932 **## CONTEXT**
 933 Application: {app_name}
 934 Model: {model_name}
 935 Total Unsupported Tasks: {len(unsupported_tasks)}
 936 Original HTML Length: {len(initial_html.strip())}
 937
 938 **## OUTPUT FORMAT**
 939 Generate a single, complete, and self-contained HTML file. The file must be fully functional, including
 940 all necessary CSS and JavaScript, from '!DOCTYPE html' to '/html'. Do not use placeholders or truncate
 941 the code.
 942
 943 **## ORIGINAL INITIAL WEBSITE (FULL)**
 944
 945 {initial_html}
 946
 947 **## UNSUPPORTED TASKS ANALYSIS**
 948 {unsupported_summary}
 949
 950 **## CODE PRESERVATION CONTRACT (Non-Regression)**
 951 {" if ablate_no_contract else (non_regression_contract_prompt or "").strip() }
 952
 953 **## IMPROVEMENT REQUIREMENTS**
 954
 955 **### 1. Task Support Issues to Address**
 956 Based on the unsupported task analysis, you must ADD missing functionality:
 957 - Add missing DOM elements that tasks expect to exist
 958 - Implement missing JavaScript functionality for user interactions
 959 - Add missing form controls and input handling
 960 - Implement missing display areas and visual feedback
 961 - Add missing navigation and UI components
 962
 963 **### 2. Implementation Guidelines**
 964 - PRESERVE all existing working functionality from initial version
 965 - ADD new HTML elements with unique IDs for automation
 966 - IMPLEMENT complete JavaScript event handlers for new features
 967 - ENSURE new UI elements are properly styled and visible
 968 - DO NOT introduce new input constraints that would block task inputs implied by tasks; validation must
 969 be permissive and must not reduce what the initial version allowed.
 970 - DO NOT auto-trigger flows on load that change initial states (no auto-generation, auto-download,
 971 auto-navigation). Start in a neutral, idle state.
 972 - FIT critical controls within a 1280x720 viewport without scrolling. Avoid multi-panel hub layouts and
 973 unnecessary panels that push controls below the fold.
 974 - IMPLEMENT missing workflows and user interaction patterns
 975
 976 **### 3. DOM Structure Requirements**
 977 - Every new interactive element MUST have a unique ID
 978 - New form controls must have proper event listeners
 979 - New content areas must update appropriately on state changes

```

972
973     - New visual feedback must be implemented for added interactions
974
975     ### 4. JavaScript Functionality Requirements
976     - All new user interactions mentioned in unsupported tasks must be fully implemented
977     - New state changes must be reflected in the DOM immediately
978     - New event handlers must properly update all related UI elements
979     - Any new game logic, scoring, timing must be complete and functional
980
981     ## OUTPUT REQUIREMENTS
982     Generate a complete and fully functional HTML file that:
983     1. Maintains all existing functionality from initial version.
984     2. Adds the missing functionality required to support the new tasks.
985     3. Implements all necessary DOM elements and JavaScript for task support.
986     4. Ensures all new features are robust and testable.
987
988     Commenter JSON (if provided)
989     - If upstream provides a commenter JSON analysis with 'actionable_changes', follow those changes first,
990     precisely and surgically.
991
992     Surgical Revision Policy
993     - Preserve existing IDs; do not rename or remove working elements from initial version.
994     - Avoid large rewrites. Patch only the functions, event handlers, and minimal markup necessary to satisfy
995     the failed/unsupported tasks.
996     - Preserve working logic from initial version; do not regress features that already work.
997     - Reuse existing elements/IDs for state wherever possible; only add new IDs if strictly necessary to
998     expose the state of new logic.
999     - Preserve initial version immediacy semantics. Do NOT introduce extra confirmation steps as prerequisites
1000    where initial version achieved completion via immediate interactions. Implement functional logic
1001    first, then expose proxies from the same code path; never update proxies without the underlying state
1002    change.
1003
1004     Please generate the complete improved HTML file:

```

CUA Policy Prompt. Below, we show the prompt used by the CUA Policy.

CUA Policy Prompt

You are a GUI agent. You are given a task and your action history, with screenshots. You need to perform the next action to complete the task.

Output Format

Thought: ...
 Action: ...

Action Space

```

click(point='x1 y1')
left_double(point='x1 y1')
right_single(point='x1 y1')
drag(start_point='x1 y1', end_point='x2 y2')
hotkey(key='ctrl c') # Split keys with a space and use lowercase. Also, do not use more than 3 keys in
one hotkey action.
type(content='xxx') # Use escape characters '\', '\"', and '\n' in content part to ensure we can parse the
content in normal python string format. If you want to submit your input, use '\n' at the end of content.
scroll(point='x1 y1', direction='down or up or right or left') # Show more information on the 'direction'
side.
wait() #Sleep for 5s and take a screenshot to check for any changes.
finished(content='xxx') # Use escape characters '\', '\"', and '\n' in content part to ensure we can parse the
content in normal python string format.

```

1026
 1027 ## Note
 1028 - Use {language} in 'Thought' part.
 1029 - Write a small plan and finally summarize your next action (with its target element) in one sentence in
 1030 'Thought' part.
 1031
 1032 ## User Instruction
 1033 {instruction}

1034
 1035 **Dashboard Commenter Prompt.** Below, we show the prompt used by the Dashboard Commenter.

1036 **Dashboard Commenter Prompt**

1038 You are diagnosing UI design issue that caused a task failure for a Computer-Use Agent (CUA). Your
 1039 goal is to conduct a root cause analysis based on a core set of design principles and output a structured
 1040 diagnostic report in JSON format. This report will guide the next iteration of UI code generation.

1041 You will be provided with two images:

1. The current website state (Resolution: {width}x{height})
2. A storyboard summarizing the failed task attempt, arranged as a grid of step screenshots (variable count) fitted into a 1920x1080 canvas

1046 Your analysis must be guided by the following Agent-Centric UI Design Principles:

1. State Visibility: Any significant state change resulting from an agent's action must be clearly and synchronously reflected in the DOM. This can be achieved by updating element attributes, text content, or values. Ambiguous or out-of-band feedback (like temporary toast notifications) is considered a violation.
2. Interaction Robustness: All UI components critical for task completion must be visible and actionable within a standard 1280x720 viewport without requiring scrolling. Elements should have clear, stable identifiers.
3. Input Permissiveness: Input fields and controls should accept the most general data format required for the task, avoiding overly restrictive client-side validation that may block agent inputs.
4. Predictable Behavior: The UI should remain in a stable, neutral state upon loading.

1056 Based on these principles, analyze the provided materials and output a compact JSON object.

1058 Output strictly as JSON with these keys only:

- issues: An array of up to 3 short strings identifying the primary UI problem categories, derived from the violated principles (e.g., "visibility", "interaction", "feedback").
- actionable_changes: An array of 3–6 diagnostic statements. Each statement must identify a specific UI element (referencing selectors/IDs) and explain which design principle it violated, providing a root cause for the failure. Example: "The element '#submit-btn' violates the Interaction Robustness principle, as it is not visible in the default viewport."
- fit_within_screen: A diagnostic boolean flag. Set to 'true' only if the primary reason for failure was a violation of the Interaction Robustness principle concerning viewport visibility.
- avoid_regressions: A confirmation flag, set to 'true', signifying that the diagnosis adheres to a "minimal intervention" philosophy. This confirms the analysis focuses solely on fixing the observed failure without disturbing unrelated, functional parts of the UI.

1069 Respond with JSON only, no extra text.

1073 **C EXPERIMENT DETAILS**

1075 For the GPT-5 experiments, we configured GPT-5 with high verbosity and high reasoning effort for
 1076 coding and low verbosity and minimal reasoning effort for commenting. For the Qwen experiments,
 1077 we used the Qwen3-Coder-30B-A3B-Instruct model for coding and Qwen2.5-VL-72B-Instruct for
 1078 commenting. Experiments that involve the CUA policy used UI-TARS-1.5-7B for the UI-TARS
 1079 CUA and OpenAI Computer-Use-Preview for the Operator CUA. The Task Proposer and the Verifier
 were both GPT-5 with high verbosity and high reasoning effort. In the CUA policy test we set the

maximum steps to 20 to prevent infinite loops and we ran the environment with Playwright. The CUA performed coordinate-based Computer Use actions only and did not interact with elements directly. This design makes the evaluation more challenging and more informative for UI design, since element layout and visibility become critical.

Effects by Refinement Round. As shown in the figure 8, iteratively applying revision can consistently bring improvements on the function completeness for all coders. Interestingly, it can be observed that the CUA success rate of GPT-5 coder may drop after repeated revision, while Qwen3-Coder-30B and GPT-4o can consistently gain from repeated revision. This indicates that the revision improvement may saturate for strong coders, but weak coders can be improved with iterative feedback and revisions.

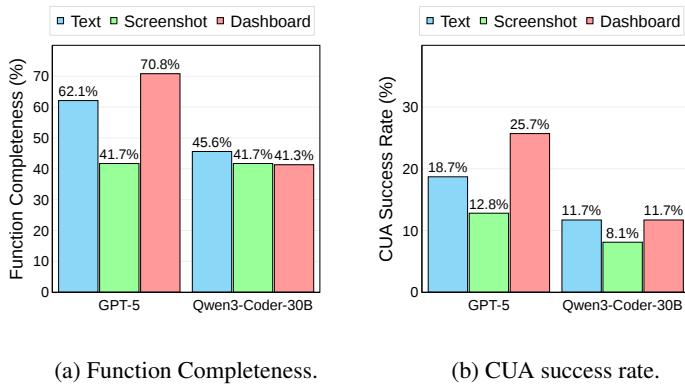


Figure 6: Ablation comparison of different commenters across two evaluation dimensions.

Figure 6 demonstrates the results when using different types of commenters for revision based on CUA navigation feedback. Besides dashboard commenter, there are two additional commenters which either use text only or screenshot only for CUA policy trajectory information. Text-only commenter will only perceive the same text information used in the dashboard, *i.e.*, task description, task expected outcome, CUA thought and action, etc. Screenshot-only commenter will only perceive the same screenshots used in the dashboard. Besides the CUA trajectory information, all the three commenters will see the full UI screenshot for UI analysis. It can be observed that dashboard can bring significant improvements for GPT-5 commenter and coder, but its performance is slightly lower than text-only commenter for Qwen commenter and coder. This indicates that dashboard offers comprehensive visual and textual cues for revision, which in turn requires strong visual perception to fully capture the information; meanwhile, a text-only commenter remains a reliable proxy for commenters with weaker perception.

D FULL STATISTICS AND EXAMPLES

Below, we display the full statistics and corresponding examples.

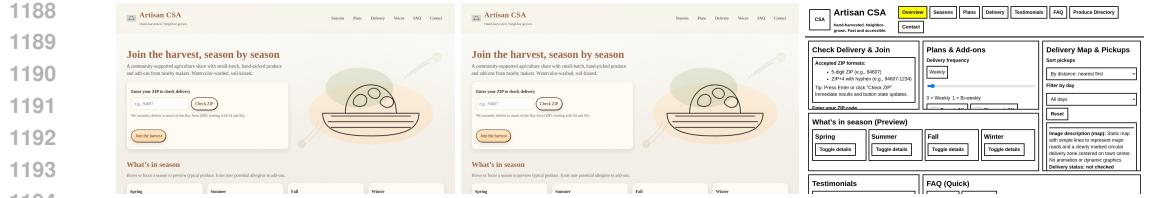
E QUALITATIVE ANALYSIS

In this section, we present four representative revision cases—*artisan-csa*, *color-match-challenge*, *csv-to-charts*, and *festival-lights-show*. Each row displays the initial UI alongside its revised versions, evaluated under two criteria: Function Test and CUA Test. Across the four cases, the revisions demonstrate distinct patterns of improvement. Revisions based on the Function Test, which addresses unsupported tasks, tend to focus on adding underlying functionality, sometimes with subtle visual changes. For example, the *festival-lights-show* revision added a crucial “Running” state indicator, and the *csv-to-charts* revision added a button to select a delimiter. In contrast, revisions based on the CUA Test consistently yield more significant visual modifications geared towards agent accessibility. For most websites, this meant adding buttons with clear boundaries and visual hints.

1134 Table 7: **Distribution and examples of six domains in AUI-Gym.** For each domain, we show a
 1135 website created by GPT-5, paired with 30 tasks (blue) simulating real-world usage. Each task is
 1136 further linked to a rule-based verifier (green).
 1137

1138 Domain	1139 #Apps	1140 Percentage	1141 Example Instruction	1142 GUI created by GPT-5
1140 App	11	21%	<p>Create a single-page app in a single HTML file with the following requirements:</p> <ul style="list-style-type: none"> - Name: Healthy Meal Tracker - Goal: Log meals and nutrition info. - Features: Ingredient list, calories per meal, daily summary. - The UI should be clean with food icons. <p>Task: Add five meals for today's date (any names/ingredients) so today's meal count reaches at least 5.</p> <p>Rule: #dailyMealCount >= 5</p>	
1146 Landing	10	19%	<p>Create a single-page app in a single HTML file with the following requirements:</p> <ul style="list-style-type: none"> - Name: Nonprofit Impact Report - Goal: Show measurable results of programs. - Features: Infographics, success stories, donation link. - The UI should be inspiring and visually engaging. <p>Task: Navigate to Success Stories and expand the first story card to reveal the full narrative.</p> <p>Rule: #slides .slide:first-child button[aria-expanded] == 'true' OR #slides .slide:first-child.expanded.exists</p>	
1154 Game	9	17%	<p>Create a single-page app in a single HTML file with the following requirements:</p> <ul style="list-style-type: none"> - Name: Typing Rain - Goal: Type falling words before they reach the bottom. - Features: Increasing difficulty, accuracy tracker, score. - The UI should be the city background with animated raindrop words. <p>Task: In a single run, achieve a score of at least 500 points.</p> <p>Rule: #scoreValue >= 500</p>	
1160 Interactive	9	17%	<p>Create a single-page app in a single HTML file with the following requirements:</p> <ul style="list-style-type: none"> - Name: Festival Lights Show - Goal: Control a virtual light show. - Features: Color changes, patterns, music sync. - The UI should be vibrant and dynamic. <p>Task: Enable Music Sync, start playback, then pause the built-in track; confirm audio status is Paused while Music Sync remains enabled.</p> <p>Rule: #audioStatus == 'Paused' AND #syncBadge != 'Sync: Off'</p>	
1168 Tool	7	13%	<p>Create a single-page app in a single HTML file with the following requirements:</p> <ul style="list-style-type: none"> - Name: Customer Journey Flow - Goal: Sketch customer journey stages and connections. - Features: Add/edit stages, connect nodes, view JSON of the flow. - The UI should be simple and full-screen. <p>Task: Create 'Social Ad' and 'Search Ad' leading to 'Landing Page', then to 'Consideration' and 'Purchase' (two branches merging into one path).</p> <p>Rule: #io-json contains 'Social Ad' AND #io-json contains 'Search Ad' AND #io-json contains 'Landing Page' AND #io-json contains 'Consideration' AND #io-json contains 'Purchase'</p>	
1178 Utility	6	12%	<p>Create a single-page app in a single HTML file with the following requirements:</p> <ul style="list-style-type: none"> - Name: Pomodoro - Goal: Time focus and break sessions. - Features: Focus/break modes, timers, basic controls. - The UI should be minimal and distraction-free. <p>Task: Start a short break and verify the mode label and starting time show a 5-minute break.</p> <p>Rule: #lblSession == 'Short Break' AND #lblTime contains '05:00'</p>	

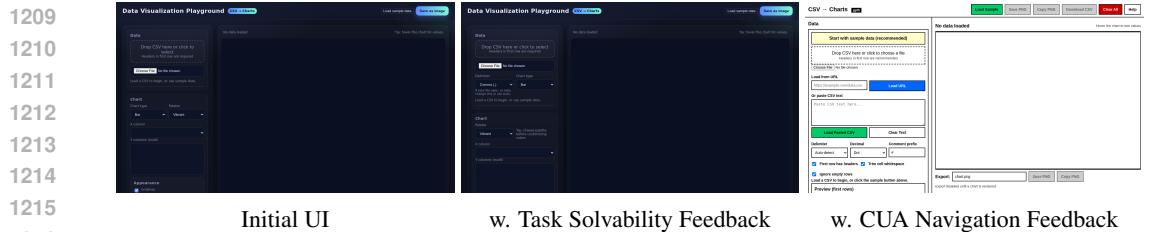
1184 In both *color-match-challenge* and *csv-to-charts*, both revision types improved accessibility by presenting more information and controls upfront, reducing the need for scrolling. A key CUA-friendly
 1185
 1186
 1187



1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
(a) **artisan-csa**: Create a single-page app, in a single HTML file, for a community-supported agriculture program with a hand-drawn, watercolor aesthetic.



1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
(b) **color-match-challenge**: Create a single-page app in a single HTML file for a fast-paced "color match" game. - Show a word (e.g., "RED") in a random font color — player must click the correct color button (not the word meaning). - Keep score based on correct answers within 30 seconds. - Use large typography, color-coded buttons, and smooth button press animations.



1209
1210
1211
1212
1213
1214
1215
1216
(c) **csv-to-charts**: Create a single-page app in a single HTML file with the following requirements: - Name: Data Visualization Playground - Goal: Upload CSV and generate charts. - Features: Chart type selector, color customization, save as image. - The UI should be modern with a focus on charts.



1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
(d) **festival-lights-show**: Create a single-page app in a single HTML file with the following requirements: - Name: Festival Lights Show - Goal: Control a virtual light show. - Features: Color changes, patterns, music sync. - The UI should be vibrant and dynamic.

1230
1231
1232
Figure 7: Qualitative comparison of UI v.s. AUI. Each row shows an initial UI (left), its revision based on function (middle), and its revision based on CUA-failed tasks (right).

1233
1234
adaptation is seen in *festival-lights-show*, where "increase" and "reduce" buttons were added as a
1235
1236
1237
1238
1239
1240
1241
complement to sliders, providing a more direct and reliable interaction method for automated agents.

F THE USE OF LARGE LANGUAGE MODELS

We used large language models (LLMs) in two limited ways. First, during manuscript preparation, LLMs were employed solely for surface-level editing (e.g., grammar correction and minor rephrasing) to improve readability; they were not used to generate research ideas, methods, experiments, or conclusions. Second, in our benchmark experiments, LLMs were included as baseline models for

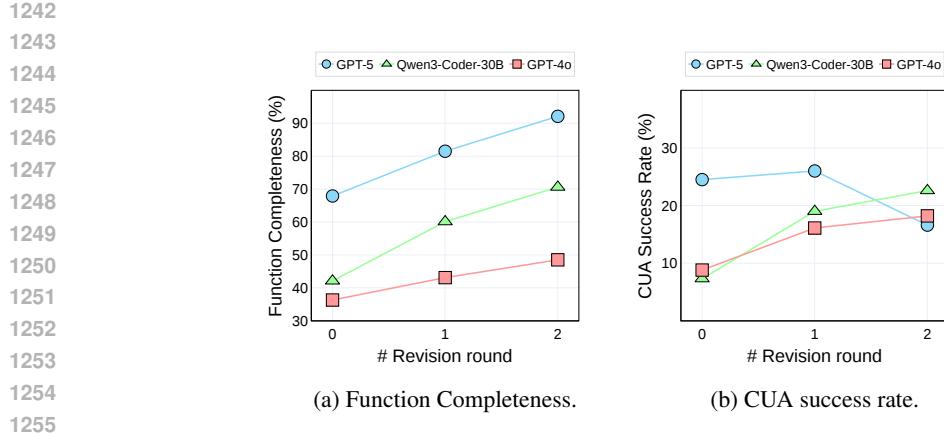


Figure 8: Effects of Revision round.