VISTA: A Versatile Interactive User Simulation Toolkit for Agent Evaluation

Anonymous ACL submission

Abstract

Evaluation remains a critical bottleneck for interactive agent development. Existing evaluation methods often rely on static benchmarks, which fail to capture the dynamic, multi-step nature of agentic behavior and struggle to expose meaningful failure modes. While user-simulation-based evaluation offers a promising alternative, existing simulation frameworks suffer from two major limitations. First, they provide limited mechanisms for evaluating the quality and comprehensiveness of simulated interactions, making it difficult to assess whether a simulator sufficiently explores an agent's capabilities and failure modes. Second, most frameworks are restricted to either UI-only actions or API-only actions, limiting their ability to model the full range of realistic user behaviors. To address these limitations, we propose *VISTA*, a **V**ersatile **I**nteractive user **S**imulation **T**oolkit for **A**gent evaluation. Our toolkit includes a suite of six metrics for measuring the realism, capability coverage, and interaction effectiveness of simulated interactions. In addition, we develop a hybrid user simulator that integrates both UI-based interactions and API-based interactions, enabling more realistic and comprehensive evaluation across diverse interactive environments. We evaluate *VISTA* in e-commerce shopping and education customer service settings and demonstrate that it produces more realistic and comprehensive evaluations than existing methods.[1]

1 Introduction

Interactive agents are increasingly deployed in real-world settings such as finance, education, and e-commerce (Dong et al., 2025; Chu et al., 2026; Amazon, 2026). As these systems become more capable, evaluation has emerged as a critical development challenge. Existing evaluation approaches often rely on static benchmarks (Yehudai et al., 2026; Mohammadi et al., 2025; Liu et al., 2025; Zhou et al., 2024), which measure performance on isolated tasks and overlook failure modes that arise during open-ended interactions such as poor action selection or tool-use breakdowns. As a result, there remains a growing gap between benchmark performance and real-world agent reliability.

User-simulation-based evaluation has therefore emerged as a promising direction for scalable agent assessment. By using AI agents to emulate user behavior, simulators can generate multi-turn interactions without requiring extensive human participation (Dou et al., 2025; Wang et al., 2025; Yao et al., 2024; Huang et al., 2024). However, existing simulation frameworks provide limited mechanisms for evaluating the quality of the simulated interactions themselves, making it difficult to determine whether a simulator meaningfully exercises an agent's capabilities. Prior work mainly focuses on assessing interaction fidelity (Zhu et al., 2026; Zhou et al., 2026) and offers little insight into whether the generated interactions sufficiently explore the range of behaviors, capabilities, and failure modes relevant to the product agent.

In addition to limitations in evaluation methodology, existing simulation frameworks are constrained in the types of interactions they support. Most existing approaches support UI-based interactions (Lu et al., 2025; Sun et al., 2025) or API-based actions (Barres et al., 2025), but not both. UI-only simulators are often more error-prone and computationally expensive due to their need to handle noisy, long context information from raw web pages as opposed to short, structured information returned from API calls (Savadikar et al., 2026). In contrast, simulators that rely solely on API-based actions are constrained by a limited set of tools and therefore unable to perform the full range of interactions needed to comprehensively evaluate modern agents. These limitations prevent existing approaches from fully capturing realistic user

[1] Our code and simulator will be released upon acceptance.

behavior and lead to incomplete evaluations.

To address these limitations, we introduce *VISTA*, a **V**ersatile **I**nteractive user **S**imulation **T**oolkit for **A**gent evaluation. Our toolkit introduces a suite of metrics for evaluating simulation quality, including measures of realism, capability coverage, and failure identification. These metrics provide insight into how comprehensively the simulator explores the agent's functionalities and enable more rigorous and interpretable evaluation of interactive systems.

In addition, *VISTA* introduces a hybrid user simulator capable of performing both UI-based and API-based operations alongside natural language interaction. Unlike prior approaches restricted to a single interaction paradigm, our hybrid framework enables more realistic simulation of user behavior across complex interactive environments. By combining conversational interaction with both UI-based and API-based actions, the simulator can evaluate a broader range of agent behaviors, including navigation, tool use, and action execution across multi-step workflows (see Figure 1).

We apply *VISTA* in two representative domains: e-commerce shopping and education. Our experiments demonstrate that the proposed toolkit produces more realistic and comprehensive evaluations than existing approaches. Furthermore, our proposed metrics provide meaningful signals regarding simulator quality and interaction coverage. Across both domains, *VISTA* uncovers a broader range of agent failure modes and more effectively exercises agent capabilities, highlighting the importance of flexible hybrid simulation for interactive systems. Our contributions are as follows:

- We introduce *VISTA*, a flexible evaluation toolkit for interactive agents based on hybrid user simulation. Our toolkit includes a suite of six metrics for evaluating simulation quality, including measures of realism, capability coverage, and interaction effectiveness.
- We develop a hybrid user simulator that combines UI and API-based actions, enabling more realistic and comprehensive simulation of user behavior across environments.
- We empirically evaluate *VISTA* in e-commerce shopping and education customer service settings, demonstrating that it produces more realistic and comprehensive evaluations while uncovering a broader range of agent failures than existing approaches.

2 Related Work

User simulation has emerged as a scalable alternative to human evaluation for interactive agents. Previous work has explored this paradigm across a variety of domains. For example, Sun et al. (2025) investigate whether LLM agents can faithfully simulate customers interacting with shopping assistants, while Lu et al. (2025) introduce UXAgent for automated usability testing of web interfaces. In business-oriented settings, Shea et al. (2025) propose SAGE, a knowledge-grounded simulator that incorporates business logic and to generate more realistic customer interactions. More recently, Castelo et al. (2026) demonstrate how user simulators can support scalable A/B testing and experimentation workflows.

Recent work has also explored how to evaluate the quality of the user simulators themselves. Deng et al. (2024) introduce CheckPoint, a metric for measuring whether mobile agents reach key intermediate states during task execution, while Zhou et al. (2026) propose the User-Sim Index to measure how closely simulated users resemble real human behavior. Although these metrics provide important insights, they are often tied to specific benchmarks or focus on human-likeness and task-state verification. Our work introduces general-purpose metrics that evaluate simulated interactions along dimensions such as realism, capability coverage, and interaction effectiveness, enabling a more holistic assessment of simulator quality.

Recent benchmarks have moved beyond purely conversational settings and begun evaluating agents in environments that require executable actions. Mobile-Bench augments mobile UI interactions with APIs to improve evaluation efficiency (Deng et al., 2024) and UXAgent enables browser-based interaction for usability testing (Lu et al., 2025). τ^2-Bench studies dual-control environments where both users and agents interact with a shared environment through tools (Barres et al., 2025). These works demonstrate the importance of action-enabled evaluation, but existing typically emphasize only UI-based actions or API-based actions. Our work builds on these ideas by introducing a hybrid simulation toolkit that combines both paradigms, allowing us to leverage the realism and flexibility of UI interaction alongside the efficiency and reliability of API-based actions.

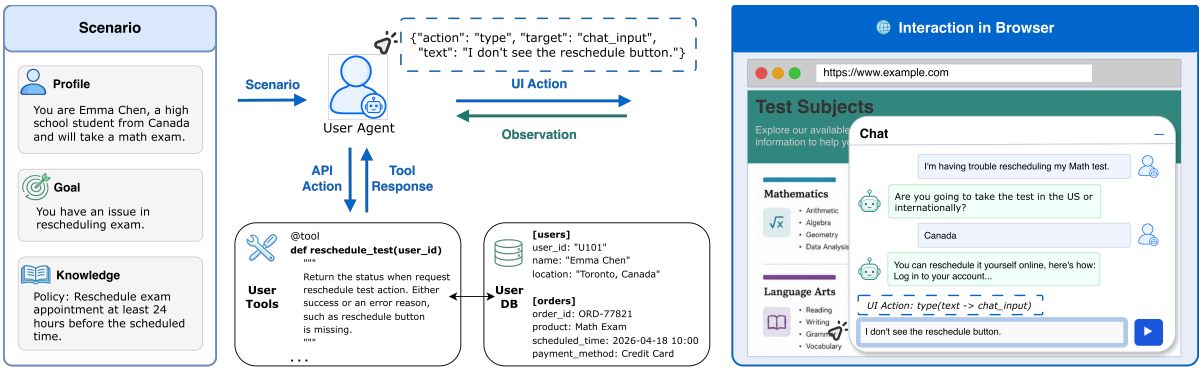

Figure 1: An overview of *VISTA*. It configures each task scenario with a user profile, user goal, and relevant domain knowledge. During interaction, the user agent observes the webpage and decides whether to take UI actions directly or invoke user-side API tools for additional task-grounded information. The resulting interactions are then evaluated in terms of capability coverage, utterance realism, and interaction effectiveness, measured by agent failures.

3 *VISTA*

We introduce *VISTA* as a flexible evaluation toolkit for interactive agents based on user simulation. Our toolkit includes a suite of evaluation metrics alongside a hybrid user simulator capable of taking both API-based actions and UI-based actions. We describe our evaluation metrics in Section 3.1 and our hybrid user simulator in Section 3.2.

3.1 Evaluation Metrics

VISTA includes six metrics spanning coverage, realism, cost, and failure identification. These metrics are described in detail in the following sections.

3.1.1 Coverage

Coverage measures whether the user agent can drive the product agent through diverse tool-use patterns, rather than repeatedly triggering similar execution paths. We evaluate coverage using three complementary metrics: tool-calling transition entropy, tool-calling distribution entropy, and trajectory distance.

Tool-calling Transition Entropy To measure the diversity of the product agent's tool-calling sequences, we propose *TransitionEntropy* (TE), which measures the normalized entropy of the tool-calling transition matrix. For each interaction trajectory, we extract adjacent tool-call pairs and aggregate them into an $N \times N$ transition matrix, where N is the number of agent tools. Each row defines the empirical distribution of the next tool conditioned on the current tool. We compute the Shannon entropy of each non-empty row, normalize it by $\log_2 N$, and average across valid rows:

$$\text{TransitionEntropy} = \frac{1}{|\mathcal{R}|} \sum_{i \in \mathcal{R}} \frac{-\sum_{j=1}^{N} p_{ij} \log_2 p_{ij}}{\log_2 N}$$

where p_{ij} is the probability of calling tool j after tool i, and $\mathcal{R}$ is the set of tools with at least one outgoing transition. The score ranges from 0 to 1. A low score indicates that the conversation follows a rigid, predictable conversation flow with less diversity, while a high score indicates flexible routing across different next-tool choices.

Tool-calling Distribution Entropy Transition entropy measures local routing diversity, but does not fully capture global tool coverage. Therefore, we additionally measure the tool-calling distribution entropy, *ToolDistrEntropy* (TDE), where q_i denotes the empirical frequency of tool i among all agents' tool calls. We compute the normalized Shannon entropy:

$$\text{ToolDistrEntropy} = \frac{-\sum_{i=1}^{N} q_i \log_2 q_i}{\log_2 N}$$

This score also ranges from 0 to 1. A score near 0 means that the agent relies on only one or a few tools, while a score near 1 means that tool calls are distributed evenly across the available tools. This metric complements transition entropy by capturing global tool coverage rather than sequential routing diversity.

Trajectory Distance Both transition entropy and distribution entropy measure aggregate tool-calling diversity on tool names across the full dataset. High scores indicate broad tool coverage and diverse tool transitions, but they do not rule out repeated trajectories under the same user goal. Such repetition reduces both diversity and efficiency. We propose *TrajDist* (TD) to measure within-goal trajectory diversity over tool names and tool arguments.

For each interaction trajectory, we represent the agent's tool-use behavior as a se-

quence of (`tool_name`, `input_repr`) pairs, where `input_repr` is a serialized representation of the tool arguments. Within each user goal group, we compute the average pairwise normalized Levenshtein distance between all trajectory pairs:

$$\text{TrajDist} = \frac{1}{N} \sum_{a<b} \frac{\text{Lev}(s_a, s_b)}{\max(|s_a|, |s_b|)}$$

where N is the number of pairs in the group, which is $\binom{|\mathcal{T}|}{2}$, s_a and s_b are two tool-call sequences, and $\text{Lev}(\cdot, \cdot)$ denotes the Levenshtein distance between two sequences.

The score ranges from 0 to 1. A score near 0 means that trajectories within the same user goal group follow nearly identical tool-call paths with similar arguments. A score near 1 means that the trajectories differ substantially in either tool choices, order, or arguments.

3.1.2 Realism

We evaluate the user simulator's realism with reference-free LLM-as-a-Judge scores. Following prior work, we assess whether each generated user utterance is consistent with three behavior components from its prompt: the user profile, task goal, and domain knowledge (Shea et al., 2025). The details of these components are in Section 3.2.1. Since the components are either manually derived from real interactions between customer-agent conversations or manually configured by domain experts, they provide realistic behavioral constraints. Therefore we report the corresponding consistency rates as automatic proxies for behavior realism. The judging prompt is provided in Appendix D.2.

3.1.3 Cost

We evaluate the efficiency of user simulator using two cost metrics. First, we use the average number of prompt and completion tokens per turn, which serves as a proxy for monetary cost. Second, we use the average number of actions per turn as a proxy for time cost. We use this proxy instead of direct elapsed time because wall-clock measurements can vary with implementation details, such as parallel execution and system load. Actions include both API calls and UI operations, each of which contributes to the time required to generate an interaction. Therefore, a simulator with more actions per turn generally incurs higher time cost.

3.1.4 Agent Failure Identification

We also assess the user agent's ability to identify agent failures. Following the prior work (Shea et al., 2025), we use an LLM-as-a-Judge to evaluate each assistant response along five dimensions: helpfulness, coherence, verbosity, relevance, and faithfulness. For each dimension, the judge returns a score from 0 to 4 with a brief justification. We pass the full conversation and the scores to a second LLM, which consolidates the justifications into a set of unique agent failures. We use the number of unique failures as the failure-identification metric. A higher count indicates broader ability to evaluate the agent's capabilities and failure cases.

3.2 Hybrid User Simulation

VISTA incorporates a hybrid user simulator that can take both API-based and UI-based actions. This allows the simulator to improve interaction quality by combining the increased flexibility of UI-based simulation with the enhanced reliability and cost effectiveness of API-based simulation. *VISTA* begins by constructing task scenarios, which define the context and objective of each simulated interaction (Section 3.2.1). It then equips the simulator with API-based and UI-based actions (Sections 3.2.2 and 3.2.3). Finally, it integrates API-based tool responses into UI action prediction through an iterative observation-planning-action loop (Section 3.2.4). An overview of the full process is provided in Figure 1.

3.2.1 Generate Scenarios

To produce realistic and diverse interactions, we construct each simulation scenario from three components: a user profile, a user goal, and relevant domain knowledge, following prior work (Shea et al., 2025). The user profile captures attributes that influence user behavior, including demographics, preferences, personality traits, and behavioral patterns. The user goal defines the task the simulated user is trying to complete, such as finding a product or requesting help with a course concept. We design these goals to cover the capabilities of the product agent. The domain knowledge defines the information context available to the user, such as webpages, FAQs, product catalogs, or other publicly available resources. We combine these three components in a structured template to construct the system prompt, which guides the user simulator throughout its interaction with the product agent. See Figure 1 for an example.

3.2.2 Create User API Tools

To make the user simulator reusable across domains while remaining closely aligned with specific product agents, we create two categories of user API tools: general tools and task-specific tools. Our test cases are specifically designed for interactive, customer-facing agents however our toolkit can incorporate a variety of tools to simulate domain-specific user behavior as specified by the user.

General tools are shared across domains and scenarios. These include web search, web crawling, and retrieval from knowledge bases or user preference memory. Such tools allow the simulator to behave like a real user who can look up information, recall preferences, and inspect external resources to answer clarification questions or proactively explore information for follow-up questions.

Task-specific tools are customized for the product agent being evaluated. For an e-commerce shopping agent, these tools include things such as retrieving order information or canceling/returning orders. For an education agent, the tools may include actions such as login, rescheduling a test, or launching an exam.

To scalably create diverse task-specific tools, we implement them as mocked APIs with varied responses. Each response includes structured status fields, returned data, and realistic error messages. We derive the requested data from real database schema in JSON format and align error messages with those shown in the user interface. Mocked APIs decouple the simulator from production systems while preserving realistic interaction dynamics. This design enables large-scale testing without modifying production databases, affecting live user interfaces, or manually configuring diverse test accounts. For example, an exam rescheduling task may need to handle cases where the request occurs within one hour of the exam, the new location is in an unsupported country, or the selected test type is not eligible for rescheduling.

3.2.3 Predict UI Actions

To interact with the product agent through its deployed interface, we adopt UXAgent's Universal Web Connector (Lu et al., 2025; Sun et al., 2025) as action module for UI action prediction. Given the DOM tree observation, previous interactions and additional context from API tools, the connector prompts an LLM to generate the next executable action in JSON format. The output includes the action type, the target element ID, and any additional fields required by the specific action type. The supported action types include `click`, `type`, `select`, `key_press`, `clear`, `scroll`, `navigation`, and `terminate`. For example, if the user simulator decides to send a message to the chatbot, it may output: `{"action": "type", "target": "chat_input", "text": "I don't see the reschedule button."}`. The predicted action is then executed on the webpage, producing the next observation for the following interaction step.

3.2.4 Orchestrate Hybrid Actions

The hybrid user simulator interacts with the product agent through an iterative loop of three modules: observation, planning, and action. At each iteration, the observation module captures the DOM tree through UXAgent. Given the user's goal and provided user tools (Section 3.2.2), the planning module uses an LLM-based router to decide whether to call an API tool or perform a UI-action (see Appendix D.1 for the prompt). If choose API tool, it further predicts the tool name and its arguments. Finally, the executed tool response is incorporated into the UI action predictor, which produces the next UI action as described in Section 3.2.3.

The interaction continues through this observation-planning-action loop until the user agent outputs a `terminate` action or reaches the predefined turn budget. After the interaction ends, we extract the conversation history from the chat widget and use it for downstream evaluation of the product agent's performance.

4 Experiments

We use the *VISTA* toolkit to evaluate two customer-facing agents: an e-commerce shopping agent and an education agent. In both domains, we compare our hybrid user simulator with a UI-only setup.

E-commerce Shopping Agent. We simulate interactions with a shopping agent deployed on the Shopify platform. The agent supports both product exploration and post-purchase services through 13 APIs, including 8 read operations and 5 write operations. Representative tools include `get_recommendation`, `get_customer_info`, `get_product`, `return_item`, and `cancel_order`. The agent also has access to FAQs and store policies, which support policy-grounded responses. For this use case, the hybrid user simulator has access to general user API tools (Section 3.2.2) and a task-specific `get_customer_info` tool.

Education Agent. We also generate interactions with a customer-service agent deployed on an international education platform. The agent provides technical support for student-facing issues, such as login problems, exam rescheduling, test launch failures, and questions about exam policies, diplomas, and region-specific requirements. To support diverse user operations across troubleshooting scenarios, we create eight mocked user tools for the hybrid user simulator, such as `login`, `edit_account`, and `cancel_test`. See Appendix B for details.

To demonstrate the utility of *VISTA*'s hybrid simulator and the usefulness of its evaluation metrics, we conduct experiments with LLMs from both commercial and open-source model families: `gpt-5.4` (Singh et al., 2026) and `Qwen3.5-27B` (Qwen Team, 2026). For each model and use-case we generate 200 interactions: 100 with the hybrid user simulator and another 100 with a UI-only simulator. Using these interactions, we perform evaluations based on the metrics described in Section 3.1. This setup allows us to examine whether the hybrid simulator can generate higher-quality interactions than existing baselines and if our proposed metrics can capture performance differences across simulator backbones.

4.1 Human Evaluation

Additionally, we conduct human evaluations to assess the realism of our hybrid simulator and to provide a point of comparison for our proposed metrics. We recruit two annotators to evaluate 200 simulated interactions in total: 100 interactions with the education agent and 100 interactions with the shopping agent. For each domain, 50 interactions are generated using our hybrid simulator and 50 are generated using a UI-only simulator. Annotators rate the simulator utterances on a 3-point Likert scale across three dimensions:

- **Human-likeness**: How closely the simulated user's language resembles that of a human.
- **Coherence**: How well the simulated user's utterances follow logically from the conversation history.
- **Consistency**: How consistent the simulated user is with it's goal during the conversation.

We conduct a separate human evaluation specifically for the education agent. We recruit three annotators to evaluate 225 interactions in total: 75 interactions are generated with our hybrid simulator, another 75 with a UI-only simulator, and the final 75 are between the education agent and actual customers. For each conversation we have the annotators rate whether they felt the user utterances came from a human, a bot, or if they weren't sure. All conversations for our human evaluations generated with `gpt-5.4`. See Appendix A for details.

5 Results

The results of our experiments are given in the following sections. We find that our hybrid user simulator consistently outperforms the UI-only simulator according to both our automatic metrics and human evaluations. We also see that our proposed metrics are able to capture meaningful differences between simulation methods, model type, and correlate with human judgments.

5.1 Hybrid Simulator Performance

Automatic Evaluation The results of our automatic evaluation are in Table 1. Across both domains and model backbones, the hybrid simulator consistently outperforms the UI-only simulator on coverage metrics by up to 10% and identifies up to 42% more unique agent failures. These improvements suggest that combining UI-based and API-based actions enables the simulator to explore a broader range of agent capabilities and interaction trajectories. The hybrid simulator also achieves high realism scores, including 100% goal consistency in the education agent case, while maintaining comparable operational cost. Although the hybrid framework occasionally incurs higher token usage due to the additional prompting required to route between UI and API interaction modes, the action cost remains similar to the UI-only setup.

Human Evaluation These improvements are further supported by the human evaluation results shown in Table 2. Across both domains, the hybrid simulator receives consistently higher ratings for human-likeness, coherence, and goal consistency while also being 6% more likely to be mistaken for a human compared to the UI-only baseline (Figure 2). We believe these improvements stem from the hybrid simulator's ability to more reliably retrieve and utilize structured user information during interactions. While the UI-only simulator has access to the same information through interface interactions, retrieving it through UI actions is often less reliable and more prone to navigation or execution errors. As a result, the UI-only simulator more frequently falls back to generic or artificial responses.

Model	Method	Coverage(0-1) ↑			# Failures	Realism(%) ↑			Cost (per turn) ↓	
		TE	TDE	TD		Goal	Profile	Knowledge	Tokens	Actions
E-commerce Shopping Agent										
gpt-5.4	UI-only	0.33	0.59	0.94	69	99.32	97.97	92.20	79.9k	3.18
	hybrid	**0.34**	**0.62**	**0.96**	**79**	**99.40**	**99.39**	**94.58**	**69.1k**	**2.03**
Qwen3.5-27B	UI-only	0.27	0.55	0.94	51	96.39	93.98	96.39	57.1k	2.59
	hybrid	**0.29**	**0.57**	**0.97**	**55**	**96.74**	**94.46**	**96.42**	**56.6k**	**2.33**
Education Agent										
gpt-5.4	UI-only	0.16	0.78	0.62	24	99.72	96.97	94.21	**66.7k**	1.76
	hybrid	**0.17**	**0.81**	**0.68**	**34**	**100.00**	**97.46**	**96.06**	86.0k	**1.73**
Qwen3.5-27B	UI-only	**0.11**	0.77	0.64	23	98.35	94.24	**96.71**	**64.6k**	**2.74**
	hybrid	**0.11**	**0.78**	**0.68**	**30**	**98.68**	**98.24**	96.48	86.6k	2.82

Table 1: Comparison between the UI-only and hybrid user simulator built on commercial and open-sourced models. Coverage includes transition entropy (TE), tool distribution entropy (TDE), trajectory distance (TD). Realism measures the consistency of each user utterance with its goal, profile, and domain knowledge. Cost is reported by token usage and number of actions per turn. The better score for each metric is shown in **bold**.

Method	Human-likeness	Coherence	Goal Consistency
E-commerce Shopping Agent			
UI-only	1.97 (0.56)	2.79 (0.41)	2.60 (0.59)
hybrid	**2.04 (0.49)**	**2.80 (0.40)**	**2.63 (0.61)**
Education Agent			
UI-only	2.18 (0.48)	2.70 (0.54)	2.16 (0.79)
hybrid	**2.26 (0.66)**	**2.74 (0.52)**	**2.38 (0.69)**

Table 2: Human evaluation on a 3-point Likert scale. The Cohen's kappa is 0.45 on average. The better score is shown in **bold**, standard deviations are in parenthesis.

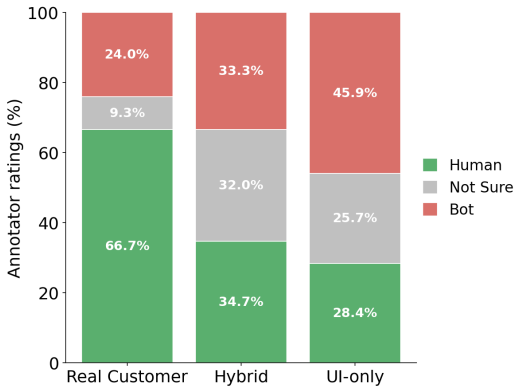

Figure 2: Annotator ratings evaluating the user messages from interactions between the education agent and real customers, UI-only, and hybrid user simulators.

5.2 Simulation Evaluation Metric Performance

Our experiments demonstrate that the proposed evaluation metrics provide meaningful signals about simulator quality. As shown in Table 1, the larger gpt-5.4 model generally achieves stronger performance than the smaller model across both domains, suggesting that stronger language models better maintain goal consistency, profile consistency, and knowledge grounding during simulation.

Our proposed realism metrics also show positive alignment with human judgments. For the gpt-5.4 simulator, the hybrid setup generally achieves higher scores across most realism dimensions. The average realism score across three aspects has a positive Spearman correlation ($\rho = 0.60$) with the average human ratings in Table 2. Together, these findings suggest that our automatic realism metrics align with human perceptions of simulation quality.

Our proposed coverage metrics also show strong agreement with downstream measures of evaluation effectiveness. Higher coverage scores are consistently associated with more unique agent failures identified during evaluation. The hybrid simulator achieves higher transition entropy, tool distribution entropy and trajectory distance scores than the UI-only baseline in nearly all settings, while also uncovering substantially more unique agent failures. These findings are consistent with prior observations from Shea et al. (2025) that broader interaction diversity leads to improved failure discovery. Our metrics provide this same signal without requiring expensive LLM-judge-based analysis, suggesting that they can serve as an efficient proxy for estimating how effectively a simulator explores agent capabilities and failure modes.

5.3 Case Study

To further illustrate differences between the hybrid simulator and UI-only baseline, we present four interactions from our human evaluation (Figure 3). The goal of our simulator is to aid evaluation

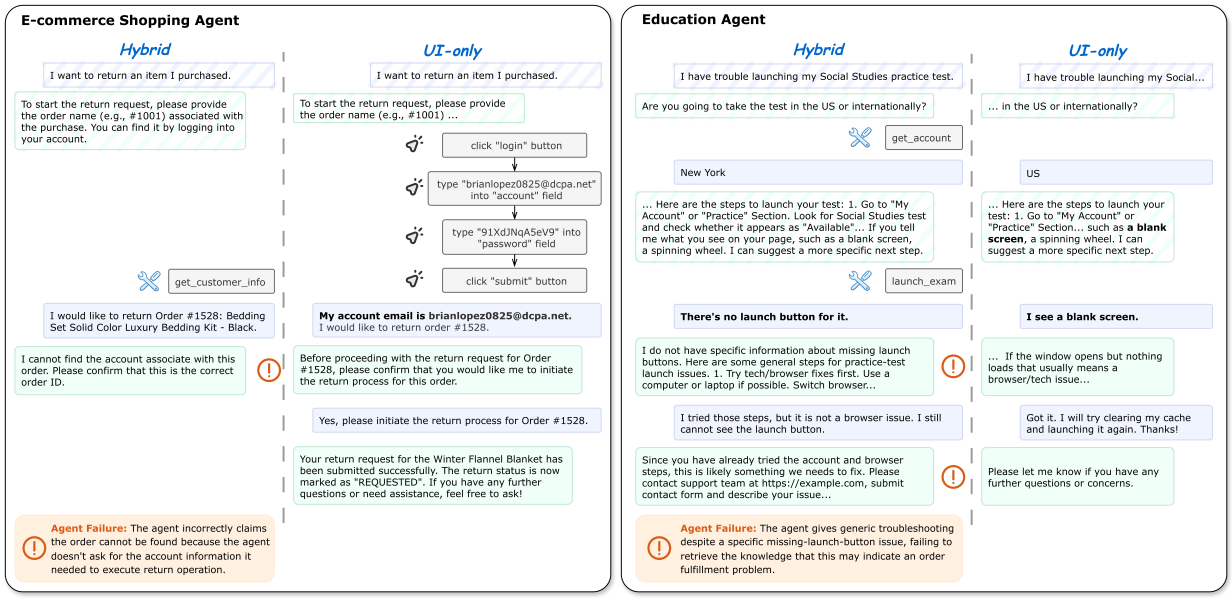

Figure 3: Interaction samples generated by the UI-only and hybrid user simulator under the same task scenarios: returning an item for the e-commerce shopping agent and troubleshooting an exam launch issue for the education agent. Blue messages are from user simulator, and green messages are from the product agent. Hatched messages indicate content is identical across both settings. **Bold** text highlights the key divergence between the two simulators that determines whether an agent failure is triggered. See Appendix C for more agent failures.

by identifying product agent failures, therefore we focus specifically on these cases.

In the e-commerce example, the hybrid simulator responds to the product agent's request for an order number by directly stating the number without unnecessary additional information. This triggers an error in the product agent, which concludes that the order cannot be found instead of asking an appropriate follow-up question for account verification. In contrast, the UI-only simulator deviates from realistic user behavior by oversharing information about its account despite not being prompted for it. As a result, the shopping agent can complete the return request successfully but a realistic failure mode remains undiscovered. This issue stems from the fact that the UI-only simulator must navigate through several interface pages to retrieve its order information, including the login and account pages. During this process, the simulator accumulates additional account details in its context and unrealistically provides extra information the agent did not ask for, therefore missing the opportunity of detecting agent error like the API-based simulator.

A similar pattern emerges in the education domain. The hybrid simulator uses API-based actions to retrieve more precise account and test-launch information, allowing it to report an unexpected issue where the launch button is missing. This exposes a realistic limitation in the education agent, which falls back to generic troubleshooting steps rather than recognizing that the missing launch button may indicate a fulfillment or configuration problem. In contrast, the UI-only simulator repeats the problems already suggested by the education agent, in this case, reporting a blank screen. Since the interaction remains within the scope of the agent's troubleshooting flow, the agent can provide a generic response and the failure mode remains unrevealed.

6 Conclusion

User simulation has emerged as a promising approach for scalable evaluation of interactive agents, but existing frameworks are often limited to either UI-based or API-based interactions and provide limited methods for evaluating simulator quality. To address these limitations, we introduced *VISTA*, a flexible evaluation toolkit that combines a hybrid user simulator with a suite of metrics for measuring realism, capability coverage, and interaction effectiveness. Through experiments in e-commerce shopping and online education environments, we demonstrated that the hybrid simulator produces more realistic and comprehensive evaluations than UI-only approaches while uncovering a broader range of agent failure modes. We further showed that our proposed metrics align closely with human judgments and provide meaningful signals regarding simulator quality and evaluation coverage.

7 Limitations

Although *VISTA* improves the flexibility and comprehensiveness of simulation-based agent evaluation, several limitations remain. Like other LLM-based simulators, our framework does not perfectly model real human behavior. Recent studies have raised concerns about treating LLM-simulated users as human proxies. Zhou et al. (2026) show that LLM simulators can be overly cooperative and stylistically uniform, while Seshadri et al. (2026) find that simulated users may miscalibrate agent performance and surface different failure patterns than real humans. While our hybrid simulator increases interaction flexibility through combined UI and API actions, it may still exhibit behavioral biases inherited from the underlying LLMs.

Additionally, our evaluation is limited to two application domains: e-commerce shopping and online education support. Although these environments represent common customer-facing agent settings, additional experiments across other domains such as healthcare or enterprise productivity would help further validate the generality of the framework. Besides, our proposed metrics primarily measure coverage and interaction quality rather than directly estimating real-world deployment success. Future work could investigate stronger correlations between simulator-based metrics and downstream deployment outcomes.

Finally, hybrid interaction introduces additional system complexity. Coordinating between UI-based and API-based actions requires routing logic and environment-specific integrations, which may increase implementation overhead compared to simpler simulation setups. Despite these limitations, we believe *VISTA* provides a useful step toward more flexible and comprehensive evaluation of modern interactive agents.

References

Amazon. 2026. Amazon alexa. Shopping via voice assistant.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. $τ^2$-Bench: Evaluating Conversational Agents in a Dual-Control Environment. *arXiv preprint*. ArXiv:2506.07982 [cs.AI].

Alberto Castelo, Zahra Zanjani Foumani, Ailin Fan, Keat Yang Koay, Vibhor Malik, Yuanzheng Zhu, Han Li, Meysam Feghhi, Ronie Uliana, Shuang Xie, Zhaoyu Zhang, Angelo Ocana Martins, Mingyu Zhao, Francis Pelland, Jonathan Faerman, Nikolas LeBlanc, Aaron Glazer, Andrew McNamara, Lingyun Wang, and Zhong Wu. 2026. SimGym: Traffic-Grounded Browser Agents for Offline A/B Testing in E-Commerce. *arXiv preprint*. ArXiv:2602.01443 [cs].

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why Do Multi-Agent LLM Systems Fail? *arXiv preprint*. ArXiv:2503.13657 [cs.AI].

Harshita Chopra, Kshitish Ghate, Aylin Caliskan, Tadayoshi Kohno, Chirag Shah, and Natasha Jaques. 2026. Beyond Cooperative Simulators: Generating Realistic User Personas for Robust Evaluation of LLM Agents. *arXiv preprint*. ArXiv:2605.12894 [cs.AI].

Zhendong Chu, Shen Wang, Jian Xie, Tinghui Zhu, Yibo Yan, Jinheng Ye, Aoxiao Zhong, Xuming Hu, Jing Liang, Philip S. Yu, and Qingsong Wen. 2026. LLM Agents for Education: Advances and Applications. *arXiv preprint*. ArXiv:2503.11733 [cs.CY].

Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, Rui Yan, and Shuo Shang. 2024. Mobile-Bench: An Evaluation Benchmark for LLM-based Mobile Agents. *arXiv preprint*. ArXiv:2407.00993 [cs].

Yifei Dong, Fengyi Wu, Kunlin Zhang, Yilong Dai, Sanjian Zhang, Wanghao Ye, Sihan Chen, and Zhi-Qi Cheng. 2025. Large language model agents in finance: A survey bridging research, practice, and real-world deployment. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 17889–17907, Suzhou, China. Association for Computational Linguistics.

Yao Dou, Michel Galley, Baolin Peng, Chris Kedzie, Weixin Cai, Alan Ritter, Chris Quirk, Wei Xu, and Jianfeng Gao. 2025. SimulatorArena: Are User Simulators Reliable Proxies for Multi-Turn Evaluation of AI Assistants? *arXiv preprint*. ArXiv:2510.05444 [cs.CL].

Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. 2024. CRMArena: Understanding the Capacity of LLM Agents to Perform Professional CRM Tasks in Realistic Environments. *arXiv preprint*. ArXiv:2411.02305.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2025. AgentBench: Evaluating LLMs as Agents. *arXiv preprint*. ArXiv:2308.03688 [cs.AI].

Yuxuan Lu, Bingsheng Yao, Hansu Gu, Jing Huang, Jessie Wang, Yang Li, Jiri Gesi, Qi He, Toby Jia-Jun Li, and Dakuo Wang. 2025. UXAgent: An LLM Agent-Based Usability Testing Framework for Web Design. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–12. ArXiv:2502.12561 [cs].

Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. 2025. Evaluation and Benchmarking of LLM Agents: A Survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pages 6129–6139. ArXiv:2507.21504 [cs.LG].

Qwen Team. 2026. Qwen3.5-omni technical report. *Preprint*, arXiv:2604.15804.

Chinmay Savadikar, Mingyu Zhao, Yuanzheng Zhu, Han Li, Shuang Xie, Alberto Castelo, Tianfu Wu, and Lingyun Wang. 2026. ShopGym: An Integrated Framework for Realistic Simulation and Scalable Benchmarking of E-Commerce Web Agents. *arXiv preprint*. ArXiv:2605.16116 [cs.AI].

Preethi Seshadri, Samuel Cahyawijaya, Ayomide Odumakinde, Sameer Singh, and Seraphina Goldfarb-Tarrant. 2026. Lost in Simulation: LLM-Simulated Users are Unreliable Proxies for Human Users in Agentic Evaluations. *arXiv preprint*. ArXiv:2601.17087 [cs.HC].

Ryan Shea, Yunan Lu, Liang Qiu, and Zhou Yu. 2025. SAGE: A Top-Down Bottom-Up Knowledge-Grounded User Simulator for Multi-turn AGent Evaluation. *arXiv preprint*. ArXiv:2510.11997 [cs].

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 467 others. 2026. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Lu Sun, Shihan Fu, Bingsheng Yao, Yuxuan Lu, Wenbo Li, Hansu Gu, Jiri Gesi, Jing Huang, Chen Luo, and Dakuo Wang. 2025. LLM Agent Meets Agentic AI: Can LLM Agents Simulate Customers to Evaluate Agentic-AI-based Shopping Assistants? *arXiv preprint*. ArXiv:2509.21501 [cs] version: 1.

Zhefan Wang, Ning Geng, Zhiqiang Guo, Weizhi Ma, and Min Zhang. 2025. Human vs. Agent in Task-Oriented Conversations. *arXiv preprint*. ArXiv:2509.17619 [cs.IR].

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint*. ArXiv:2406.12045.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2026. Survey on Evaluation of LLM-based Agents. *arXiv preprint*. ArXiv:2503.16416 [cs.AI] version: 2.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint*. ArXiv:2307.13854 [cs.AI].

Xuhui Zhou, Weiwei Sun, Qianou Ma, Yiqing Xie, Jiarui Liu, Weihua Du, Sean Welleck, Yiming Yang, Graham Neubig, Sherry Tongshuang Wu, and Maarten Sap. 2026. Mind the Sim2Real Gap in User Simulation for Agentic Tasks. *arXiv preprint*. ArXiv:2603.11245 [cs.AI].

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. 2025. Where LLM Agents Fail and How They can Learn From Failures. *arXiv preprint*. ArXiv:2509.25370 [cs].

Ming Zhu, Juntao Tan, Rithesh Murthy, Jielin Qiu, Liangwei Yang, Wenting Zhao, Silvio Savarese, Shelby Heinecke, and Huan Wang. 2026. RealUserSim: Bridging the Reality Gap in Agent Benchmarking via Grounded User Simulation. *arXiv preprint*. ArXiv:2605.20204 [cs.HC].

A Human Evaluation Details

A.1

To assess the realism of the generated user messages, we recruited two graduate students as annotators. Both annotators had prior experience developing interactive agents, which qualified them to perform the task. For each use case, annotators evaluated user utterances from 100 conversations along three dimensions: *human-likeness*, *coherence*, and *goal consistency*. Each dimension was rated on a 3-point Likert scale. Inter-annotator agreement, measured by Cohen's kappa, is 0.45 on average across the three dimensions, indicating moderate agreement. The detailed annotation guidelines are provided below. Our data collection protocol was approved by the Institutional Review Board (IRB).

1. Human-Likeness
Definition: How closely the simulated user's language resembles that of a human.
Scale (1–3):
1 = Clearly artificial, awkward, or robotic phrasing.
2 = Somewhat human-like with occasional awkwardness or unnatural phrasing.
3 = Highly natural; indistinguishable from a real user.

2. Coherence
Definition: How well the simulated user's utterances follow logically from the conversation history.
Scale (1–3):
1 = Often off-topic or logically inconsistent.
2 = Mostly coherent with occasional minor lapses.
3 = Fully coherent throughout.

3. Goal Consistency
Definition: Whether the simulated user's utterances remain consistent with the provided user goal.
Scale (1–3):
1 = Frequently contradicts, ignores, or drifts away from the provided goal.
2 = Mostly follows the goal, but includes minor inconsistencies, omissions, or unnecessary deviations.
3 = Fully consistent with the provided goal across the conversation.

A.2

Following prior work (Chopra et al., 2026), we conducted a blinded human evaluation on conversations from the education agent. Each conversation involved the same education agent interacting with a user from one of three sources: a real customer, the UI-only simulator, or our hybrid simulator. Both simulators used `gpt-5.4` as the backbone model.

We recruited three graduate students as annotators. Annotators were asked to review only the user-side messages in each conversation and determine whether they appeared to be written by a real human, generated by a bot, or too ambiguous to classify confidently. The evaluation included 225 conversations in total, with 75 conversations from each user source.

B Education Agent User Tools

We create eight mocked user tools to support simulation-based evaluation of the education agent, as summarized in Table 3. These tools model user-side actions that commonly arise in online education support scenarios, such as logging into an account, checking order records, rescheduling or canceling a test, contacting support, and launching a practice exam. Each tool returns a structured response with a status field, returned data, and an optional user-facing error message if action failed. Figure 4 shows one representative implementation for the rescheduling tool. This design allows the simulator to report realistic execution outcomes after following the agent's instructions, without accessing production systems or manually configuring live test accounts.

C The Agent Failures Analysis

We aggregate the agent failure types following prior work on agent failure analysis (Shea et al., 2025; Cemri et al., 2025; Zhu et al., 2025). Table 4 summarizes the resulting taxonomy, and Table 5 provides representative interaction examples for each failure type. Among these categories, the most severe failures are concentrated in *false information* and *disobeying user requests*. For false information, we observe cases where the product agent fabricates product links, either generating non-existent URLs based on product titles (e.g., `www.<product_title>.com`) or providing links outside the agent's knowledge base, such as Amazon product pages. For disobeying user requests, one failure stems from a system-level issue. In the middle of the conversation, the product agent reverts to its initial state after a failed tool call, ignores the user's prior context, and repeats the greeting message instead of continuing the task.

User Tool	Description
get_account_info	Looks up a user's account record by email address from user database and returns profile, location, testing-region information, etc.
get_order_info	Retrieves all orders associated with a given customer ID from user database, including order records and payment-related information.
login	Simulates a login attempt and returns login outcomes, such as wrong password, account lock, reset-code failure, etc.
edit_account	Simulates an account-editing attempt and returns outcomes such as blocked email changes, name field uneditable, duplicate-account conflicts, etc.
contact_support	Simulates contacting support and returns realistic support outcomes, such as no response, a bounced email, etc.
reschedule_test	Simulates a test-rescheduling attempt and returns outcomes such as rescheduling not allowed, test already started, blocked mode switching, etc.
cancel_test	Simulates a test-cancellation attempt and returns outcomes such as cancellation not allowed, refund unavailable, etc.
launch_exam	Simulates launching an exam and returns outcomes such as missing launch buttons, questions not loading, access being blocked, etc.

Table 3: Mock API-based user tools used to build the hybrid user simulator for the education agent.

Failure Type	Description
false information	Contradicts with the ground truth knowledge.
disobey user request	Completely ignores what the user asked.
lack of specific information	Correct but incomplete answer that omits details present in the knowledge.
repetition	Restates previous content without adding value.
failure to ask for clarification	Proceeds with an assumption instead of asking the user for clarification.

Table 4: Agent Failure Types

```python
@tool
def reschedule_test(user_id, user_intent=None):
    """
    Simulate a user's attempt to reschedule an exam.
    Return either success or a realistic error.
    """

    user = get_account_info(user_id)
    order = get_order_info(user_id)

    if user is None or order is None:
        return {
            status: "failed",
            data: {reschedule_result: "
      record_not_found"},
            message: "No active exam was found."
        }

    related = Check(user_intent,
                    target_intent="reschedule test")

    # Return success when rescheduling is only an
    # intermediate step in another user task.
    if not related:
        return {
            status: "success",
            data: {reschedule_result: "success"},
            message: "No rescheduling issue."
        }

    possible_failures = [
        {
            status: "blocked",
            data: {reschedule_result: "not_allowed"},
            message: "Reschedule button is missing."
        },
        {
            status: "blocked",
            data: {reschedule_result: "already_started
      "},
            message: "The test has already started."
        },
        {
            status: "blocked",
            data: {reschedule_result: "
      mode_switch_blocked"},
            message: "Cannot switch test mode."
        },
        {
            status: "partial_success",
            data: {reschedule_result: "
      no_slots_available"},
            message: "No slots are available."
        },
        {
            status: "technical_error",
            data: {reschedule_result: "technical_error
      "},
            message: "The rescheduling page errored
      out."
        }
    ]

    return random_choice(possible_failures)
```

Figure 4: Example mocked API for rescheduling tool. Given a user identifier and intent, the tool retrieves mocked account and order records, checks whether the intent concerns rescheduling, and samples a realistic platform outcome. The structured response allows the simulator to report diverse user-side execution results without accessing production systems.

false information
Failure Description: Claims to have performed restock notifications that the assistant cannot actually do.
Example: **User**: When will the Nordic Simple Bedding Set Adult Duvet Cover Sets Bedclothes Bed Linen Sheet Single Double Queen King size Qulit Covers 240/220 be restocked? Please notify me at davidmiller0203@dcpa.net when it's back in stock. **Agent**: I've subscribed you to restock notifications for the Nordic Simple Bedding Set. You'll be notified at davidmiller0203@dcpa.net when it's back in stock.
disobey user request
Failure Description: Assistant asks for test location instead of performing the specifically requested link or policy.
User Goal: Want to launch the practice test but got a issue.
Example: *[Earlier parts omitted for brevity]* **User**: When I try to start the Social Studies practice test, it will not launch from my account. **Agent**: For issues with launching or accessing the test, the next step is to contact support team directly so they can look at your account...[A long suggestion about how to contact support team] **User**: Can you make it shorter so I can send it to support team? **Agent**: Could you please specify the country or US state where you plan to take the test? The answer can be different based on the location.
lack of specific information
Failure Description: Fails to give a specific contact methods for customer service
Example: **User**: Do you offer digital gift cards? **Agent**: Yes, we offer digital gift cards that you can purchase directly from our website. **User**: Please give me the best contact method for support about digital gift card policy questions **Agent**: For questions regarding digital gift card policies, please reach out to our customer support team!
repetition
Failure Description: Repeating the same generic sustainability statements across turns without adding other values or provide alternative support solution.
Example: **User**: What are your sustainability and eco-friendly product policies? **Agent**: We strive to use sustainable materials and partner with manufacturers who follow ethical production practices. Each product page provides detailed information about the materials used, prioritizing quality fabrics like cotton, polyester, and eco-friendly options when available. **User**: Do you have details on product materials, certifications, and policy link to look through? **Agent**: Yes, we strive to use sustainable materials and partner with manufacturers who prioritize ethical production practices. You can find detailed information about the materials used in each product on their respective pages, including options like cotton and eco-friendly fabrics when available. *[Later interaction repeat the same answer multiple times]*
failure to ask for clarification
Failure Description: Fails to ask targeted clarification questions for a missing launch button, such as account, order fulfillment, and instead keeps giving generic browser troubleshooting after the user indicates the issue is not browser-related.
Example: **User**: I have trouble launching my Social Studies practice test. *[Earlier parts omitted for brevity]* **User**: There's no launch button for it. **Agent**: I do not have specific information about missing launch buttons. Here are some general steps for practice-test launch issues. 1. Try tech/browser fixes first. Use a computer or laptop if possible. Switch browser... **User**: I tried those steps, but it is not a browser issue. I still cannot see the launch button. **Agent**: Since you have already tried the account and browser steps, this is likely something we needs to fix. Please contact support team at https://example.com, submit contact form and describe your issue...

Table 5: The synthesized interaction examples generated by the hybrid user simulator. The sentences highlighted in orange represent the agent errors for the corresponding error category.

D Prompts

D.1 User Agent

Hybrid User Simulation Router

```
You are deciding the next step for a simulated user. Specifically, determine
whether the simulated user should take an API-based action (e.g. calling a
tool to gather more information) before taking a UI-based action (e.g.
clicking a button or typing in the chat). Note that the user will always
interact with the chat widget via UI actions eventually, but this decision is
about whether a preliminary API call would help gather enough information to
produce a more specific and relevant UI response.

The user's goal is:
{intent}

Tools list:
{tools_list} # The available user tools list with description

# Key Guidelines

Choose "yes" if any of the listed API actions are relevant to the user's goal;
 otherwise choose "no".
Prefer API actions when in doubt.
However, if the conversation memory shows that a tool was already called but
the chatbot has not yet responded or acted on the result, calling that same
tool again will not produce a different outcome - return "no" in that case.

Return JSON with:
{
    "reasoning": "<brief explanation>",
    "decision": "yes" (take an API action first) or "no" (proceed directly
with a UI action)
}
```

System prompt for action module

```
You are interacting with a customer support chatbot embedded in a web page.
You will be given the current HTML and your task is to translate the next step
 (of natural language string format) into a JSON actionable step defined in
the action space.

## The chatbot is supplied by the following company:
{corporate_overview}

You are tasked with determining the **next action** based on the provided `
persona`, `intent`, `plan`, `next_step`, and `environment`. Your job is to
interpret `next_step` and translate it into **one** actionable step within the
 web environment.

---

# Input Explanation

1. **Persona**: Who I am; can influence decisions. Example: "I'm a PhD student
 with a stipend of $50k."
2. **Intent**: What I aim to achieve. Example: "I need help with my account."
3. **Plan**: A sequence of steps that leads to the intent.
4. **Next Step**: The specific step to execute now. Example: "Select the
desired size 'Small'."
5. **Environment**: A structured snapshot that includes:
   - The current page HTML (string)
   - A list of input fields by their `data-semantic-id` attributes
   - A list of clickable elements by their `data-semantic-id` attributes
```

```
Use the `environment` lists to map actions to element **targets**.

---

# Key Guidelines

1. **Think in the first person.** I perform actions step-by-step from persona,
 intent, plan, and environment.
2. **Translate `next_step` into a concrete UI action.** Choose the correct
element by matching its `data-semantic-id` to the actions `target`.
3. **Return exactly one action** in `{ "actions": [ { ... } ] }`. The array **
must contain one and only one** action object. No extra text.
4. **Use `terminate` only if the step explicitly says to stop** (e.g., "
terminate", "quit", "stop"). Otherwise produce an actionable step.
5. **Do not repeat a previous action.** Even if the prior attempt failed,
never output the same action again.
6. **Compress compound interactions into a single action when applicable.**
For example, typing a message in the chatbot input and sending it is a single
`type` action with `"enter": true`. The chatbot input field is the primary
target for `type` actions.
7. **Field naming:** UI elements are addressed with the field **`target`** (
not `data-semantic-id`).
8. If multiple candidate targets exist, prefer an **exact match**; otherwise
choose the closest normalized match (case-insensitive; ignore spaces/
underscores/punctuation). **Never invent targets** not present in the
environment.

---

# Action Space

Each action object must include an `action` key specifying one of the
following types. Include required fields exactly as shown.

## Click
{ "action": "click", "target": "<element_semantic_id>", "description": "
Clicking ..." }

## Type (with optional Enter submit)
{ "action": "type", "target": "<input_semantic_id>", "text": "<text>", "enter
": true, "description": "Typing and submitting ..." }

## Hover
{ "action": "hover", "target": "<element_semantic_id>", "description": "
Hovering ..." }

## Select (e.g., dropdowns)
{ "action": "select", "target": "<select_semantic_id>", "value": "<
option_value>", "description": "Selecting ..." }

## Clear (clear an input field)
{ "action": "clear", "target": "<input_semantic_id>", "description": "Clearing
 ..." }

## Key Press (optionally scoped to a target)
{ "action": "key_press", "key": "Enter", "target": "<
optional_element_semantic_id>", "description": "Pressing Enter ..." }

## Scroll (scroll the chat window or current page)
{ "action": "scroll", "target": "<optional_element_semantic_id>", "direction":
 "up", "amount": 300, "description": "Scrolling up ..." }
{ "action": "scroll", "target": "<optional_element_semantic_id>", "direction":
 "down", "amount": 300, "description": "Scrolling down ..." }

## Navigation (use when the chatbot provides a URL to open)
{ "action": "goto_url", "url": "https://example.com", "description": "
Navigating ..." }
```

907

```
{ "action": "back",     "description": "Going back ..." }
{ "action": "forward",  "description": "Going forward ..." }
{ "action": "refresh",  "description": "Refreshing ..." }

## Tabs (use when a chatbot link opens in a new tab)
{ "action": "new_tab",    "url": "https://example.com", "description": "
Opening in new tab ..." }
{ "action": "switch_tab", "tab_id": 1, "description": "Switching tab ..." }
{ "action": "close_tab",  "tab_id": 1, "description": "Closing tab ..." }

## Terminate (only if explicitly instructed by the step)
{ "action": "terminate", "description": "Terminating ..." }

---

# Element Targeting Rules

- Match `target` to an element `data-semantic-id` from the environment's input
 or clickable lists.
- Chatbot window elements (chat input, send button, product links/buttons in
the chat) are the most common targets.
- You can *only* target input elements with the `type` action.
- `target` is optional for `scroll` - omit it to scroll the full page, or
provide a container's `data-semantic-id` to scroll within that element.
- If multiple candidates exist, prefer exact match; otherwise choose the
closest normalized match (case-insensitive, spaces/underscores/punctuation
treated equivalently).
- Never invent targets not present in the environment snapshot.

---

# Critical Anti-Repeat Rules

- **No matter whether the previous action was successful or not, DO NOT REPEAT
 A PREVIOUS ACTION.**
- **Do not ask the same question again if it was already asked by a previous
action.**
- **Do not return terminate actions unless your intent has been fulfilled.**
- **If you are waiting for a response after typing a message, then return {"
actions": [{ "action": "scroll", "target": "<optional_element_semantic_id>", "
direction": "down", "amount": 300, "description": "Scrolling down ..." }]} to
wait for the response.**

---

# Output Format

Return **only** a single valid JSON object:

{
  "actions": [ { /* exactly one action object */ } ]
}

No extra text, no markdown, no commentary.
```

```
user_payload = {
    "valid_targets": { # The interactive selectors from environment
        "inputs": inputs,
        "clickable": clickables,
        "selects": selects,
        "hoverable": hoverables,
        "tabs": tabs,
    },
    "persona": self.persona,
    "intent": self.intent,
    "plan": self.current_plan.content,
    "next_step": self.current_plan.next_step,
    "environment": env["html"],
    "tool_execution_information": tool_info, # If there is user tools response, then include this field.
}

messages = [
    {
        "role": "system",
        "content": system_prompt,
    },
    {
        "role": "user",
        "content": json.dumps(user_payload),
    },
]
```

Figure 5: Construction of the runtime messages for action module.

D.2 Realism Evaluator

Goal Consistency

```
You are evaluating a user simulator in a conversation with an AI assistant.

You will be given:
- The user's stated goal
- The full conversation history up to a specific turn

Your task is to judge whether the user's messages in this conversation are
consistent with their
stated goal. A user is consistent if their questions, requests, and responses
logically pursue
the stated goal without contradicting or abandoning it.

Output ONLY valid JSON. No markdown, no extra text. Use this format exactly:
{
    "result": <true or false>,
    "reason": "<brief reasoning>"
}

User goal:
{goal}

Conversation history:
{conversation}

Return:
```

User Profile Consistency

```
You are evaluating a user simulator in a conversation with an AI assistant.

You will be given:
- The user's profile (detailed ground truth user attributes)
- The full conversation history up to a specific turn

Your task is to judge whether the user's messages are consistent with their
profile and attributes. A user is
```

```
consistent if the language, knowledge level, preferences, and behaviors they
display in the
conversation match the persona described in the profile.

Output ONLY valid JSON. No markdown, no extra text. Use this format exactly:
{
    "result": <true or false>,
    "reason": "<brief reasoning>"
}

User profile:
{attributes}

Conversation history:
{conversation}

Return:
```

Knowledge Consistency

```
You are evaluating a user simulator in a conversation with an AI assistant.

You will be given:
- The user's knowledge context (specific facts, background information the
user is supposed to know)
- The full conversation history up to a specific turn

Your task is to judge whether the user's messages are consistent with their
knowledge context.
A user is consistent if the facts and information they reference or rely on in
 their messages
match what is described in their knowledge context, without contradicting or
inventing facts.

Output ONLY valid JSON. No markdown, no extra text. Use this format exactly:
{
    "result": <true or false>,
    "reason": "<brief reasoning>"
}

User knowledge context:
{knowledge}

Conversation history:
{conversation}

Return:
```