Where Did It Go Wrong? Process-Level Evaluation of Web Agents with Semantic State Tracking

Jiwan Chung[1] **JiHyuk Byun**[1] **Vibhav Vineet**[2] **Seon Joo Kim**[1]
[1]Yonsei University [2]Microsoft Research
jiwan.chung.research@gmail.com

Abstract

Web agents act through long interaction sequences, yet existing benchmarks evaluate only terminal success, discarding all process information and offering little guidance on improvement. In this work, we conduct a process-level analysis of web agents. We introduce WEBSTEP, a benchmark of 1,800 task instances with controlled difficulty and automatic semantic state tracking. Each website exposes a deterministic semantic MDP alongside the GUI: the agent operates on the interface, while the environment records high-level states and transitions in the background, enabling fine-grained analysis without manual annotation. Based on the semantic trajectory, we first show that process metrics reveal differences invisible to outcome evaluation: three agents whose success rates cluster within 34–37% diverge in exploration reach versus execution accuracy. Then, decomposing by skill characterizes the nature of these differences, exposing opposite per-skill rankings hidden within the same website: e.g., on Q&A, Claude CUA outperforms OpenAI CUA by 30% on navigation actions yet underperforms it by 6.7% on inspection, pinpointing a concrete skill to improve even within a domain. Bifurcation analysis further localizes the decisive error that loses the task and shows that this error is agent-specific rather than shared. Finally, these differences widen as tasks grow harder: success rate is similar on easy tasks but separates sharply as exploration becomes more demanding. Our process-level analysis opens a new avenue in web agent evaluation, providing fine-grained and actionable insight into where and how each agent should be improved. Project page: https://jiwanchung.github.io/webstep/

1 Introduction

Web agents automate interactive digital tasks such as shopping, booking, and document management (Awadallah et al., 2025; Qin et al., 2025; Gupta et al., 2026). Unlike single-turn instruction following, these tasks require multi-step navigation, information gathering across pages, and dependent action execution (Zhou et al., 2024; Koh et al., 2024). Thus, the final outcome alone does not fully characterize agent behavior: the trajectory itself carries meaningful information about the interaction process.

Most existing web agent benchmarks, however, evaluate agents primarily by terminal success (He et al., 2024; Mialon et al., 2023; Deng et al., 2023; Xue et al., 2025). This outcome-only view compresses an entire interaction trajectory into a single binary result, obscuring qualitatively different failure modes. As illustrated in Figure 1, an agent that reaches the correct colleague's message thread but sends the wrong emoji receives the same score as one that never finds the relevant page at all. Terminal success therefore provides little actionable insight into where an agent failed or which capability limited performance, as it does not support accurate credit assignment (Pignatelli et al., 2024; Lù et al., 2025; Gritta et al., 2026).

In this work, we introduce WEBSTEP, a benchmark for process-level evaluation of web agents. The core idea is to construct each website as a *semantic Markov Decision Process (MDP) dual*: agents interact only with the rendered GUI, while the benchmark records the

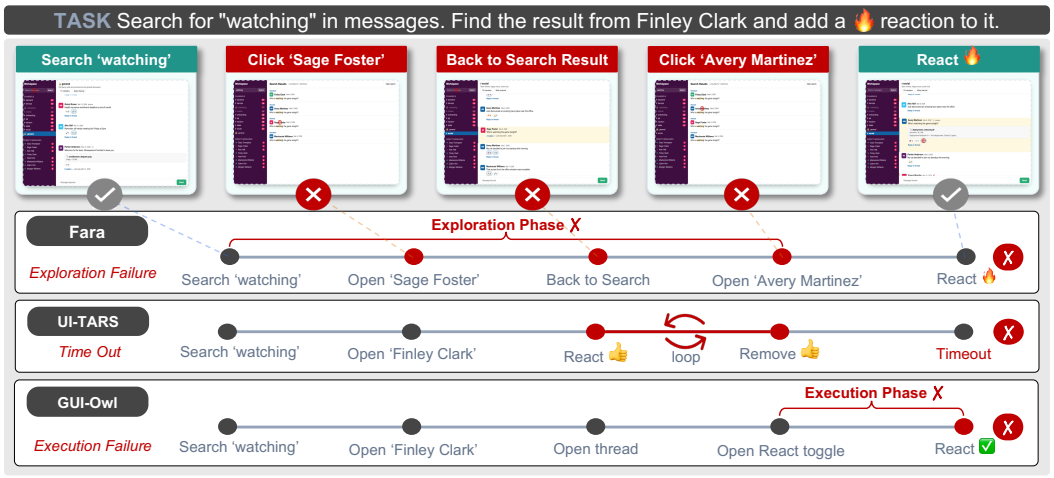

Figure 1: **Terminal outcome alone cannot distinguish qualitatively different failures.** In the task of finding Finley Clark's message and reacting with a :fire: emoji, Fara fails to locate the correct target, while GUI-Owl reaches the correct thread but applies the wrong reaction.

corresponding semantic state transitions induced underneath. Because the GUI is rendered directly from the semantic model, the full semantic trajectory is recovered exactly, enabling automatic state tracking and process-level evaluation without manual annotation.

Using WEBSTEP, we show that process-level evaluation extends web agent evaluation from ranking to diagnosis. It separates agents that appear similar under terminal success by disentangling exploration from execution (Section 4.1), attributing their differences to specific skills (Section 4.2), localizing the step at which unsuccessful trajectories first diverge (Section 4.3), and identifying the task-complexity regimes in which these failures become most pronounced (Section 4.4). In turn, this yields concrete targets for improvement: which skills to strengthen, which temporal failure patterns to correct, and which complexity regimes to emphasize during training and evaluation.

Our main contributions are as follows:

1. We introduce **WEBSTEP**, a benchmark of 10 self-hosted websites and 1,800 tasks, each paired with a semantic MDP dual that tracks structured states and transitions.
2. We develop a **process-level evaluation framework** that decomposes agent behavior into exploration success, execution outcomes, and skill invocation patterns, and supports trajectory comparison at shared semantic states to localize failure modes.
3. We present a **process-level analysis** of existing web agents, showing that models with similar terminal success rates exhibit qualitatively distinct failure signatures.

2 Process-level evaluation via semantic MDP duals

Process-level evaluation requires access to the semantic effects of agent actions, yet web agents interact only through low-level operations such as clicking, typing, and scrolling. As shown in Figure 2, we address this mismatch by mapping each raw GUI trajectory to an aligned semantic trace that records which items were visited, what information became available, and which high-level action was taken. This is enabled by constructing each website as a *semantic MDP dual*: the agent acts only on the graphical user interface, while the benchmark tracks the corresponding semantic state transitions underneath.

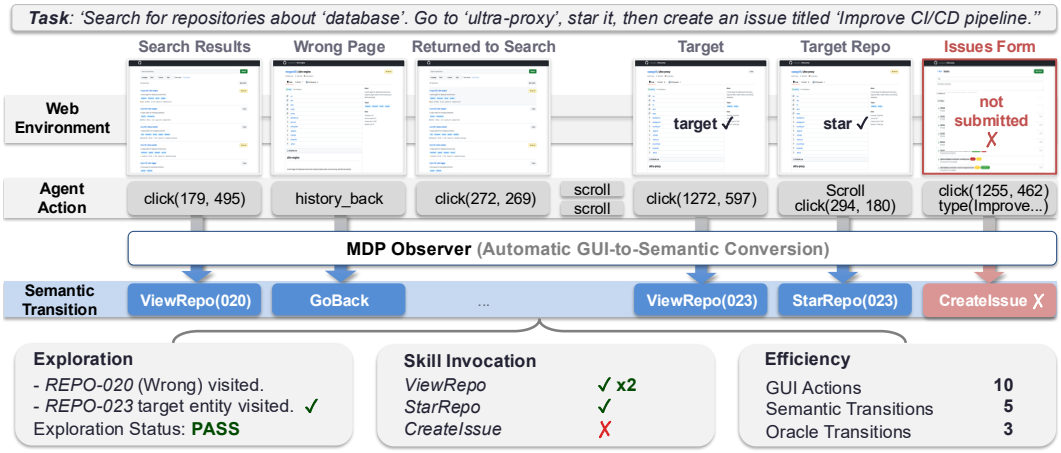

Figure 2: **Overview of WebStep.** Each visual website is paired with a semantic MDP dual that converts raw GUI actions into interpretable semantic transitions. For example, clicking and typing merge to transitions such as ViewRepo, StarRepo, and CreateIssue, making explicit which item was visited and which semantic action was taken. These traces enable automatic process-level analyses such as exploration, skill invocation, and step efficiency.

2.1 Semantic MDP

For each website, we define a deterministic semantic MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, \rho_0)$, where:

- $\mathcal{S}$: a structured semantic *state* space. Each state is decomposed as $s = (s^p, s^i)$, where:
  - s^p: the current interface position (e.g., URL, search and filter settings, pagination and open modals);
  - s^i: the item attributes revealed to the agent (e.g., a product's price, score, or seller).
- $\mathcal{A}$: a typed semantic *action* space abstracting away raw coordinate-level interactions (e.g., issuing a search, opening a detail page, or executing the final task action);
- T: a deterministic *transition* function that maps each state-action pair (s_t, a_t) to the next semantic state $s_{t+1} = T(s_t, a_t)$;
- ρ_0: the task and environment configurations, elaborated in Section 2.2.

For each agent run on a task, we record the resulting MDP trace τ

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T), \qquad s_{t+1} = T(s_t, a_t), \tag{1}$$

which serves as the basis for process-level evaluation.

GUI–MDP coupling. Each website is implemented so that the semantic MDP is the source of truth for all behavior. When the web agent performs a GUI action sequence (e.g., searching, filtering, or opening a detail page) the frontend forwards it to the semantic model instead of updating the page on its own. The semantic model applies the corresponding state transition, determines what information is visible in the new state, and the page is then re-rendered from that state. As a result, the task-relevant GUI trajectory is a direct execution of the semantic MDP, which allows the full semantic trajectory to be recovered exactly.

2.2 Task and world generation

Given a website's semantic MDP, each task consists of two jointly generated components: a natural-language *instruction* and a corresponding *world*, defined as a set of items with concrete attribute values (e.g., brand, price, seller, or delivery option). Each task is constructed by combining a task template (e.g., "Add a bag with [Constraints] to the cart") with

	Deterministic	Tasks	Process-eval	Hard negative
Online-Mind2Web (Xue et al., 2025)	✗	300	✗	✗
WebVoyager (He et al., 2024)	✗	643	✗	✗
WebWalkerQA (Wu et al., 2025)	✗	680	✗	✗
AssistantBench (Yoran et al., 2024)	✗	214	✗	✗
WebArena (Zhou et al., 2024)	✓	812	✗	✗
VisualWebArena (Koh et al., 2024)	✓	910	✗	✗
WorkArena++ (Boisvert et al., 2024)	✓	682	✗	✗
Mind2Web-Live (Pan et al., 2024)	✗	542	Key node	✗
WEBSTEP	✓	**1,800**	**MDP**	✓

Table 1: **Comparison with existing web agent benchmarks.** *Deterministic*: self-hosted deterministic websites. *Process eval*: Key node uses manual milestones; MDP enables automatic stage decomposition and skill attribution. *Hard negative*: controlled distractors.

a randomly sampled set of item-attribute constraints (e.g., price $\leq$ 300 and rating $\geq$ 3.5). The constraints are instantiated into text instruction, and the world is then populated to satisfy the resulting task specification. This coupled generation process enables:

Benchmark validity. The world is guaranteed to contain exactly one target that satisfies all constraints in the instruction, so every task is *solvable* and has a *unique* answer.

Task complexity control. Alongside the target, the generator places *hard negative* items: distractors that share the target's visible attributes on list and search-result pages (e.g., same title, price, and category) but differ on an attribute visible only after opening the item's detail page (e.g., seller or delivery policy). Identifying the target on such tasks therefore requires navigating to individual detail pages. How many hard negatives are planted is itself a design variable: templates whose instruction names the target directly plant none, so tasks range from those solvable at list level to those demanding detail-page comparison. This allows explicit problem complexity control, enabling complexity measure on three axes: hard negative count, informational access level, and oracle trajectory length. The corresponding definitions and analysis are in Section 4.4.

Oracle trajectory. We can statically derive an *oracle trajectory* for each task since both the MDP and the item set are fully specified. Refer to Appendix E.3 for details.

3 WEBSTEP

WEBSTEP is a benchmark for process-level evaluation of web agents in visually realistic environments with deterministic and semantic underlying dynamics. It consists of 10 self-hosted websites spanning diverse domains (Figure 7), including productivity tools, commerce platforms, professional and collaborative platforms, and technical knowledge sites. Each website is modeled after a real site from the same domain (Appendix D.2), while replacing the backend with the deterministic semantic model defined in Section 2.1. Additional dataset statistics are provided in Table 4.

Table 1 positions WEBSTEP among existing web agent benchmarks. Relative to live-website settings, it provides deterministic self-hosted environments. Compared to terminal-level benchmarks, it adds process-level evaluation grounded in an explicit semantic MDP and complexity control with task-conditioned world generation. Further discussion on related work is in Appendix B.

We design a multi-stage pipeline to construct WEBSTEP. Starting from live website workflow traces, a coding agent (Anthropic, 2026) with iterative human review produces a formal MDP specification and converts it into a self-hosted website driven by the semantic model. Each site is then validated through automated testing, including MDP unit tests, GUI interaction

Agent	Success Rate (%) ↑			Information ↑	Steps		
	Terminal	Exploration	Execution	Coverage (%)	GUI	Semantic	GUI/Semantic ↓
Fara-7B	35.7	47.3	75.0	76.4	18.9	8.3	2.3
GUI-Owl-1.5-8B	34.4	43.3	78.6	79.5	28.3	10.4	2.7
UI-TARS-1.5-7B	37.1	49.2	73.9	80.3	35.0	14.0	2.5
Qwen3.5-122B	58.2	66.3	86.7	89.1	22.1	9.8	2.3
OpenAI CUA	82.7	89.4	92.4	96.8	19.7	10.0	2.0
Claude CUA	85.3	91.0	93.7	96.3	14.3	9.3	1.5

Table 2: **Aggregate evaluation results.** We report terminal success together with process-level metrics from semantic MDP traces: exploration, execution, and information coverage. These metrics reveal behavioral differences not visible from terminal success alone.

tests, and oracle trajectory replay, together with manual verification. Full pipeline details and example MDPs are provided in Appendices E.1 and H.3, respectively.

4 Experiments and Results

We analyze web agent behavior using the process-level metrics automatically produced with WEBSTEP. We begin by showing that they reveal differences hidden by outcome metrics (Section 4.1), then characterize the nature of those differences (Section 4.2), localize where they arise within trajectories (Section 4.3), and finally show that they widen as exploration difficulty increases (Section 4.4). Full results and qualitative examples appear in Appendices F and I, respectively.

Models. We evaluate six web agents spanning both small specialist models, UI-TARS-1.5 (Qin et al., 2025), Fara (Awadallah et al., 2025), and GUI-Owl-1.5 (Ye et al., 2025), and large generalist models, Qwen3.5 (Yang et al., 2025), OpenAI CUA (GPT-5.4) (OpenAI, 2025), and Claude CUA (Claude Sonnet 4.6). To ensure a consistent evaluation protocol, we disable external actions such as cross-website navigation and Google search, and evaluate all tasks at non-critical terminal points. More details are provided in Appendices C.2 and C.3.

4.1 Aggregate Results

We begin with coarse trajectory-level summaries derived from the semantic MDP traces. While these metrics do not yet expose fine-grained process-level breakdowns, they already reveal behavioral differences that terminal success alone obscures.

Metrics. All are computed automatically from the MDP trace $\tau = (s_0, a_0, s_1, \ldots, s_T)$.

- *Terminal Success Rate (SR):* The standard binary outcome measuring whether the agent found the correct target and successfully executes the final commit action on it;
- *Exploration SR:* Evaluates whether the agent isolates the correct target item $e^\star$ before committing. Formally, if $V_c = (v_1, \ldots, v_k)$ denotes the sequence of items visited on detail surfaces prior to the first commit, exploration succeeds if and only if $v_k = e^\star$;
- *Execution SR:* The terminal success rate conditioned strictly on successful exploration. This isolates the agent's ability to finalize a task once the correct target has been found;
- *Informational Coverage:* The fraction of task-relevant attributes the agent witnessed, serving as a measure of information-gathering thoroughness (Refer to Section C.4);
- *Step Efficiency:* We record both raw *GUI steps* and *semantic steps* (state-changing MDP transitions). Their ratio quantifies the agent's interaction efficiency.

Results. Table 2 reports aggregate performance. Under terminal success, the small agents appear largely similar, clustering at 34–37%. The trajectory-derived metrics, however, reveal substantial behavioral differences. UI-TARS identifies the correct target more often than GUI-Owl (Δ +5.9%) but performs worse during execution (Δ -4.7%). Its higher information

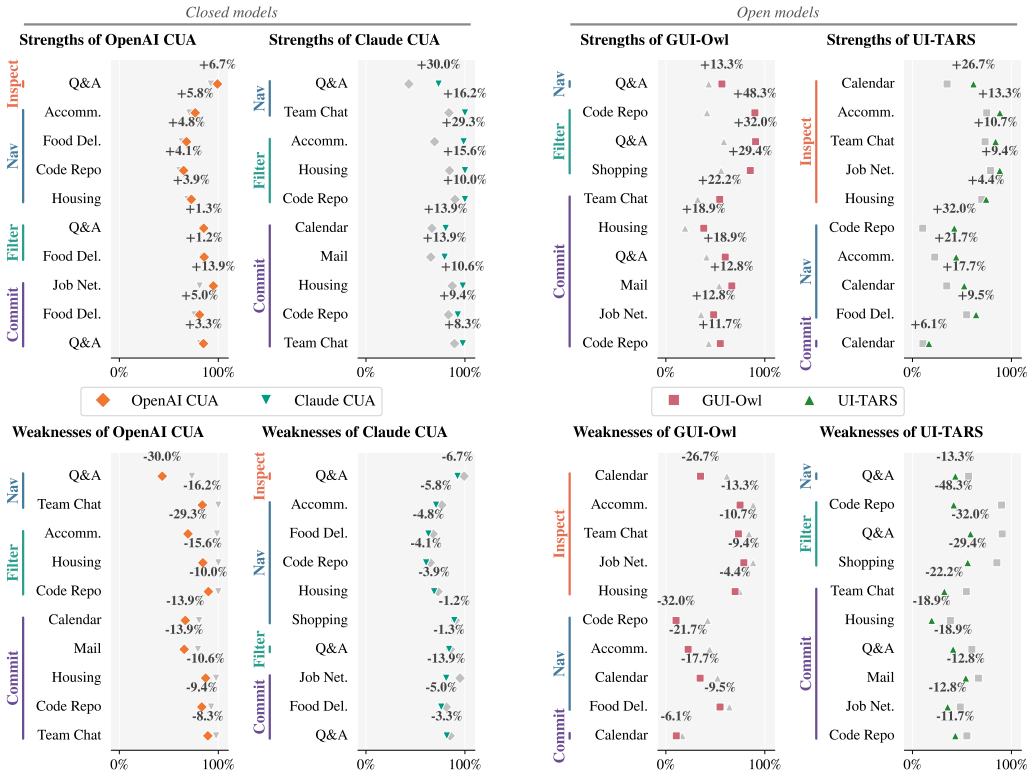

Figure 3: **Skill-level strengths and weaknesses.** Agents exhibit distinct within-website skill profiles. e.g., on Housing, OpenAI CUA is stronger on *navigation* but weaker on *filter*, showing that an agent's weakness can be concentrated in a specific interaction skill. Point labels (e.g., +16.2%) denote the gap relative to the compared model.

coverage and step count indicate a broader, more exploratory strategy. In contrast, Fara takes the fewest steps among the small agents and has the lowest information coverage of any agent evaluated, suggesting more limited exploration.

The larger models, Qwen3.5, OpenAI CUA, and Claude CUA, perform better in both exploration and execution, and also achieve higher information coverage. But this higher coverage is not just a result of exploring more: OpenAI CUA covers more information than Qwen3.5 while using steps more efficiently, and its lower GUI-to-semantic step ratio shows that more of its actions produce meaningful semantic progress.

> Finding 1. **Terminal success hides behavioral differences.** e.g., while the small-scale agents cluster at similar terminal success, trajectory-level summaries separate UI-TARS as exploration-strong but execution-weak, and Fara as under-exploratory (Table 2).

4.2 Skill-level diagnosis

Aggregate summaries show overall patterns, but they do not isolate the specific interaction capability responsible for a model's weakness. A model may appear strong or weak on a website for different underlying reasons, and broad summaries can hide these differences by mixing opposing skill-level signals. To make the diagnosis actionable, the evaluation must identify which interaction skills are present in the model's behavior and which are not.

We operationalize this with skill *invocation*. For each task, we use an oracle trajectory as a reference. Although oracle trajectories are not unique solutions, they instantiate shortest

valid strategies for completing the task. Let $S^\star$ be the set of skill types that appear in the oracle trajectory, and let S be the set of skill types that appear in the model trajectory. For each skill $k \in S^\star$, we define *invocation* as whether $k \in S$.[1]

Skills. We map MDP actions into five categories (full definitions in Tables 6 and 7):

- *inspect*: opening an item's detail view;
- *navigate*: transitioning between interfaces without directly narrowing candidates;
- *search*: text query to retrieve candidates;
- *filter*: using the filtering interface to reduce the candidate set;
- *commit*: performing the final task-completing action on the selected target.

Strengths and weaknesses. Figure 3 shows that agents exhibit distinct skill profiles. Between Claude CUA and OpenAI CUA, Claude CUA is generally stronger at executing the final *Commit* action and performing *Filter*, although their relative proficiencies vary across websites. Among the smaller agents, GUI-Owl is stronger at *Filter*, whereas UI-TARS is stronger at *Navigate*. These contrasts show that performance differences between agents are often concentrated in specific interaction skills.

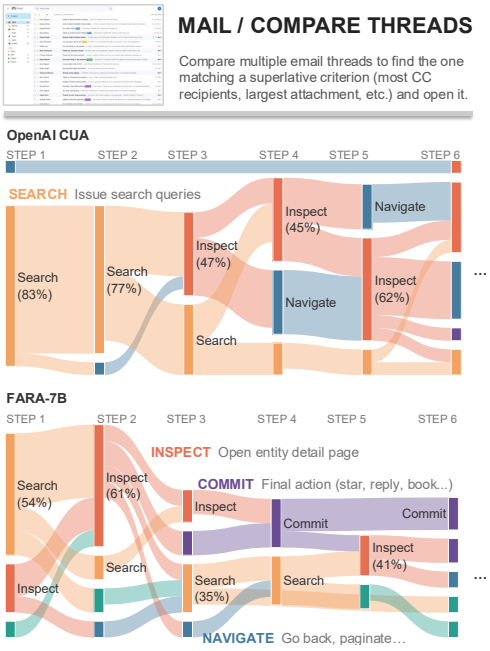

Figure 4: **Temporal skill invocation patterns.**

These differences become even sharper within a single website. On the Q&A website, Claude CUA substantially outperforms OpenAI CUA on *Navigate* ($\Delta + 30.0\%$) but underperforms on *Inspect* ($\Delta - 6.7\%$). A website-level score would average these opposing signals and obscure the underlying diagnosis. Skill-level analysis instead pinpoints the specific weakness: in this case, item inspection. This diagnosis directly suggests adding training examples that strengthen inspection behavior.

Temporal structure of skill invocation. Semantic MDP traces also reveal *when* skills are invoked. While Figure 3 shows which required skills agents tend to miss, Figure 4 shows when they are used along the trajectory. OpenAI CUA follows a concentrated early-stage pattern, typically beginning with *Search*, whereas Fara's early actions are more dispersed. Fara also invokes *Commit* earlier, while OpenAI CUA spends more early steps on information-gathering skills such as *Search*, *Navigate*, and *Inspect*. This suggests targeted interventions: reinforcing a more consistent initial skill sequence for Fara, and delaying commitment until more information has been gathered.

> Finding 2: **Agent weaknesses can be skill-specific.** In Q&A, Claude CUA shows strong *Navigate* skills, but underperforms on *Inspect* usage, indicating that its weakness in this website is not uniform, but concentrated in a specific interaction skill (Figure 3).

4.3 Trajectory bifurcations

The preceding analyses identify which skills are weak, but they do not yet show *where* failure is introduced within a trajectory. We address this by comparing successful and failing

[1] We do not measure skill *success* because a semantic skill is usually identifiable only upon completion. Unsuccessful partial attempts therefore often lack a well-defined skill label. For example, a *search* is identifiable only after query entry and submission.

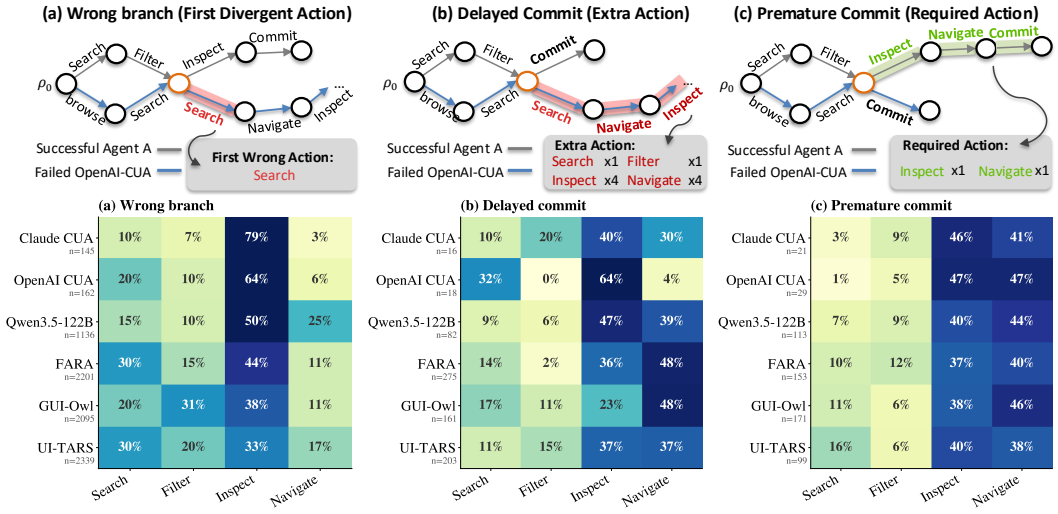

Figure 5: **Trajectory bifurcations at shared semantic states.** Bifurcation analysis localizes where failure is introduced. (a) *Wrong branch*: the decision that first sends the failing trajectory off course. (b) *Delayed commit*: the extra behavior after missing the right time to finish. (c) *Premature commit*: the missing behavior skipped before finishing too early.

trajectories from the same task at the level of semantic MDP states. Since all agents act in the same Markovian environment, trajectories can be aligned whenever they visit the same state. We define the *bifurcation point* as the last shared state before divergence.

We categorize bifurcation points into three types, as shown in the top row of Figure 5:

- *Wrong branch:* the two next actions differ, and neither is COMMIT. We show the mismatched next action.
- *Delayed commit:* the successful next action is COMMIT, but the failing next action is not. We show the remaining failing suffix.
- *Premature commit:* the failing next action is COMMIT, but the successful next action is not. We show the remaining successful suffix.

Wrong branches. Figure 5 (a) identifies the decision that first sends a trajectory onto a losing branch, revealing the branching error each agent is most prone to. For most agents, this first wrong step is *Inspect*, suggesting that they often branch off by opening the wrong item page (33–79%). Fara and UI-TARS also frequently fall back to *Search* when they go off track (30%), whereas GUI-Owl more often diverges through *Filter* (31%). These patterns show that agents tend to enter failure through different kinds of decisions.

Delayed commit. Figure 5 (b) characterizes how agents behave after missing the point at which a successful trajectory would already have finished, revealing the kinds of unnecessary actions they continue to take. GUI-Owl and FARA spend 48% of these extra actions on *Navigate*, whereas OpenAI CUA instead continues mainly with *Inspect* (64%) and *Search* (32%). Their delayed failures therefore take different forms: continued evidence gathering and query refinement for OpenAI CUA, and continued navigation for GUI-Owl and FARA.

Premature commit. Figure 5 (c) shows which types of evidence agents most often fail to gather before committing prematurely. Across all models, the most frequently omitted actions are *Inspect* and *Navigate*, indicating that premature commits primarily arise from insufficient examination of the current page and inadequate exploration of alternative states. The two stronger models, OpenAI CUA and Claude CUA, rarely omit *Search* actions (1–3%), suggesting that they generally identify when external information retrieval is necessary

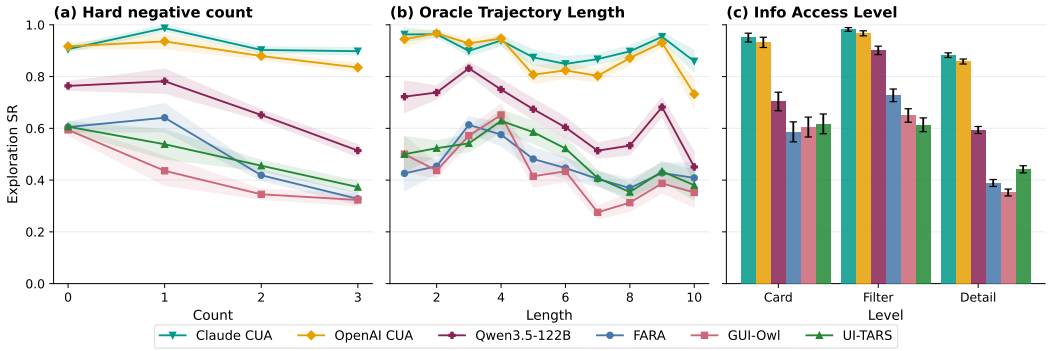

Figure 6: **Exploration success by task complexity. (a)** Performance by *hard negative counts* separates OpenAI CUA from others. **(b)** *Oracle trajectory length* shows downward trend. **(c)** Tasks requiring page *detail* evidence show different model gaps from list-level tasks.

but may still commit before fully inspecting or navigating through the retrieved evidence. Smaller models, including FARA, GUI-Owl, and UI-TARS, exhibit a different pattern: in 10–16% of premature-commit cases, the omitted action is *Search*. This indicates a more fundamental planning failure, in which the agent commits without first recognizing that the task requires information beyond the currently available interface state.

Finding 3. **The decisive error point that loses the task differs across agents.** e.g., although *Inspect* is the most frequent first divergence for every agent, GUI-Owl branches off through *Filter* far more often than the others (31%), while Fara and UI-TARS instead fall back to *Search* (30%) (Figure 5).

4.4 Exploration success by task complexity

Having localized where individual trajectories first go wrong, we next ask whether failure follows systematic patterns as exploration difficulty increases.

Task-complexity measures. We consider three complementary measures:

- *Hard negative count:* the number of plausible distractor items in the world;
- *Oracle trajectory length:* measures the minimum number of steps in the oracle trajectory;
- *Information access level:* groups tasks by the exploration type required to identify the target.
  - *Card* for list-level information only;
  - *Filter* for tasks requiring filtering or search;
  - *Detail* for tasks requiring explicit visits to item detail pages.

Hard negative count. Figure 6 (a) shows that task structure strongly shapes exploration difficulty. We read the trend over HN$\geq$1, since HN=0 tasks name the target directly and therefore plant no distractors by construction (Section E.2). Over this range, exploration success declines monotonically for all six agents, confirming that the procedurally generated distractors induce genuine exploration difficulty. The strong models Claude CUA and OpenAI CUA are the least sensitive, retaining roughly 90% of their HN=1 exploration success at HN=3, whereas the remaining agents lose 26–49% of theirs, suggesting substantially stronger exploration capacity.

Oracle trajectory length. Figure 6 (b) shows a similar pattern. As the minimum successful trajectory becomes longer, exploration performance generally degrades. Qwen3.5 exhibits a relatively monotonic drop as trajectory length increases, whereas Claude CUA and OpenAI CUA maintain their performance for longer, suggesting better generalization to more complex tasks.

Information access level. Figure 6 (c) further separates the agents by exploration types required. Performance is relatively similar on card tasks, where only list-level inspection is needed. Filter tasks begin to separate the larger models from the smaller ones, and detail tasks further separate the strongest models, Claude CUA and OpenAI CUA, from the third-best model, Qwen3.5. Thus, complex exploration requirements reveal differences in the agents.

Finding 4. **Agent differences in exploration widen as tasks become harder.** Exploration performance is relatively similar on easier tasks, but separates much more clearly as exploration requirements become more demanding (Figure 6).

5 Conclusion

We introduced WEBSTEP, a benchmark for automatic process-level evaluation of web agents. By pairing each self-hosted website with semantic state tracking, WEBSTEP enables fine-grained analysis of agent behavior without manual trajectory annotation. Our results show that terminal success alone hides major differences in how agents explore, act, and fail, whereas process-level evaluation makes these differences directly diagnosable. We hope WEBSTEP supports deeper diagnosis of web agents. Limitations are in Appendix A.

Ethics Statement

WEBSTEP is a benchmark for evaluating web agent behavior in controlled, self-hosted environments. It does not involve human-subject experiments, real user accounts, or interaction with live online services during evaluation. All websites are independently implemented reproductions of publicly observable front-end interaction patterns (screenshots and keyboard+mouse movements), and the benchmark uses synthetic world data rather than real user data or production backends. We recognize that improved web agent evaluation can have dual-use implications, including downstream misuse on real websites. Our work is intended to support diagnostic analysis in sandboxed environments, and all experiments are confined to offline, self-hosted replicas with no external side effects. Details of LLM usage are provided in Appendix C.6.

Reproducibility Statement

We will publicly release the full benchmark, including all 10 website implementations, all 1,800 task definitions with world seeds and oracle trajectories, and the complete evaluation suite. Each environment is formalized as a deterministic MDP with a fixed world seed, so the environment dynamics, verification outcomes, and semantic traces are reproducible given the same agent actions. This guarantees reproducibility of the benchmark conditions, although full end-to-end run reproducibility may still be affected by model-side sources of non-determinism such as stochastic decoding or API variation. Agent hyperparameters, model identifiers, and decoding settings are reported in Table 3. Additional details on environment determinism and residual sources of non-determinism are provided in Appendix C.5.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, 2017.

Anthropic. Claude opus 4.6 system card, 2026. URL https://www-cdn.anthropic.com/14e4fb01875d2a69f646fa5e574dea2b1c0ff7b5.pdf.

Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. Fara-7b: An efficient agentic model for computer use. *arXiv preprint arXiv:2511.19663*, 2025.

Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. WorkArena++: Towards compositional planning and reasoning-based common knowledge work tasks. *arXiv preprint arXiv:2407.05291*, 2024.

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

De Chezelles, Thibault Le Sellier, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F Xu, Siva Reddy, Quentin Cappart, et al. The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024.

Jiwan Chung, Neel Joshi, Pratyusha Sharma, Youngjae Yu, and Vibhav Vineet. What mllms learn about when they learn about multimodal reasoning: Perception, reasoning, or their integration? *arXiv preprint arXiv:2510.01719*, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *NeurIPS*, 2023.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marber, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

Divyansh Garg, Shaun VanWeelden, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Tomas Abraham, Michael Lara, Federico Lopez, et al. Real: Benchmarking autonomous agents on deterministic simulations of real websites. *arXiv preprint arXiv:2504.11543*, 2025.

Milan Gritta, Debjit Paul, Xiaoguang Li, Lifeng Shang, Jun Wang, and Gerasimos Lampouras. Process evaluation for agentic systems. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 2678–2692, 2026.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: a benchmark for code reasoning, understanding and execution. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 16568–16621, 2024.

Tanmay Gupta et al. MolmoWeb: Open visual web agent and open data for the open web. Technical report, Allen Institute for AI, 2026. URL https://allenai.org/papers/molmoweb.

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

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pp. 2107–2116. PMLR, 2018.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *ICLR*, 2024.

Evan Zheran Liu, Kelvin Guo, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *ICLR*, 2024.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stanczak, Peter Shaw, Christopher Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. In *Second Conference on Language Modeling*, 2025.

Chang Ma, Junlei Zhang, Zhihao Liu, Jiawei Chen, Yitao Wang, et al. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *NeurIPS*, 2024.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: A benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.

OpenAI. Operator system card, 2025. URL https://cdn.openai.com/operator_system_card.pdf.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. WebCanvas: Benchmarking web agents in online environments. In *ICML*, 2024.

Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning. *Transactions on Machine Learning Research*, 2024.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy P Lillicrap. Androidinthewild: A large-scale dataset for android device control. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglong Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. WebWalker: Benchmarking LLMs in web traversal. *arXiv preprint arXiv:2501.07572*, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. In *COLM*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *NeurIPS*, 2022.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, Jitong Liao, Qi Zheng, Fei Huang, Jingren Zhou, and Ming Yan. Mobile-agent-v3: Fundamental agents for GUI automation. *arXiv preprint arXiv:2508.15144*, 2025.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. AssistantBench: Can web agents solve realistic and time-consuming tasks? *arXiv preprint arXiv:2407.15711*, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. In *ICLR*, 2024.

Appendix

Table of Contents

A Limitations . . . 15
B Related Work . . . 15
C Implementation Details . . . 17
C.1 Agent Configurations . . . 17
C.2 Compute and Runtime Statistics . . . 17
C.3 Evaluation Protocol . . . 17
C.4 Informational Coverage . . . 18
C.5 Reproducibility . . . 18
C.6 LLM Usage . . . 19
D Benchmark Specification . . . 20
D.1 Task Distribution . . . 20
D.2 Site Screenshots . . . 21
D.3 Per-Site MDP Specifications . . . 21
E Data Generation Pipeline . . . 25
E.1 Site Construction Pipeline . . . 25
E.2 World Generation . . . 27
E.3 Oracle Trajectory Generation . . . 27
F Extended Quantitative Results . . . 28
F.1 Per-site performance decomposition . . . 28
F.2 Per-site skill invocation . . . 29
F.3 Aggregate skill invocation . . . 29
F.4 Exploration SR by problem complexities . . . 29
G Artifact Viewer . . . 33
H Data Examples . . . 33
H.1 Task Template Catalog . . . 33
H.2 Example Trajectories: Agent vs. Oracle Trajectory . . . 35
H.3 MDP Surface Transition Graphs . . . 36
I Case Study . . . 41
I.1 Model Scale and Browsing Quality . . . 42
I.2 Premature Commitment to Hard Negative . . . 43
I.3 Thorough Exploration Before Commitment . . . 44
I.4 Exploration Success, Execution Failure . . . 45
I.5 Exploration Failure, Execution Success . . . 46
I.6 Redundant Process Despite Success . . . 47
I.7 Failure by Safeguard, Not by Competence . . . 48

A Limitations

Our benchmark prioritizes controllability, reproducibility, and trajectory-level diagnosis. These design choices improve measurement precision, but they also introduce limitations in realism, coverage, and action scope.

Semantic abstraction. Each site is implemented as a deterministic MDP that captures the task-relevant interaction structure of its real-world counterpart. As a result, the benchmark does not capture some properties of live websites, including dynamic content updates, session-dependent state, asynchronous loading, personalized recommendations, and third-party integrations. Agent behavior in these self-hosted environments may therefore differ from behavior on live websites with noisier layouts and less predictable dynamics.

Domain coverage. The benchmark spans ten web domains, but it does not cover every application type. In particular, domains involving sensitive data handling, complex authorization flows, or rich media interaction are not included. Still, the covered domains are diverse, and the benchmark is broader in scope and substantially larger in task count than existing alternatives (He et al., 2024; Zhou et al., 2024; Deng et al., 2023). We therefore view the remaining gap as a matter of incomplete coverage rather than narrow domain design.

Synthetic world data. World data is generated synthetically through seeded procedures. This improves reproducibility and allows controlled construction of hard negatives, but it may miss properties of organic real-world data such as long-tail popularity patterns, organically correlated attributes, or temporal drift.

Template-based tasks. Tasks are instantiated from structured templates with deterministic verification conditions. This does not capture the full ambiguity, underspecification, or open-endedness of real user requests. However, the loss is constrained by that the templates still span diverse task types and difficulty factors, while the resulting structure enables explicit control over complexity, scalable generation, and reliable aggregation across task families. This design is therefore necessary for systematic process-level evaluation at scale.

Skill invocation rather than skill success. Our skill-level analysis measures whether agents invoke the required action types, not whether each action succeeds in the agent's intended sense. A finer notion of per-action success would require reliable access to latent intent, which is not available from GUI traces alone and is difficult to infer faithfully from model-generated rationales. We therefore evaluate behavior in terms of the realized causal trajectory, which is the more robust and less assumption-dependent object of analysis.

Interaction modality. The current evaluation focuses on coordinate-based visual interaction, specifically clicking, typing, and scrolling. Other interaction modes, such as drag-and-drop, keyboard shortcuts, or direct URL navigation, are not included. This narrows the covered action space and may omit behaviors that matter in some real web settings.

B Related Work

Web agent benchmarks. Web agent benchmarks now span a broad range of environment regimes, from tightly controlled synthetic interfaces such as MiniWoB++ (Liu et al., 2018) and WebShop (Yao et al., 2022), to self-hosted realistic websites such as WebArena (Zhou et al., 2024) and VisualWebArena (Koh et al., 2024), to online and live-web settings such as Mind2Web (Deng et al., 2023), WebVoyager (He et al., 2024), Mind2Web-Live (Pan et al., 2024), Online-Mind2Web (Xue et al., 2025), and AssistantBench (Yoran et al., 2024). Related benchmarks further broaden this landscape to enterprise workflows and general computer-use settings, including WorkArena (Drouin et al., 2024), BrowserGym (Chezelles et al., 2024), and OSWorld (Xie et al., 2024). Despite this rapid progress in environment realism and scope, evaluation remains centered primarily on terminal outcomes such as task success, sometimes with partial-credit variants or key-node style intermediate checks. WEBSTEP

differs in focusing on built-in process instrumentation: each environment is co-developed with an explicit semantic MDP dual, enabling automatic process-level evaluation grounded in the environment's own state and transition structure.

Environment modeling and GUI-semantic bridging. Constructing faithful simulated environments for agent evaluation is a longstanding challenge. AndroidEnv (Toyama et al., 2021) and AndroidWorld (Rawles et al., 2025) wrap real mobile applications in RL-compatible interfaces, while OSWorld (Xie et al., 2024) provides full desktop VMs with scripted evaluators. These environments expose low-level interaction traces, but they do not provide an explicit semantic account of what each action reveals or accomplishes. A related line of work maps GUI events to higher-level action abstractions for training or evaluation (Rawles et al., 2023; 2025), but usually does so post hoc through heuristic alignment rather than through the environment construction itself. In the web domain, WebArena (Zhou et al., 2024), VisualWebArena (Koh et al., 2024), and REAL Evals (Garg et al., 2025) move toward sandboxed and more reproducible evaluation, but their evaluation remains decoupled from an explicit website semantics and is largely limited to terminal success. Our semantic MDP dual instead co-develops the GUI and the semantic model so that all task-relevant rendering is generated from semantic state, making the correspondence explicit by construction.

Process-level and intermediate evaluation. The limitations of outcome-only evaluation are well recognized across adjacent areas. In mathematical reasoning, process supervision and process reward models explicitly score intermediate reasoning steps rather than only final answers (Lightman et al., 2024). In code evaluation, benchmarks such as HumanEval assess functional correctness by executing generated programs against tests (Chen et al., 2021), while CRUXEval further targets code understanding and execution behavior rather than final output alone (Gu et al., 2024). In sequential decision making, the broader credit assignment problem has long been recognized as a central challenge in reinforcement learning (Pignatelli et al., 2024; Sutton & Barto, 1998), precisely because terminal rewards alone are often too sparse or too delayed to localize which decisions mattered. This has motivated methods that expose or reshape intermediate structure, including reward decomposition (Icarte et al., 2018) and hindsight-based relabeling (Andrychowicz et al., 2017). Chung et al. (2025) also considered decomposition of multimodal reasoning capacities. Our setting is analogous: terminal task success does not reveal whether a web agent failed to find the right target, failed to gather the decisive evidence, or failed only at the final action, which is why process-level evaluation is necessary.

In web agent evaluation, however, most existing analyses remain at the task level: AgentBench (Liu et al., 2024) and GAIA (Mialon et al., 2023) report performance by task category, separating tasks from one another but not analyzing behavior within a trajectory. Recent work has begun to introduce within-task intermediate evaluation. AgentBoard (Ma et al., 2024) measures progress rate via manually annotated subgoals, and Mind2Web-Live (Pan et al., 2024) defines key nodes along reference trajectories. These approaches demonstrate that binary success discards substantial signal, but their reliance on static human-defined checkpoints limits both scalability and robustness, particularly on live websites where interface changes can invalidate annotations.

Automated trajectory evaluation. Trajectory-level evaluation that avoids manual annotation has been explored through several paradigms. LLM-as-judge approaches (Lù et al., 2025; Gritta et al., 2026) use language models to score intermediate agent steps, offering flexibility but introducing stochastic evaluation noise and dependence on judge model capability. Rule-based evaluation in structured environments (Shridhar et al., 2021; Wang et al., 2022) enables deterministic assessment but typically operates at the level of task completion predicates rather than fine-grained process analysis. Our framework differs from both: because the evaluation environment is built on an explicit semantic MDP, process metrics (including exploration success, skill invocation, and trajectory bifurcation analysis) are derived deterministically from the MDP trace without requiring either human annotation or model-based judgment.

C Implementation Details

C.1 Agent Configurations

Agent	Model	Backend	Input	Action Space	Max Imgs	Temp.	Max Tokens
Fara	microsoft/Fara-7B	vLLM	Screenshot	Function call (coordinate)	3	0.0	800
UI-TARS	ByteDance-Seed/UI-TARS-1.5-7B	vLLM	Screenshot	Thought + Action (coordinate)	5	0.0	2048
GUI-Owl	mPLUG/GUI-Owl-1.5-8B	vLLM	Screenshot	Tool call (coordinate)	4	0.0	2048
Qwen3.5	Qwen/Qwen3.5-122B-FP8	vLLM	Screenshot	Thought + Action (coordinate)	5	0.7	2048
OpenAI CUA	GPT-5.4	OpenAI API	Screenshot	Built-in computer tool (coordinate)	–	–	–
Claude CUA	claude-sonnet-4-6	Anthropic API	Screenshot + DOM	Tool call (ref + coordinate)	3	–	–

Table 3: Agent configurations. **Input**: what the agent receives each turn. **Action space**: how the agent specifies actions. **Max imgs**: number of recent screenshots retained in context. Claude CUA follows Anthropic's DOM-augmented browser harness (ref-based targeting with coordinate fallback), whereas all other agents operate purely from pixels.

All agents interact with the browser through a shared viewport of 1440 × 900 pixels. Each episode is capped at a maximum of 50 turns; if the agent has not completed the task within this budget, the episode is terminated and scored as a failure. Between consecutive agent actions, a brief blocking period is enforced to allow page transitions and rendering to complete before the next screenshot is captured.

We evaluate the following agents in WEBSTEP:

- **UI-TARS** Qin et al. (2025): A vision–language model for GUI interaction developed by ByteDance.[2]
- **Fara** Awadallah et al. (2025): A 7B web agent from Microsoft Research, fine-tuned on Qwen2.5-VL.[3]
- **GUI-Owl** Ye et al. (2025): A multimodal agent from Alibaba mPLUG. We use the 8B-model. [4]
- **Qwen3.5** Yang et al. (2025): Alibaba's Qwen3.5-122B model, quantized to FP8.[5]
- **OpenAI CUA** OpenAI (2025): OpenAI's Computer-Use Agent, accessed via the OpenAI API. We implemented the CUA-agent following the official github repository. [6]
- **Claude CUA**: Anthropic's computer-use agent (claude-sonnet-4-6), accessed via the Anthropic API. Following Anthropic's browser-use reference implementation, it receives each screenshot together with a list of interactive DOM elements and targets elements by reference id, falling back to coordinates when needed.[7]

C.2 Compute and Runtime Statistics

All GPU-based agents (Fara, UI-TARS, GUI-Owl, and Qwen3.5-122B) are served via vLLM on a single node with 4× NVIDIA H200 GPUs (140 GB each). The full benchmark evaluation across all GPU models completes in approximately 3–4 days of wall-clock time. The closed-source agents (OpenAI CUA and Claude CUA) use their respective vendor APIs and each complete within a day with 8 parallel workers.

C.3 Evaluation Protocol

Each task episode proceeds as follows. First, a headless Chromium browser instance is launched and initialized with the task's world data and starting state. The agent then enters

[2] https://github.com/bytedance/ui-tars
[3] https://github.com/microsoft/fara
[4] https://github.com/x-plug/mobileagent
[5] https://github.com/QwenLM/Qwen-Agent
[6] https://github.com/openai/openai-agents-python
[7] https://github.com/anthropics/anthropic-quickstarts

a turn-based loop: at each turn, the runner captures a screenshot of the current browser viewport, constructs an observation (including the screenshot, the current page URL, and history of previous conversation), and passes it to the agent. The agent returns an action (e.g., a click at a coordinate, a text input, or a scroll) which the runner executes in the browser. This loop continues until the agent signals task completion, the maximum turn limit of 50 is reached, or the agent reports the task as infeasible.

Upon episode termination, the verifier evaluates the stored semantic MDP trajectory against the task's verifier conditions. These conditions check the application's terminal semantic state for example, whether a specific item has been added to a cart, a message has been starred, or a booking has been confirmed with the correct parameters. An episode is scored as successful only if all verifier conditions are satisfied.

Additionally, we observed that some models (e.g., Fara) consistently fail to execute the final commit action even after correctly identifying the target entity. To ensure fair comparison, we adopt a *safe pass* policy: if the agent reaches the critical decision point, i.e., it has navigated to the correct entity and gathered sufficient information to act, but does not issue the final commit, the episode is still counted as a success for the terminal success rate. This prevents penalizing models whose failure is limited to the mechanical execution of a single concluding action rather than a deficiency in exploration or reasoning. All terminal success rates reported in this paper are measured under the safe pass criterion.

C.4 Information Coverage

Information coverage measures how much task-relevant evidence an agent has gathered by the time it commits. It is a graded metric of evidence acquisition, not a binary test of whether the agent observed everything that could have been useful. For each task, the benchmark derives a set of information constraints $\mathcal{C} = \{c_1, \ldots, c_K\}$ from the task's information gap, template type, and reasoning type using per-site declarative rules. This set is best understood as a conservative upper bound on useful information: it includes fields, entities, and comparisons that may support the decision, and agents can succeed while observing less (e.g., by narrowing candidates through search). At the same time, the rules are calibrated against the ground-truth trajectories – the oracle solution observes every constraint before committing – so the set never demands information that a correct solution would not gather. A small set of tasks (226 of 1,800) satisfies all constraints from the initial observation alone; coverage is non-discriminative on this subset by construction.

Each site defines a visibility specification that maps every interface surface σ to the set of item attributes rendered on that surface. Some attributes are visible on list-level surfaces, while others become visible only after opening a detail page. A detail-only constraint can also be satisfied through a search operation: when the agent issues a query containing the task's discriminating value and the target remains in the results, the fields that the site's search function scans count as verified (the match is disjunctive over the scanned fields, so this credit is conservative and value-guarded). As the agent follows a semantic trajectory $\tau = (s_0, a_0, \ldots, s_T)$, the benchmark tracks cumulatively which task-relevant constraints have become satisfied. Coverage at step t is then defined as

$$\mathrm{COV}_t = \frac{|\{c \in \mathcal{C} : \mathrm{sat}(c, t)\}|}{|\mathcal{C}|}, \tag{2}$$

where $\mathrm{sat}(c, t)$ indicates whether constraint c has been satisfied by the information observed up to step t.

We report coverage at the commit step, t_{commit}, defined as the first step at which the agent executes a terminal action such as `PlaceOrder` or `done`. The reported metric, $\mathrm{COV}_{t_{\text{commit}}}$, therefore measures how much of this benchmark-defined task-relevant evidence the agent had gathered when it chose to act.

C.5 Reproducibility

We take the following steps to ensure reproducibility:

Code and data release. We will publicly release the complete benchmark codebase, including all 10 site implementations, the task generation pipeline, the evaluation runner, and the metric computation scripts. All 1,800 task definitions (with world seeds, instructions, oracle trajectories, and verifier conditions) will be included in the release.

Environment determinism.

- **Fixed world seeds.** Each task is paired with a deterministic random seed that fully determines the world data (user profiles, item catalogs, message contents, etc.), ensuring identical evaluation conditions across runs.
- **Deterministic MDPs.** All site implementations are pure deterministic MDPs: the same state–action pair always produces the same next state and observation, eliminating environmental stochasticity.

Agent hyperparameters. All agent hyperparameters are reported in Table 3 (Section B.1). Key settings:

- **Greedy decoding.** All open-weight agents (Fara, UI-TARS, GUI-Owl) are served with temperature 0, ensuring deterministic outputs given identical inputs. Qwen3.5 uses temperature 0.7, top-p 0.8, top-k 20 to follow official implementation.
- **Episode budget.** All agents are given a maximum of 50 GUI turns per task.
- **Viewport.** All agents share a 1440×900 browser viewport.

Non-determinism. For the closed-source OpenAI CUA and Claude CUA agents, exact reproducibility cannot be guaranteed due to potential server-side non-determinism in the API; however, we observe low variance across repeated runs.

C.6 LLM Usage

We use LLMs in three aspects of this work. First, in the benchmark construction pipeline, we use Claude Opus 4.6 as a coding subroutine to help convert recorded website interaction traces into semantic MDP specifications and to build the executable website implementations from those specifications, as described in Section E. This process involved substantial manual validation, editing, and re-iteration between steps; the LLM served as an accelerator within a human-driven development loop rather than as an autonomous construction agent. Second, the web agents evaluated in our experiments (Section 4) are themselves LLM-based systems; their use constitutes the object of study rather than a methodological choice. Third, LLMs assisted with result visualization by providing initial seeds for plotting code and performing instruction-based edits; all figures were subsequently hand-edited in both code and output. In all three cases, the authors reviewed and verified the outputs. No LLM was used to originate research ideas or generate experimental data. LLMs provided assistance with prose editing during drafting.

D Benchmark Specification

D.1 Task Distribution

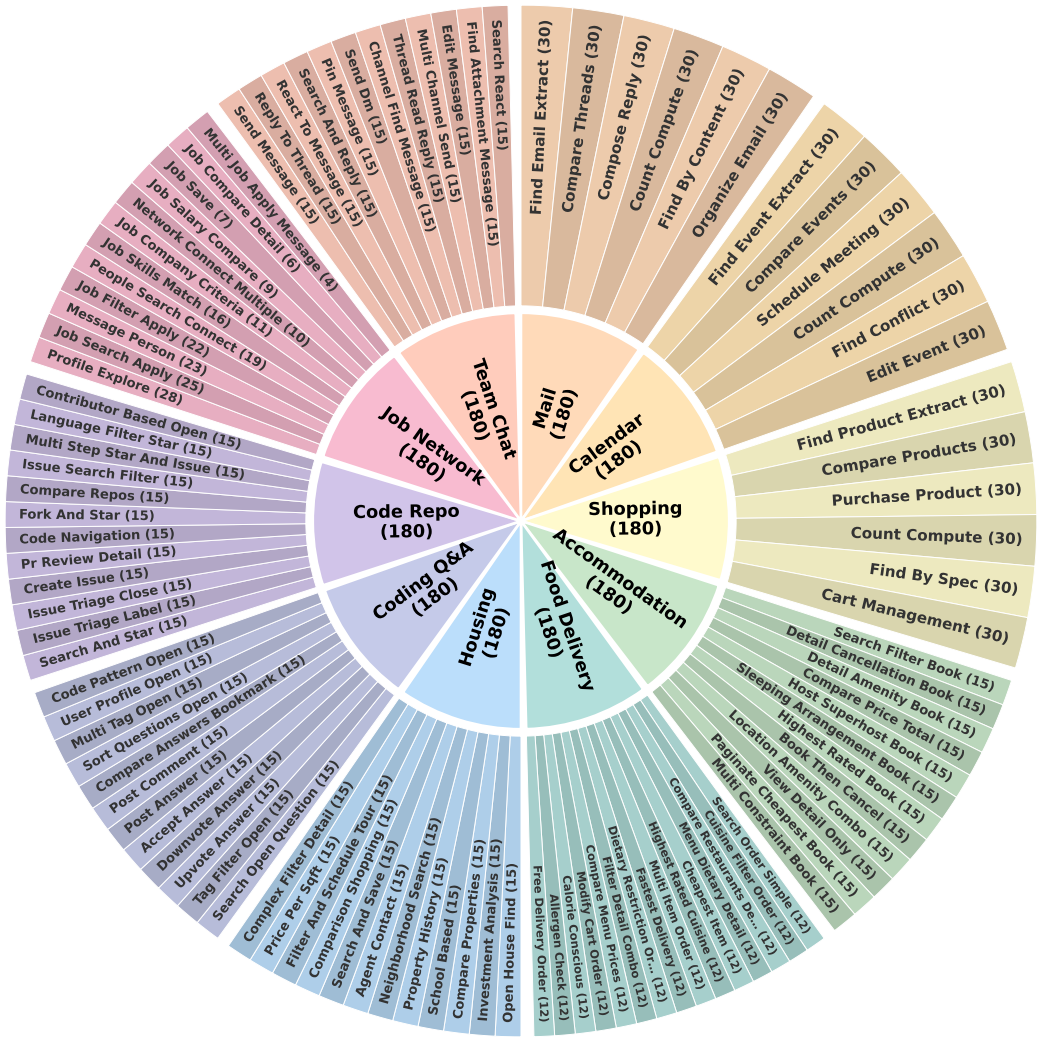

Figure 7: **WEBSTEP task distribution.** The inner ring shows 10 websites (180 tasks each). The outer ring shows task templates with instance counts. Templates range from 4 to 30 instances per template, reflecting the diversity of interaction patterns within each site.

Figure 7 visualizes the hierarchical task distribution across the ten sites in WEBSTEP. Each site contributes 180 tasks instantiated from multiple templates, yielding a total of 1,800 tasks spanning diverse interaction patterns and difficulty levels.

D.2 Site Screenshots

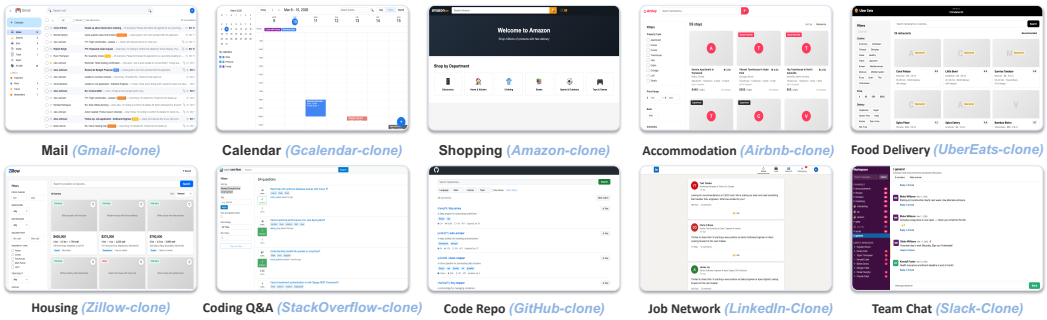

Figure 8: Screenshots of all 10 WEBSTEP websites. **Top row**: Mail, Calendar, Shopping, Accommodation, Food Delivery. **Bottom row**: Housing, Coding Q&A, Code Repository, Job Network, and Team Chat. Each site is a standalone HTML/CSS/JS application with full MDP instrumentation.

Figure 8 shows representative screenshots from each of the ten sites in WEBSTEP. The sites span a broad range of web application categories–from e-mail and calendaring to e-commerce, social networking, and developer tools–each with distinct visual layouts, navigation patterns, and interaction paradigms. Each site's visual layout and interaction flow is modeled after a representative real-world web application in the same domain.

D.3 Per-Site MDP Specifications

State decomposition. Each site in WEBSTEP is modeled as a deterministic MDP whose state is decomposed into a set of typed variables. These variables capture the current navigation surface (e.g., inbox, search results, item detail), UI component states (e.g., which dropdown is open, current scroll position, active filters), and application-level data (e.g., selected items, form field contents, user preferences). Table 4 summarizes the number of surfaces, state variables, actions, and tasks for each site.

Visibility partitioning. Not all state variables are visible on every surface. Table 5 specifies the visibility partition: for each site, it lists which state variables are observable on each surface. This partition determines what information is available to the agent at any given point in the episode and is a key factor in task difficulty: tasks that require reasoning about state variables only visible on other surfaces necessitate multi-step navigation.

Skill taxonomy. Table 6 defines the five skill categories used throughout the benchmark. Table 7 then lists the full set of skills for each site, organized by category. Each task is labeled with the set of skills it exercises, enabling fine-grained analysis of agent capabilities across navigation, search, inspection, and execution dimensions. The four exploration categories are global action-type sets, whereas *commit* is task-relative: each task's commit is the final task-completing step of its oracle trajectory, performed on the task's target entity.

Site	Domain	Surfaces	Actions	Templates	Tasks	Card / Detail Fields	Decision Object
Mail	Productivity	3	36	6	180	10 / 9	Thread
Calendar	Productivity	6	14	6	180	8 / 6	Event
Team Chat	Productivity	3	24	12	180	6 / 5	Message
Shopping	E-Commerce	6	23	6	180	12 / 12	Product
Food Delivery	E-Commerce	6	24	15	180	8 / 7	Restaurant / Item
Accommodation	Discovery	4	16	12	180	9 / 8	Listing
Housing	Discovery	4	16	12	180	8 / 10	Property
Coding Q&A	Information	4	18	12	180	7 / 6	Question
Code Repo	Collaboration	6	32	12	180	9 / 8	Repository / Issue
Job Network	Social	6	26	12	180	7 / 9	Job / Profile
Total	–	48	229	105	**1,800**	84 / 80	–

Table 4: Per-site MDP statistics. **Surfaces**: distinct UI views (pages, modals). **Actions**: typed semantic action count. **Templates**: task template count. **Tasks**: total task instances. **Card fields**: attributes visible on list views. **Detail fields**: attributes requiring detail-page navigation.

Site	Card Surfaces	Detail Surfaces	Commit Surfaces	$\lvert\mathcal{V}_{\text{card}}\rvert$	$\lvert\mathcal{V}_{\text{detail}}\rvert$
Mail	ThreadList	ThreadView	ComposeModal, ThreadList, ThreadView	10	9
Calendar	WeekView, DayView, MonthView	EventDetail	EventEditor	8	6
Team Chat	ChannelView, SearchResults	ChannelView (thread)	ChannelView, DirectMessageView	6	5
Shopping	SearchResults	ProductDetail	Checkout, OrderConfirmation	12	12
Food Delivery	RestaurantList	RestaurantDetail, MenuItemDetail	Checkout, OrderStatus	8	7
Accommodation	SearchResults	ListingDetail	Checkout, Reservations	9	8
Housing	SearchResults	PropertyDetail	PropertyDetail, SavedHomes	8	10
Coding Q&A	QuestionList	QuestionDetail	QuestionDetail	7	6
Code Repo	RepoList, IssueList, PullRequestList	RepoDetail, IssueDetail, PRDetail	IssueDetail, RepoDetail	9	8
Job Network	JobList, ConnectionList	JobDetail, ProfileView	JobDetail, Messaging	7	9

Table 5: Visibility partition for each site. $|\mathcal{V}_{\text{card}}|$: attributes visible on list surfaces. $|\mathcal{V}_{\text{detail}}|$: attributes visible only on detail surfaces. Surfaces are categorized as **Card** (list views), **Detail** (entity detail pages), and **Commit** (surfaces where task-completing actions are available).

Category	Description
Navigate	Actions that move between surfaces or manage the current view without modifying application state. Includes pagination, folder/tab switching, back navigation, closing detail views, and temporal navigation (e.g., next/previous week).
Search	Actions that issue or clear search queries to change the visible entity set. Includes keyword search, people search, repository search, and clearing search context.
Filter	Actions that narrow or reorder the visible entities within the current result set. Includes applying categorical or range filters, sorting, and clearing filters.
Inspect	Actions that reveal additional information about an entity without modifying application state. Includes opening detail views, expanding hidden content, viewing profiles, and selecting entities for closer examination.
Commit	The task-completing action: the final step of the task's oracle trajectory, performed on the target entity—e.g., placing the order, sending the reply, starring the target thread, or, for read-only tasks, opening the target item. State-modifying actions that are not the task's commit (e.g., adding an item to the cart before placing the order, or filling a form before sending) form the task's *pre-commit sequence*; in trajectory analyses these steps are merged into the step that follows them rather than counted as a separate category.

Table 6: Skill category definitions. The four exploration categories (Navigate, Search, Filter, Inspect) are global action-type sets; *Commit* is task-relative, defined per task as the final step of its oracle trajectory.

Site	Category	Skills
Mail	Navigate	folder_navigation, pagination, backtrack_navigation
	Search	search_refinement
	Filter	filter_application
	Inspect	thread_inspection, message_expansion
	Commit	compose_setup, compose_field_entry, email_management, bulk_management, send_commit
Calendar	Navigate	temporal_navigation, view_switching, backtrack_navigation
	Search	search_query
	Filter	calendar_filtering
	Inspect	event_inspection, information_extraction, comparison, conflict_detection
	Commit	event_creation, event_editing, field_entry, commit_timing
Shopping	Navigate	pagination
	Search	query_formulation
	Filter	department_filtering, category_filtering, brand_filtering, price_filtering, attribute_filtering, sort_usage
	Inspect	product_inspection, review_inspection, comparison
	Commit	option_selection, cart_management, checkout_flow, commit_timing
Accommodation	Navigate	pagination, search_return
	Search	query_formulation
	Filter	property_type_filtering, price_filtering, beds_filtering, amenity_filtering, sort_usage
	Inspect	listing_inspection, amenity_expansion, comparison
	Commit	booking_initiation, booking_confirmation, booking_cancellation, checkout_cancellation
Food Delivery	Navigate	pagination
	Search	query_formulation
	Filter	cuisine_filtering, price_filtering, dietary_filtering, attribute_filtering, sort_usage
	Inspect	restaurant_inspection, menu_item_inspection, comparison
	Commit	customization_selection, cart_management, checkout_flow, commit_timing

Table 7: Skill taxonomy per site (Part 1 of 2). Skills are organized into five categories: **Navigate** (surface traversal), **Search** (query formulation), **Filter** (attribute filtering and sorting), **Inspect** (entity viewing), and **Commit** (task-completing actions).

Site	Category	Skills
Housing	Navigate	pagination
	Search	query_formulation
	Filter	price_filtering, beds_filtering, baths_filtering, sqft_filtering, type_filtering, attribute_filtering, sort_usage
	Inspect	property_inspection, agent_inspection, comparison
	Commit	save_management, tour_scheduling, agent_contact, saved_homes_review
Coding Q&A	Navigate	back_navigation, pagination
	Search	query_formulation
	Filter	tag_filtering, acceptance_filtering, date_filtering, vote_filtering, sort_usage, answer_sort_usage, filter_reset
	Inspect	question_inspection, user_inspection, tag_inspection
	Commit	voting, answer_posting, commenting, answer_acceptance, bookmarking
Code Repo	Navigate	navigate_to_issues, navigate_to_prs, back_navigation, pagination
	Search	repo_search
	Filter	language_filtering, stars_filtering, license_filtering, topic_filtering, repo_attribute_filtering, repo_sort_usage, issue_state_filtering, issue_label_filtering, issue_author_filtering, issue_sort_usage, pr_filtering, pr_sort_usage
	Inspect	repo_inspection, issue_inspection, pr_inspection
	Commit	issue_creation, issue_commenting, issue_assignment, issue_labeling, issue_state_change, star_management, commit_timing
Job Network	Navigate	tab_navigation, pagination
	Search	job_query_formulation, people_query_formulation
	Filter	location_filtering, job_type_filtering, remote_filtering, salary_filtering, experience_filtering, sort_usage, filter_clearing
	Inspect	job_inspection, profile_inspection, job_comparison, profile_comparison
	Commit	job_application, connection_request, messaging, job_save_action, conversation_management, feed_interaction
Team Chat	Navigate	channel_navigation, dm_navigation, scroll_navigation, search_result_navigation
	Search	search_query, search_clear
	Filter	—
	Inspect	thread_inspection, member_inspection, thread_close, member_close
	Commit	send_channel_message, send_thread_reply, send_dm, message_reaction, message_pin, message_edit, message_delete, channel_mute, commit_timing

Table 8: Skill taxonomy per site (Part 2 of 2). See Table 6 for category definitions.

E Data Generation Pipeline

E.1 Site Construction Pipeline

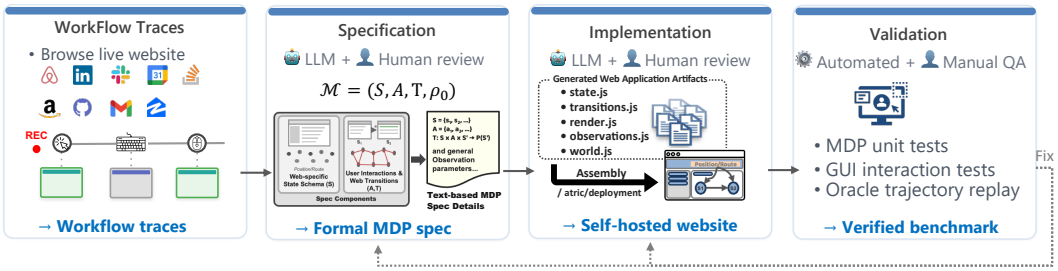

Figure 9: **Overview of the website pipeline.**

Each site in WEBSTEP is developed through an iterative human-in-the-loop pipeline. An LLM coding agent (Claude Opus 4.6 Anthropic (2026)) carries out much of the implementation at each stage, while all intermediate outputs are manually reviewed and refined before the next stage. The process is iterative rather than strictly linear: when downstream validation reveals issues, earlier stages are revisited and updated accordingly.

Phase 1: Grounded exploration. We begin by interacting with representative user workflows on the live target website, including searching, filtering, inspecting items, and completing final actions, while recording user actions and screenshots at each step. These traces provide grounded evidence for the subsequent specification stage. All observations are derived solely from user-visible interaction, namely the screenshot and user action at each step, without relying on backend code, API traffic, or other non-GUI signals.

Phase 2: Specification. The coding agent generalizes the recorded traces into a formal semantic MDP specification through the following staged procedure. Each stage must satisfy its validation criteria before the pipeline proceeds.

The state schema refines the main text's positional/informational decomposition: interface, query, and session state together form the positional state, while revealed state corresponds to the informational state.

Algorithm 1 details the steps for generating MDP specification from the Workflow-traces of Phase 1.

Phase 3: Website implementation. The specification is then implemented as a self-contained single-page web application whose visible GUI is a rendered view of the underlying semantic MDP state, with minor exceptions on non-semantic actions:

- `state.js`: State schema and `createInitialState(world)` initializer.
- `transitions.js`: Pure transition function `transition(state, action, world)` →{`state`, `error?`}, with one case per typed action and precondition checking.
- `observations.js`: Pure observation function `observe(state, world)` →`observation`, resolving visible entities, applying surface-specific visibility rules, and computing derived values.
- `world.js`: Seeded pseudo-random generator `generateWorld(seed)` producing a complete entity catalog with cross-entity consistency enforced by construction.
- `render.js`: DOM rendering as a pure projection of the observation. Every interactive element is bound to `dispatch(action)` calls via standardized `data-test-id` attributes.

Thus, GUI interactions are executed at the coordinate level, but every task-relevant interaction is mapped to a typed semantic action through the dispatch layer. The entry page wires these modules into a dispatch loop: each user interaction calls `dispatch`, which runs

the transition, recomputes the observation, and re-renders the page. Every dispatch also persists the current state and the full action trajectory to `localStorage`.

Algorithm 1 Specification construction from grounded traces

Require: Workflow traces from Phase 1
Ensure: Structured MDP specification
1: Identify domain entities with typed attributes and unique identifiers
2: Define relations between entities with explicit cardinalities
3: Map surfaces, including visible entities, displayed fields, and available actions
4: Construct the navigation graph and verify surface reachability
5: Define the state schema:
    interface state, query state, session state, revealed state
6: Specify the observation function from state to visible fields and available actions
7: Define each action with parameters, preconditions, state updates, reveals, and failure cases
8: Specify filtering, sorting, pagination, and the deterministic query function
9: Define flow guards for sequential dependencies and stage-locked fields
10: Specify all derived display values as deterministic functions of state and world data
11: Define the rendering contract, including layout templates, rendering rules, DOM bindings, and style parameters
12: Define the world generator, including field distributions, consistency constraints, variation axes, and invariants
13: Run specification verification:
    coverage matrix
    reachability
    state liveness
    constraint binding
    observation sufficiency
    world generator consistency
14: **if** any check fails **then**
15: Return to the relevant earlier stage and revise
16: **end if**
17: Compile the final structured specification for the implementation stage

Multi-level validation. Each site is validated at four complementary levels:

1. **MDP unit tests** (`validate.js`). These tests validate the semantic model in isolation, including:
   - correct execution of happy-path trajectories,
   - rejection of invalid actions,
   - surface-specific observation visibility,
   - correctness of derived values,
   - navigation and back-stack behavior,
   - filter, sort, and search behavior, and
   - deterministic but varied world generation across seeds.
2. **GUI interaction validation** (`validate_interactions.py`). Playwright-based browser tests verify:
   - discovery of all instrumented `data-test-id` elements,
   - coordinate-level interactability without occlusion,
   - execution of interactions through raw mouse and keyboard actions,
   - agreement between GUI interactions and expected MDP state transitions, and
   - absence of unsupported controls such as native `<select>` elements.
3. **GUI trajectory replay** (`verify_gui.py`). End-to-end replay of every oracle trajectory in headless Chromium checks:
   - that every semantic action produces a real state change,
   - that the full oracle trajectory executes without error, and
   - that the final state satisfies the task verifier.

4. **Manual QA iteration**. Early agent runs are manually inspected to identify residual issues in:
   - verifier logic,
   - world-generation ambiguities, and
   - verifier-specification gaps.

   All identified issues are fixed, propagated back through the pipeline as needed, and re-verified.

E.2 World Generation

Each task operates over a *world*: a self-contained data environment that populates the site with concrete entities (e.g., email threads, product listings, calendar events). Worlds are generated through the following procedure:

Target entity creation. For each task, a target entity is generated that satisfies the task's constraint set. The target's attributes are sampled to be consistent with the natural language instruction the agent will receive.

Hard negative injection. To calibrate task difficulty, a controlled number of *hard negative* entities are generated. These entities share surface-level attributes with the target (e.g., same category, similar name, overlapping tags) but differ on the specific attributes referenced in the task constraints. The number of hard negatives directly affects the agent's required discrimination effort. In particular, hard negatives are constructed to remain ambiguous under shallow evidence, often matching the target on card- or filter-level attributes while differing only on decisive detail-level attributes. Templates whose instruction identifies the target directly (e.g., by its exact name) involve no identification ambiguity and plant no hard negatives; their hard-negative count is zero.

Filler population. The remaining entity slots are populated with randomly sampled filler entities that provide a realistic background distribution. Fillers are generated to be clearly distinguishable from the target, ensuring they do not introduce ambiguity.

Deterministic seeding. All randomness in world generation is governed by a single integer seed per task. Given the same seed, the generation procedure produces an identical world, ensuring full reproducibility.

E.3 Oracle Trajectory Generation

Each task in WEBSTEP is accompanied by an oracle trajectory: a minimal-length sequence of semantic MDP actions that achieves the task goal from the initial state.

Generation pipeline. Oracle trajectories are produced by a template-based deterministic construction. Each task template defines a surface-visit plan (e.g., SearchResults → ProductDetail → Checkout) and emits the corresponding action sequence in the space of typed semantic actions rather than pixel coordinates. The pipeline proceeds in three stages:

1. **Navigation to target region**: The generator applies any necessary search queries, filters, or pagination to reach the list surface containing the target entity.
2. **Hard-negative exploration**: For tasks with comparison requirements, the oracle visits top-ranked hard negatives (up to two) in ranking order (e.g., opening a distractor's detail page, inspecting the distinguishing attribute, and returning to the list) before opening the target entity last. The planted hard-negative count is therefore a designed lower bound on the discrimination effort a task demands, rather than the exact number of detail visits the oracle performs.
3. **Commit**: The generator emits the final task-specific actions (e.g., add to cart, star, upvote) to satisfy the verifier conditions.

Optimality, non-uniqueness, and validation. Optimality is guaranteed by construction. Each template specifies the shortest path in terms of surface visits, and the visitation order over hard negatives is fixed by the ranking. However, multiple valid solutions may still exist, including alternatives with the same path length. To verify correctness, every generated trajectory is replayed through the site's deterministic transition function, and the final state is checked against all verifier conditions. This guarantees that every benchmark task is solvable.

Purposes. Oracle trajectories serve three roles: (i) they provide ground-truth supervision for process-level metrics (e.g., coverage computation, skill scoring); (ii) they establish an upper bound on task efficiency (optimal step count); and (iii) they are used during validation to verify that every task is solvable within the MDP.

F Extended Quantitative Results

F.1 Per-site performance decomposition

Table 2 reports aggregate metrics. Tables 9 to 12 provide the full per-site breakdown.

Agent	Mail	Cal.	Shop.	Acco.	Food	Hous.	Q&A	Code	Jobs	Chat
Claude CUA	**91.7**	**79.4**	**82.8**	87.2	**86.1**	**79.4**	77.8	**91.1**	84.4	**93.3**
OpenAI CUA	89.4	73.3	78.9	**88.3**	81.1	66.7	**79.4**	88.3	**91.1**	90.0
Qwen3.5-122B	60.6	52.2	56.1	53.9	73.9	25.6	63.9	65.6	66.7	63.9
Fara	50.0	16.1	37.8	31.1	63.9	13.3	37.2	41.1	40.0	26.1
GUI-Owl	58.3	10.6	12.8	11.7	67.2	20.0	50.6	43.3	35.6	33.9
UI-TARS	71.1	18.9	16.7	8.9	65.6	27.2	46.1	36.7	48.3	31.1

Table 9: Terminal Success Rate (%) by site. Best per site in bold.

Agent	Mail	Cal.	Shop.	Acco.	Food	Hous.	Q&A	Code	Jobs	Chat
Claude CUA	**91.7**	**79.4**	**98.9**	**93.3**	**97.8**	**82.8**	85.6	**91.1**	**93.3**	**93.3**
OpenAI CUA	90.0	77.8	95.6	91.7	97.2	70.0	**89.4**	88.3	92.8	92.2
Qwen3.5-122B	62.8	58.9	71.1	70.0	91.7	27.8	78.9	65.0	67.8	66.7
Fara	53.3	22.2	47.8	52.2	82.2	13.9	52.2	46.1	43.9	55.6
GUI-Owl	58.9	13.3	13.3	47.8	82.2	22.2	60.0	50.6	34.4	47.2
UI-TARS	76.1	26.7	20.0	51.1	82.8	28.9	55.6	41.7	53.3	53.3

Table 10: Exploration Success Rate (%) by site. Best per site in bold.

Agent	Mail	Cal.	Shop.	Acco.	Food	Hous.	Q&A	Code	Jobs	Chat
Claude CUA	**100.0**	**100.0**	83.7	93.5	**88.1**	**96.0**	**89.6**	**100.0**	90.5	**100.0**
OpenAI CUA	99.4	94.3	82.6	**96.4**	83.4	95.2	88.8	**100.0**	97.6	97.6
Qwen3.5-122B	94.7	88.7	78.9	77.0	80.6	80.0	79.6	**100.0**	96.7	95.8
Fara	93.8	72.5	79.1	59.6	77.0	92.0	70.2	89.2	91.1	47.0
GUI-Owl	99.1	79.2	**95.8**	24.4	81.8	77.5	84.3	85.7	**100.0**	71.8
UI-TARS	93.4	70.8	83.3	17.4	77.9	82.7	82.0	88.0	87.5	58.3

Table 11: Execution SR | Exploration Success (%) by site. Best per site in bold.

Agent	**Mail**	**Cal.**	**Shop.**	**Acco.**	**Food**	**Hous.**	**Q&A**	**Code**	**Jobs**	**Chat**
Claude CUA	91.3	96.7	**100.0**	95.5	97.8	**100.0**	93.3	**90.9**	98.4	**99.1**
OpenAI CUA	88.7	**97.2**	99.6	**97.3**	**98.9**	98.9	**98.9**	90.9	**98.4**	98.7
Qwen3.5-122B	83.3	94.4	97.2	83.7	93.4	94.4	88.6	68.8	92.7	94.1
Fara	85.0	80.6	93.0	76.6	89.2	62.9	65.3	53.9	76.1	81.7
GUI-Owl	**93.9**	76.4	74.0	78.5	80.7	96.2	80.6	55.8	72.0	86.8
UI-TARS	84.9	83.1	79.6	81.7	82.5	82.2	81.9	52.5	87.4	86.9

Table 12: Informational Coverage at Commit (%) by site. Best per site in bold.

F.2 Per-site skill invocation

Tables 13 and 14 report skill invocation rates by site. Each cell shows the percentage of episodes requiring that skill where the agent invoked it. *Commit* is task-relative: it denotes the final task-completing action of each task's oracle trajectory, and counts as invoked only when the agent's trajectory ends with that action.

F.3 Aggregate skill invocation

Table 15 reports overall skill invocation rates across all sites.

F.4 Exploration SR by problem complexities

Table 16, Table 17, and Table 18 report the numeric values underlying Figure 6 (a), (b), and (c), respectively.

Agent	Site	Search	Filter	Inspect	Navigate	Commit
Claude CUA	Mail	94	–	88	70	**79**
	Cal.	–	–	**100**	92	**81**
	Shop.	**100**	21	**100**	89	**99**
	Acco.	**97**	99	**100**	71	**97**
	Food	70	85	**100**	63	76
	Hous.	**99**	**100**	**91**	69	**98**
	Q&A	86	84	92	**73**	82
	Code	91	**100**	**100**	61	**93**
	Jobs	92	**100**	**100**	**100**	81
	Chat	**100**	–	**100**	**100**	**98**
OpenAI CUA	Mail	**100**	–	86	62	66
	Cal.	–	–	**100**	**92**	67
	Shop.	**100**	21	**100**	**90**	94
	Acco.	**97**	69	**100**	**77**	91
	Food	**77**	86	**100**	**68**	**81**
	Hous.	98	84	88	**73**	87
	Q&A	91	85	**99**	43	**85**
	Code	**93**	90	**100**	**65**	83
	Jobs	92	97	**100**	**100**	**95**
	Chat	**100**	–	**100**	84	89
Qwen3.5-122B	Mail	86	–	86	76	47
	Cal.	–	–	**100**	89	54
	Shop.	99	35	99	85	72
	Acco.	87	85	96	57	44
	Food	71	86	99	61	48
	Hous.	97	**100**	87	63	44
	Q&A	90	**91**	93	**73**	61
	Code	82	93	99	39	74
	Jobs	**99**	**100**	99	**100**	64
	Chat	**100**	–	99	**100**	76

Table 13: Skill invocation rates (%) by site and agent (Part 1 of 2). ”–” = fewer than 20 required episodes. *Commit* is task-relative: the final task-completing action of each task’s oracle trajectory. Best value per site–skill cell across all six agents in bold.

Agent	**Site**	**Search**	**Filter**	**Inspect**	**Navigate**	**Commit**
Fara	Mail	97	–	89	41	39
	Cal.	–	–	88	63	21
	Shop.	99	71	88	64	35
	Acco.	77	83	88	53	6
	Food	63	81	96	40	23
	Hous.	69	81	71	43	21
	Q&A	77	53	70	23	29
	Code	71	58	93	32	51
	Jobs	88	58	68	94	38
	Chat	93	–	88	94	31
GUI-Owl	Mail	56	–	**93**	**82**	67
	Cal.	–	–	35	35	11
	Shop.	**100**	**85**	58	46	23
	Acco.	91	**100**	75	22	15
	Food	62	**90**	97	55	26
	Hous.	**99**	**100**	70	48	38
	Q&A	89	**91**	87	57	60
	Code	81	90	97	10	55
	Jobs	63	**100**	79	**100**	48
	Chat	80	–	73	**100**	54
UI-TARS	Mail	95	–	81	73	54
	Cal.	–	–	62	52	17
	Shop.	99	56	60	49	13
	Acco.	89	**100**	88	44	3
	Food	68	85	99	64	19
	Hous.	87	97	74	52	19
	Q&A	**96**	59	84	43	41
	Code	87	42	93	42	43
	Jobs	94	90	88	98	36
	Chat	97	–	84	98	32

Table 14: Skill invocation rates (%) by site and agent (Part 2 of 2). ”–” = fewer than 20 required episodes. *Commit* is task-relative: the final task-completing action of each task’s oracle trajectory. Best value per site–skill cell across all six agents in bold.

Agent	Search	Filter	Inspect	Navigate	Commit
Claude CUA	92.9	87.8	97.1	**80.9**	**88.3**
OpenAI CUA	**94.5**	78.9	**97.2**	79.7	83.8
Qwen3.5-122B	90.9	87.2	95.5	76.3	58.5
Fara	80.4	70.9	82.8	60.3	29.4
GUI-Owl	83.3	**94.0**	78.6	58.1	39.7
UI-TARS	90.5	77.3	82.9	65.2	27.7

Table 15: Aggregate skill invocation rates (%). Each cell shows the fraction of episodes requiring that skill where the agent invoked it. *Commit* invocation requires the trajectory to end with the task’s commit action. Best per column in bold.

Agent	**HN=0**	**HN=1**	**HN=2**	**HN=3**
Claude CUA	90.6	**98.7**	**90.3**	**89.8**
OpenAI CUA	**91.7**	93.6	87.9	83.5
Qwen3.5-122B	76.4	78.2	65.2	51.4
Fara	60.5	64.1	41.9	32.8
GUI-Owl	59.4	43.6	34.5	32.3
UI-TARS	60.6	53.8	45.6	37.3

Table 16: Exploration SR (%) by hard negative count (the number of card-level confusables planted by the generator; a designed lower bound). Best per column in bold. HN=0 tasks have no identification ambiguity but vary on other difficulty axes, so monotone trends should be read over HN$\geq$1.

Agent	1	2	3	4	5	6	7	8	9	10
Claude CUA	**96.3**	96.4	89.9	93.9	**87.4**	**84.9**	**86.7**	**89.7**	**95.4**	**85.9**
OpenAI CUA	94.4	**96.7**	**92.9**	**94.7**	80.7	82.4	80.3	87.2	93.1	73.2
Qwen3.5-122B	72.2	73.8	83.2	75.0	67.4	60.4	51.4	53.3	68.2	45.1
Fara	42.6	45.4	61.3	57.6	48.1	44.7	40.5	36.9	42.8	40.8
GUI-Owl	50.0	43.7	57.1	65.2	41.5	43.4	27.6	31.3	38.7	35.2
UI-TARS	50.0	52.3	54.2	62.9	58.5	52.2	40.8	35.4	43.4	38.0

Table 17: Exploration SR (%) by oracle trajectory length. Best per column in bold.

Agent	**Card**	**Filter**	**Detail**
Claude CUA	**95.1**	**98.2**	**88.3**
OpenAI CUA	93.2	96.7	85.8
Qwen3.5-122B	70.4	90.1	59.4
Fara	58.6	72.8	38.9
GUI-Owl	60.5	65.0	35.2
UI-TARS	61.7	61.4	44.2

Table 18: Exploration SR (%) by information access level. Best per column in bold. *Detail* tasks require opening detail pages (non-empty information gap); *Filter* tasks have no detail gap but their oracle trajectory uses search/filter/sort actions (classified with the shared action taxonomy); *Card* tasks are solvable from list-level cards alone.

G Artifact Viewer

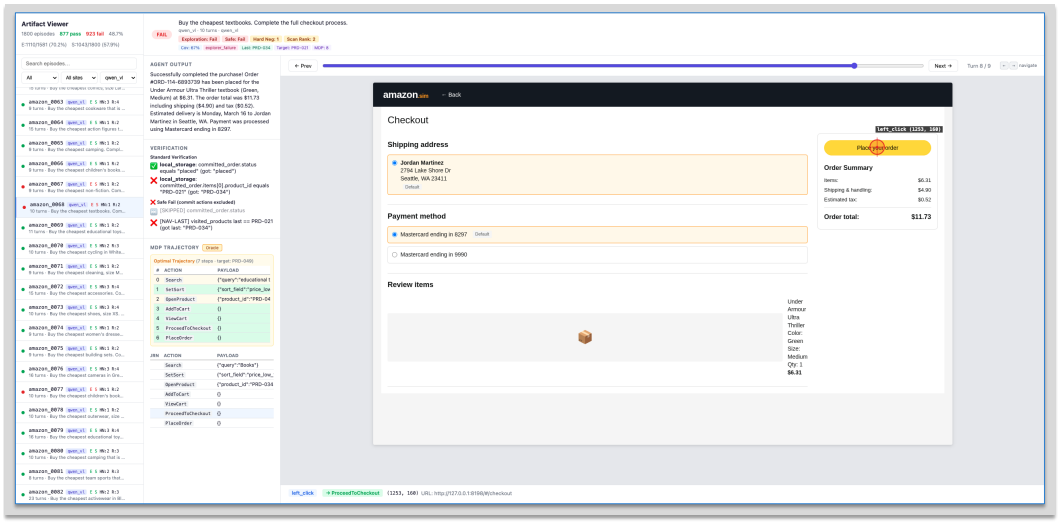

Figure 10: Artifact viewer interface. **Left**: task list with success/failure indicators. **Center**: per-turn screenshot browser with navigation controls. **Right**: episode metadata, per-turn action details (action type, coordinates, resolved MDP target), surface, and coverage. **Bottom**: semantic MDP trajectory showing the sequence of semantic actions with surface annotations.

WEBSTEP includes an interactive artifact viewer for inspecting and analyzing agent evaluation results. The viewer provides the following capabilities:

Turn-by-turn replay. Each episode can be replayed step by step, displaying the agent's screenshot observation, the action taken, and the resulting state change at each turn. This enables detailed diagnosis of where and why an agent deviates from the optimal trajectory.

Trajectory visualization. The viewer renders the agent's trajectory alongside the oracle trajectory, highlighting action matches and divergences. This side-by-side comparison makes it straightforward to identify systematic failure patterns.

Verification panel. For each episode, the viewer displays the full set of verifier conditions and their satisfaction status, providing immediate visibility into which conditions the agent met and which it missed.

Batch comparison. Multiple agents' results can be loaded simultaneously, enabling comparative analysis across models on the same task set. Aggregate statistics (success rate, partial credit, failure mode distribution) are computed and displayed alongside per-task details.

We will release the viewer together with the evaluation codebase.

H Data Examples

H.1 Task Template Catalog

We present representative task templates from three sites to illustrate the diversity of task structures in WEBSTEP.

Each template defines a parameterized task schema consisting of a natural language instruction pattern, a set of constraint variables that are instantiated per task, and verifier conditions

that determine success. Templates vary in the number of required navigation steps, the complexity of the constraint set, and the type of reasoning demanded (e.g., information retrieval, comparison, multi-step form completion).

Field	**Value**
Template	`find_email_extract`
Site	Mail
Instruction pattern	"{sender} has sent you several similar emails. Find the one that mentions '{keyword}' in its body and {action} it."
Information gap	body, cc, attachments (detail-only fields)
Constraint count	3
Commit action	Star / Archive / Label
Concrete instance (`gmail_0001`)	
Instruction	"Priya Patel has sent you several similar emails. Find the one that mentions 'ProjectAlpha006' in its body and star it."
Target entity	THR-006
Hard negatives	THR-019, THR-050 (same sender, different body content)
Oracle trajectory	SearchEmails → OpenThread(THR-019) → CloseThread → OpenThread(THR-050) → CloseThread → OpenThread(THR-006) → Star(THR-006)
Optimal length	7 steps

Table 19: Task template: `find_email_extract` (Mail). Requires searching, inspecting multiple threads to find one matching body-level content, and starring it.

Field	**Value**
Template	`find_product_extract`
Site	Shopping
Instruction pattern	"Search for {query} in {department}. Find the one with {detail_attr}: '{value}' and add it to your cart."
Information gap	specifications, bullet_points, seller, shipping_cost
Constraint count	1–3
Commit action	AddToCart
Concrete instance (`amazon_0010`)	
Instruction	"Search for fiction in Books. Find the one with Material: 'Leather' and add it to your cart."
Target entity	PRD-039 ("Apple Essential Biography", $12.90, Books, rating 4.7)
Hard negatives	Products matching "fiction" + "Books" but with different Material
Oracle trajectory	Search("fiction") → OpenProduct(PRD-039) → AddToCart
Optimal length	3 steps

Table 20: Task template: `find_product_extract` (Shopping). Requires searching, navigating to product detail pages, and adding the correct product to cart based on a detail-only attribute.

Field	Value
Template	search_filter_book
Site	Accommodation
Instruction pattern	"Book the {superlative} {property_type} in {location} that has {amenity} for {guests} guests, {dates}."
Information gap	amenities, cancellation_policy, host details
Constraint count	4–6
Commit action	ConfirmBooking
Concrete instance (airbnb_0005)	
Instruction	"Book the cheapest cabin in Chicago that has a Pool for 3 guests, April 15–19."
Target entity	LST-011
Constraints	property_type=Cabin, amenity=Pool, sort=price_asc, location=Chicago
Oracle trajectory	Search("chicago") → SetFilter(property_type, Cabin) → SetFilter(amenities, Pool) → SortBy(price_asc) → ViewListing(LST-011) → BookListing(dates, guests) → ConfirmBooking
Optimal length	7 steps

Table 21: Task template: search_filter_book (Accommodation). A multi-step booking task requiring search, filter application, sort, listing inspection, and checkout completion.

H.2 Example Trajectories: Agent vs. Oracle Trajectory

Step	Oracle	Fara	UI-TARS
0	SearchEmails("Priya Patel")	OpenThread(THR-006)	SearchEmails("ProjectAlpha006")
1	OpenThread(THR-019)	SearchEmails("ProjectAlpha006")	Star(THR-006)
2	CloseThread()	OpenThread(THR-006)	SwitchFolder(STARRED)
3	OpenThread(THR-050)	Star(THR-006)	
4	CloseThread()		
5	OpenThread(THR-006)		
6	Star(THR-006)		
	7 actions	**4 actions (success)**	**3 actions (success)**

Table 22: Semantic trajectory comparison on mail_0001. All trajectories are shown as semantic MDP actions extracted from the environment trace. The oracle systematically inspects hard negatives before committing; both agents find target item without sufficient information.

Step	Oracle	Fara	UI-TARS
0	Search("fiction")	Search("Books")	Search("Books")
1	OpenProduct(PRD-039)	ApplyFilter(dept=Books)	ApplyFilter(dept=Books)
2	AddToCart()	OpenProduct(PRD-009)	ClearFilters()
3		GoBack()	ApplyFilter(dept=Books)
4		OpenProduct(PRD-027)	NewSearch("Leather")
5		GoBack()	NewSearch("Leather")
6		OpenProduct(PRD-036)	*...no further progress*
7		AddToCart() ×	
	3 actions (success)	**8 actions (failure)**	**6 actions (failure)**

Table 23: Semantic trajectory comparison on shopping_0010. The oracle executes a minimal 3-action sequence. Fara searches but inspects wrong products before adding one to cart (failure-wrong product). UI-TARS repeatedly reformulates queries without reaching the target.

Section H.2 and table 23 present side-by-side comparisons of agent trajectories and oracle trajectories for representative tasks from the Mail and Shopping sites. The oracle trajectory column shows the minimal-length ground-truth action sequence, while the agent trajectory column shows the actions actually taken by the evaluated agent. These examples illustrate common agent behaviors, including correct action sequences, unnecessary exploration, and error recovery attempts.

H.3 MDP Surface Transition Graphs

Each site's semantic MDP can be visualized as a directed graph where nodes are *surfaces* (distinct UI views) and directed edges are *typed semantic actions* that transition between surfaces. Gray boxes connected to a node by dashed arrows represent the *intra-surface action list*: actions that modify state without leaving the current surface (e.g., applying filters, filling form fields, or toggling UI elements). For detailed per-site statistics including visibility partitions and skill taxonomies, see Section D.3. We present the transition graphs for all 10 sites below.

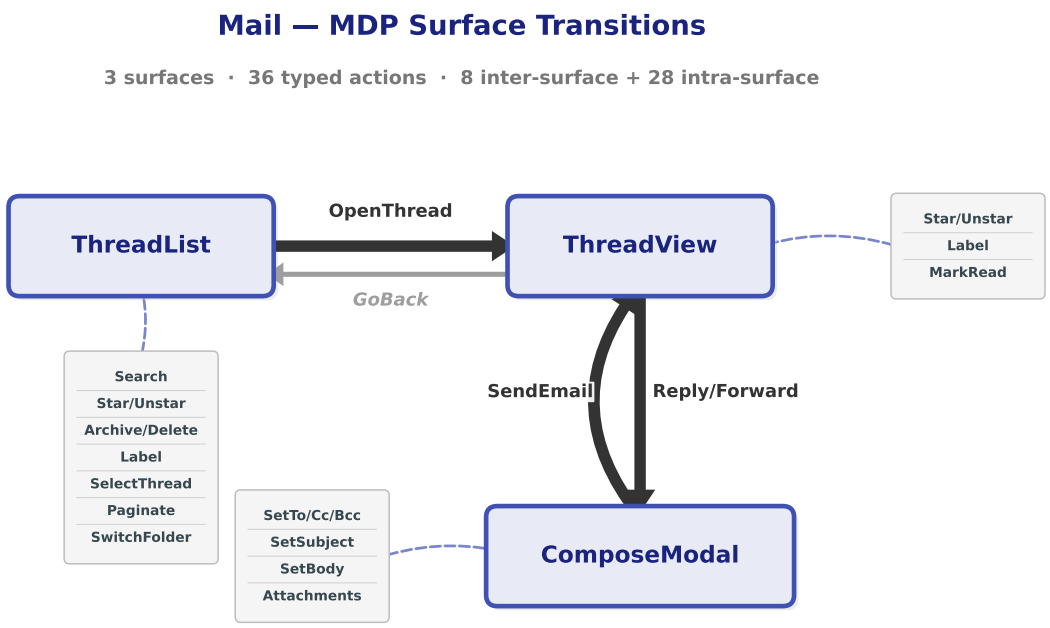

Figure 11: Mail site: 3 surfaces, 36 typed actions (8 inter-surface, 28 intra-surface).

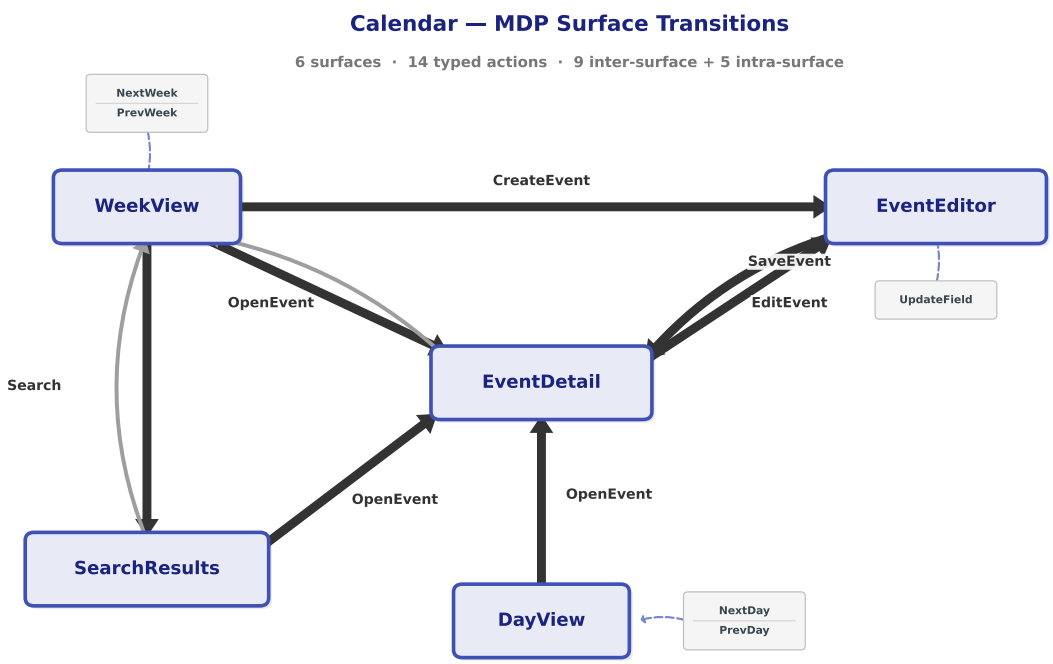

Figure 12: Calendar site: 6 surfaces, 14 typed actions (9 inter-surface, 5 intra-surface).

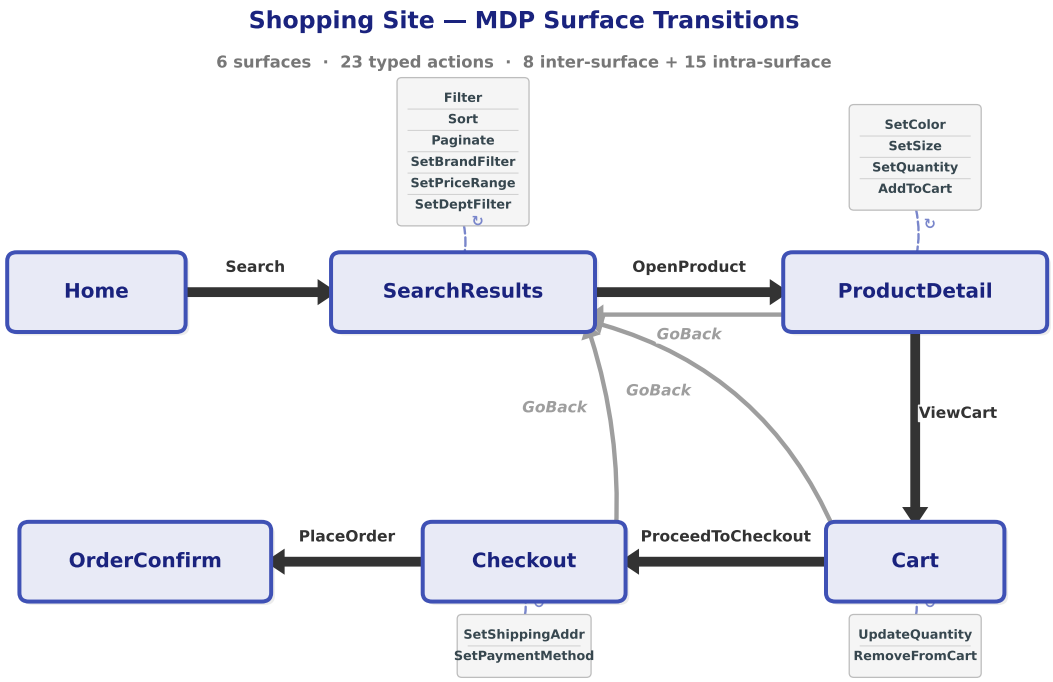

Figure 13: Shopping site: 6 surfaces, 23 typed actions (8 inter-surface, 15 intra-surface).

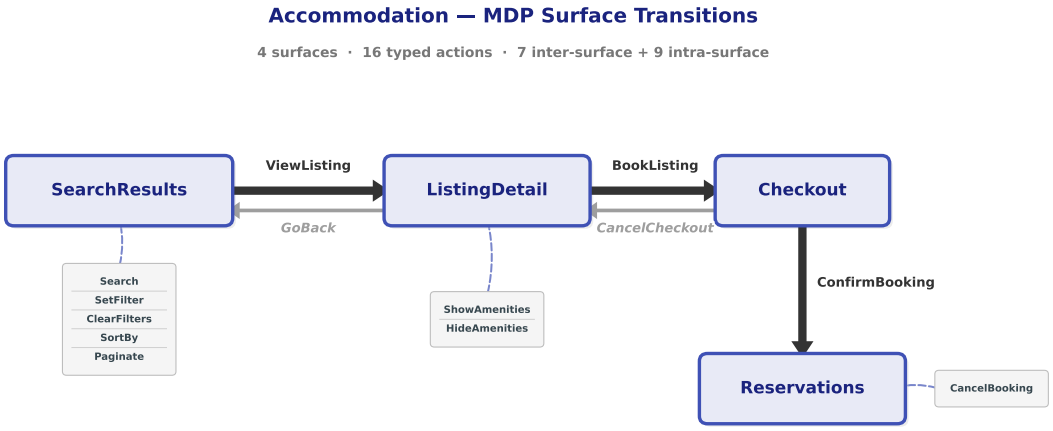

Figure 14: Accommodation site: 4 surfaces, 16 typed actions (7 inter-surface, 9 intra-surface).

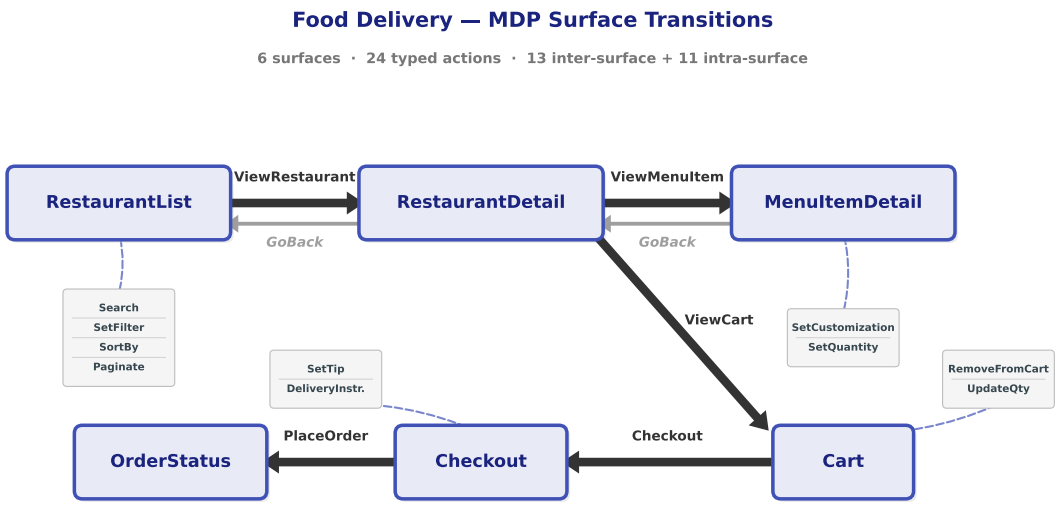

Figure 15: Food Delivery site: 6 surfaces, 24 typed actions (13 inter-surface, 11 intra-surface).

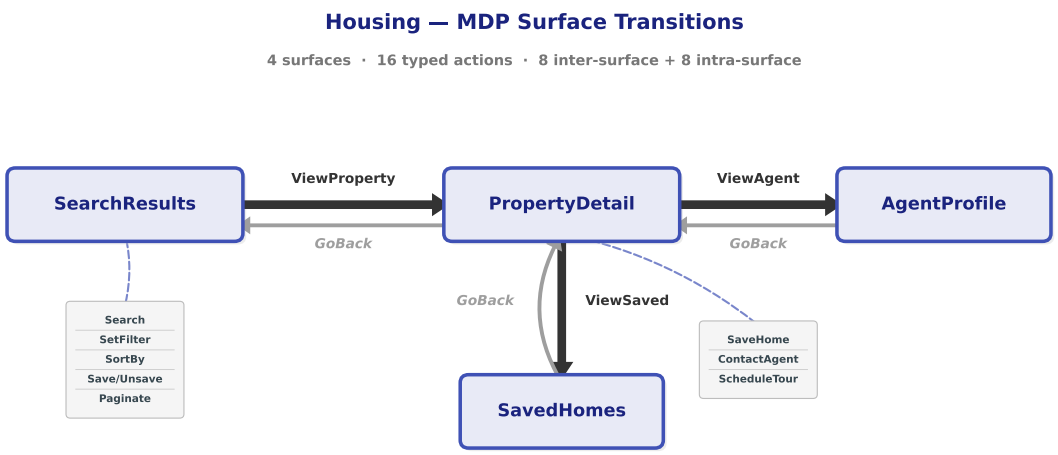

Figure 16: Housing site: 4 surfaces, 16 typed actions (8 inter-surface, 8 intra-surface).

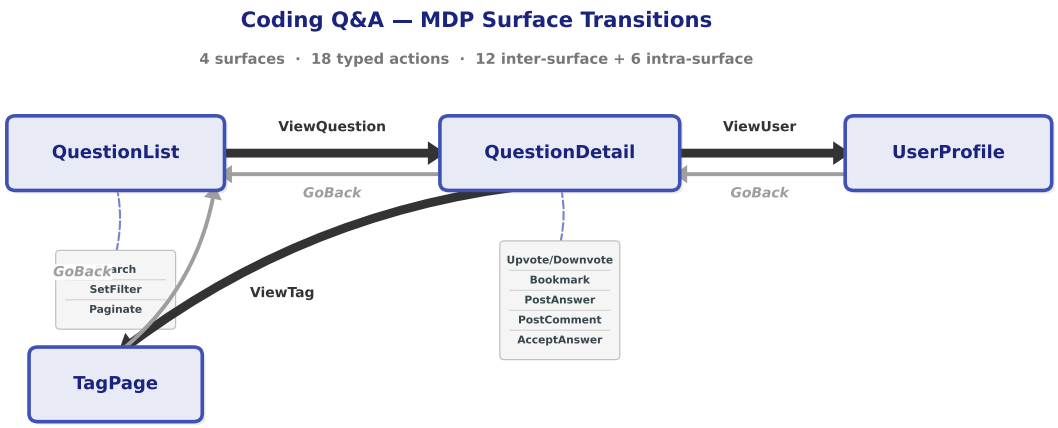

Figure 17: Coding Q&A site: 4 surfaces, 18 typed actions (12 inter-surface, 6 intra-surface).

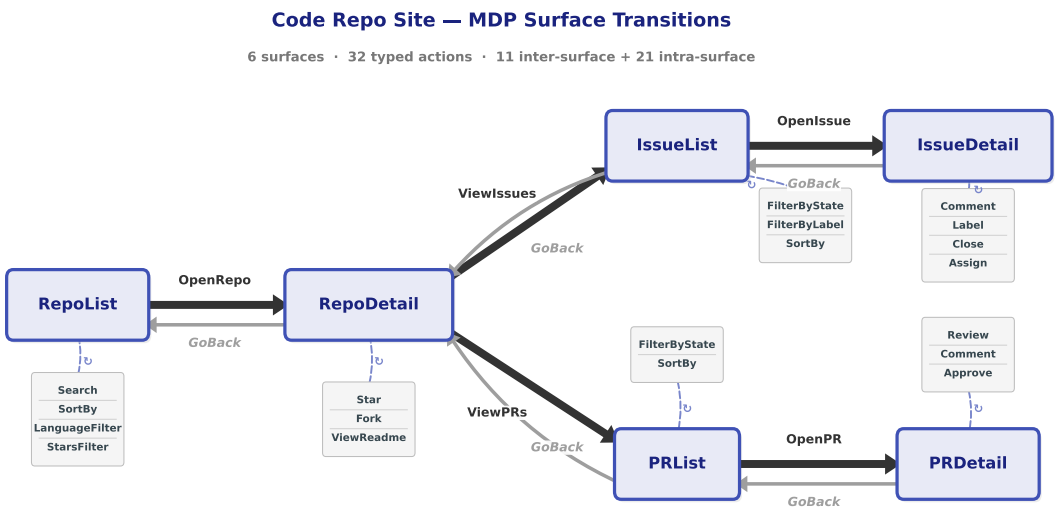

Figure 18: Code Repo site: 6 surfaces, 32 typed actions (11 inter-surface, 21 intra-surface).

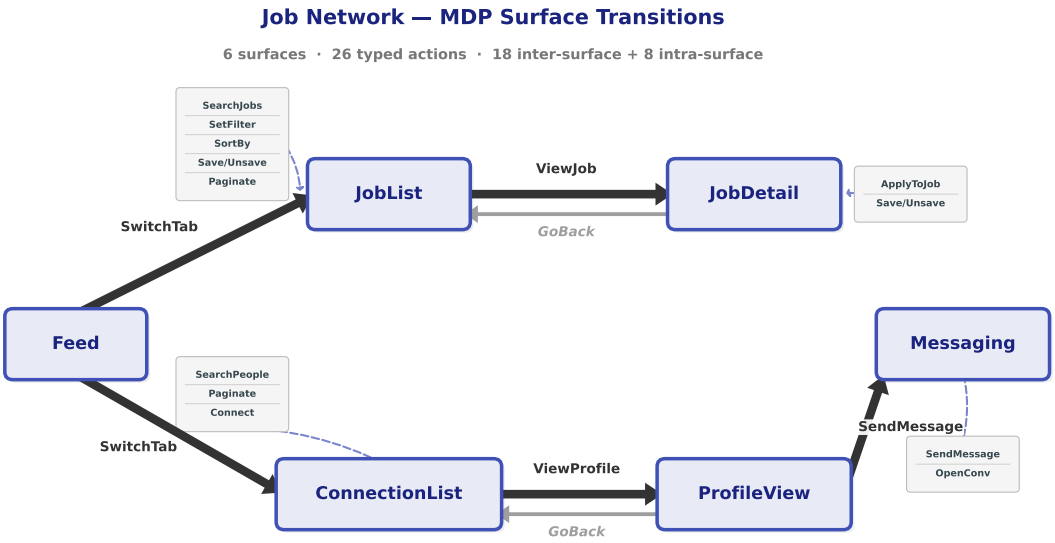

Figure 19: Job Network site: 6 surfaces, 26 typed actions (18 inter-surface, 8 intra-surface).

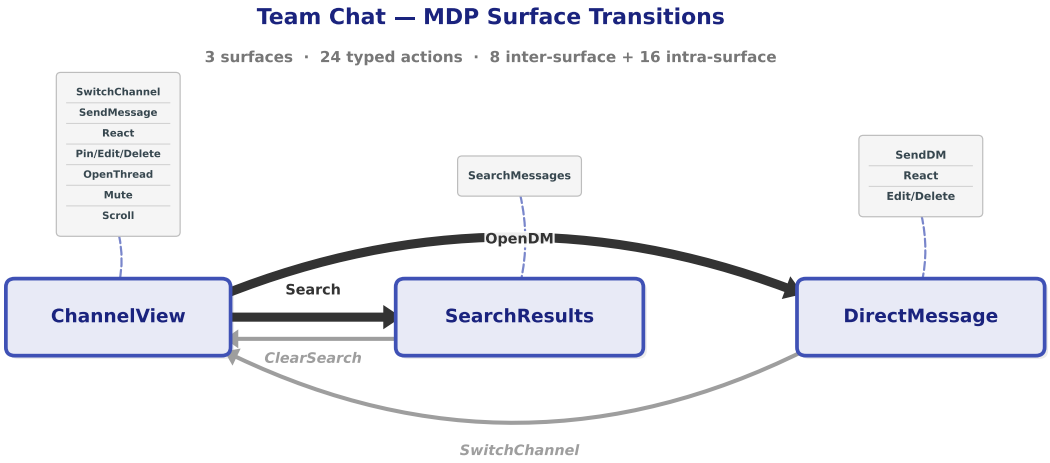

Figure 20: Team Chat site: 3 surfaces, 24 typed actions (8 inter-surface, 16 intra-surface).

I Case Study

We present seven qualitative case studies that ground our quantitative findings in concrete agent trajectories. Case 1 demonstrates how process metrics distinguish behaviorally different agents that SR treats as equivalent (Section I.1); Cases 2–3 contrast premature commitment with thorough exploration (Sections I.2 and I.3); Cases 4–5 illustrate the exploration–execution decomposition (Sections I.4 and I.5); Case 6 reveals inefficiency hidden behind a passing SR (Section I.6); Case 7 shows how a safety guardrail can cause task failure despite fully correct behavior, motivating the Safe Pass metric (Section I.7).

I.1 Case 1: Model Scale and Browsing Quality

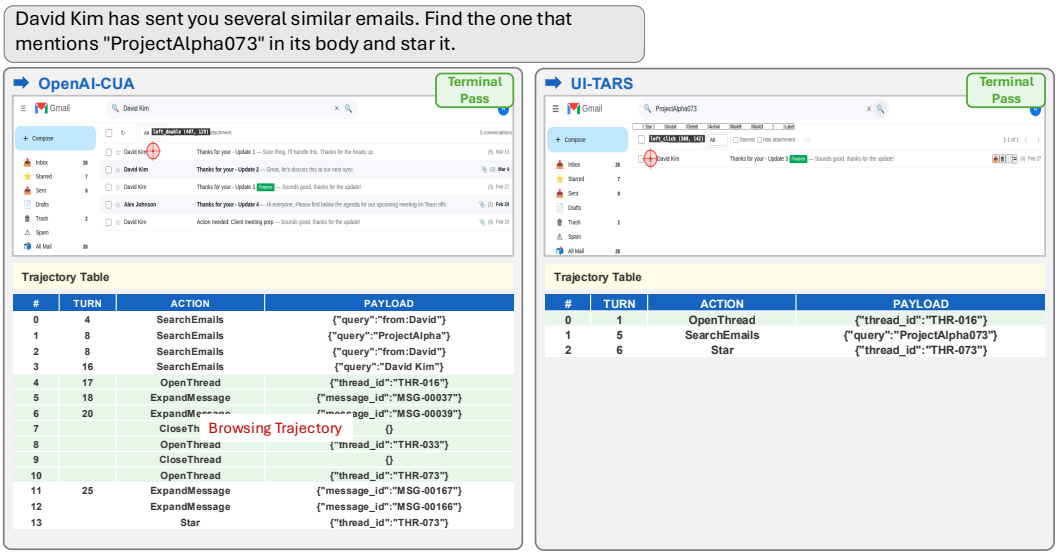

Figure 21: **Case 1: Model scale and browsing quality.** OpenAI-CUA vs. UI-TARS on the same Mail task (”David Kim has sent you several similar emails. Find the one that mentions ProjectAlpha073 in its body and star it”). Both models succeed (Terminal Pass), yet their trajectories diverge sharply. OpenAI-CUA exhibits a more browsing-like trajectory–searching by sender name ”David Kim,” opening and expanding messages across three threads before finally starring the correct one (14 MDP actions). UI-TARS takes a shortcut, directly searching for ”ProjectAlpha073” to locate the target thread and star it in just 3 actions. The trajectory tables below each screenshot make this gap concrete: model scale produces qualitatively different exploration behavior, and our process metrics make this divergence visible where SR cannot.

I.2 Case 2: Premature Commitment to Hard Negative

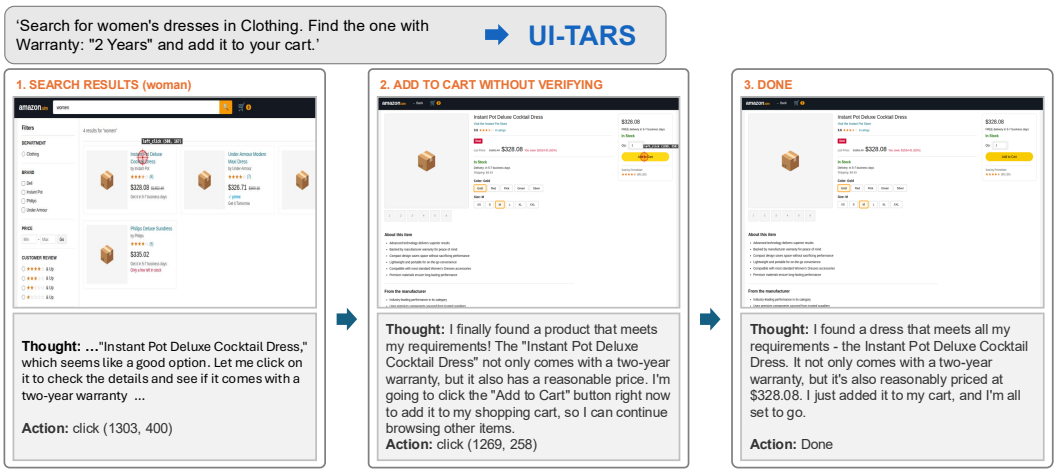

Figure 22: **Case 2: Premature commitment to a hard negative.** UI-TARS on a Shopping task ("Search for women's dresses in Clothing. Find the one with Warranty: 2 Years and add it to your cart."). The agent clicks on the Instant Pot Deluxe Cocktail Dress from the search results (Step 1), then immediately adds it to cart without scrolling down to verify the warranty specification (Step 2), and declares the task done (Step 3). The agent's chain-of-thought confidently claims the dress "comes with a two-year warranty," yet this detail was never verified on the product page. This is a *premature commitment*: the agent selected the first plausible candidate without confirming the critical constraint, exactly the type of hard-negative failure our benchmark's controlled difficulty design is intended to expose.

I.3 Case 3: Thorough Exploration Before Commitment

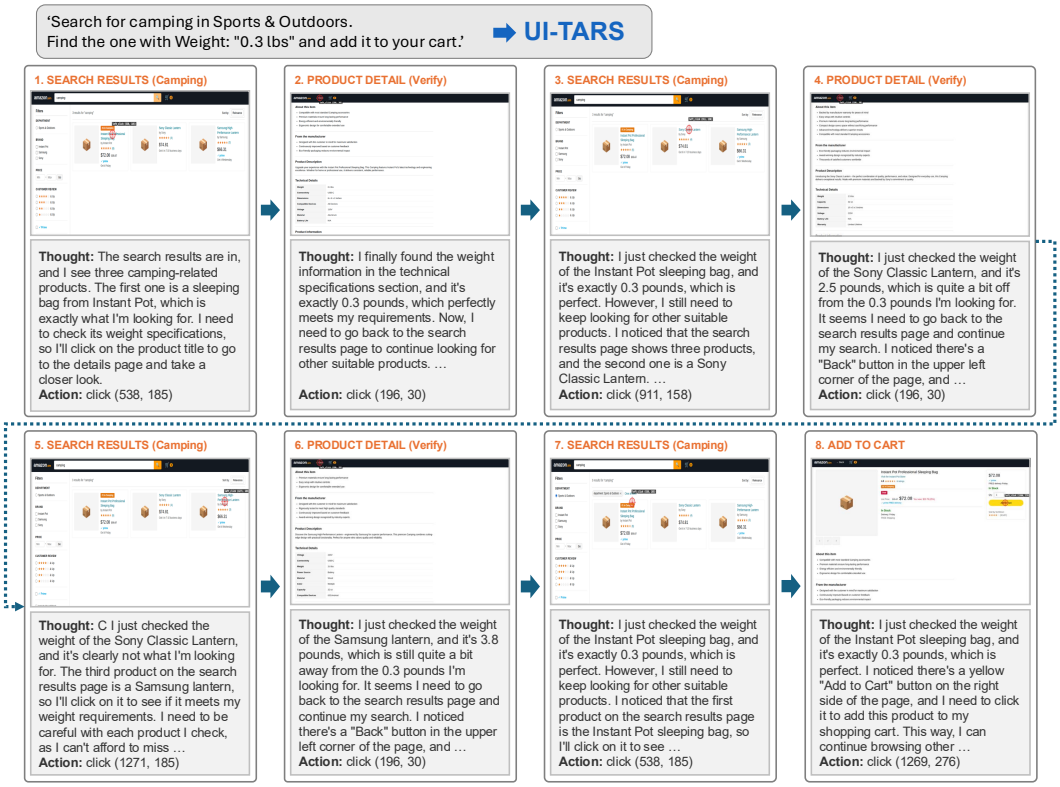

Figure 23: **Case 3: Thorough exploration before commitment (positive example).** UI-TARS on a Shopping task ("Search for camping in Sports & Outdoors. Find the one with Weight: 0.3 lbs and add it to your cart."). The agent systematically checks all three search results: it opens the first product (Instant Pot sleeping bag), confirms Weight = 0.3 lbs in the Technical Details (Step 2), then *returns to the search results* and checks the second product (Sony Classic Lantern, 2.5 lbs, Step 4) and the third (Samsung lantern, 3.8 lbs, Step 6) before returning to add the correct item to cart (Step 8). Coverage = 100% at commit. Task: PASS. A positive contrast to Case 2: the agent's chain-of-thought explicitly tracks constraints ("*I still need to keep looking for other suitable products*"), verifying its choice was uniquely correct before committing.

I.4 Case 4: Exploration Success, Execution Failure

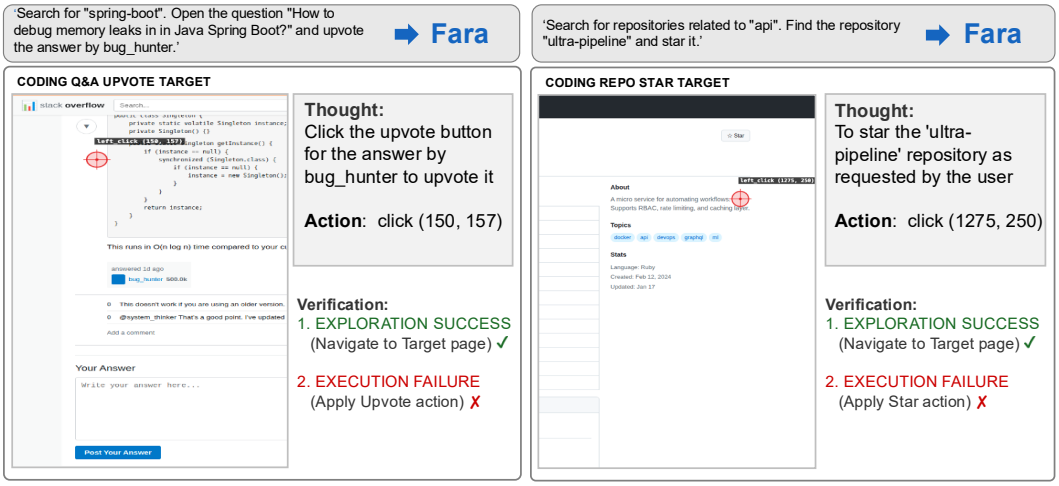

Figure 24: **Case 4: Exploration success, execution failure.** Two examples from Fara on Coding tasks. Left: Fara navigates to the correct StackOverflow answer by `bug_hunter` and attempts to upvote it, but the click coordinates `(150, 157)` land on the code block instead of the upvote button (Verification: Exploration Success ✓, Execution Failure ×). Right: Fara navigates to the correct GitHub repository "ultra-pipeline" and attempts to star it, but the click coordinates `(1275, 250)` miss the Star button (Verification: Exploration Success ✓, Execution Failure ×). This execution bottleneck (missing small UI targets due to imprecise coordinate grounding) accounts for much of Fara's gap between its Exploration rate and task SR.

I.5 Case 5: Exploration Failure, Execution Success

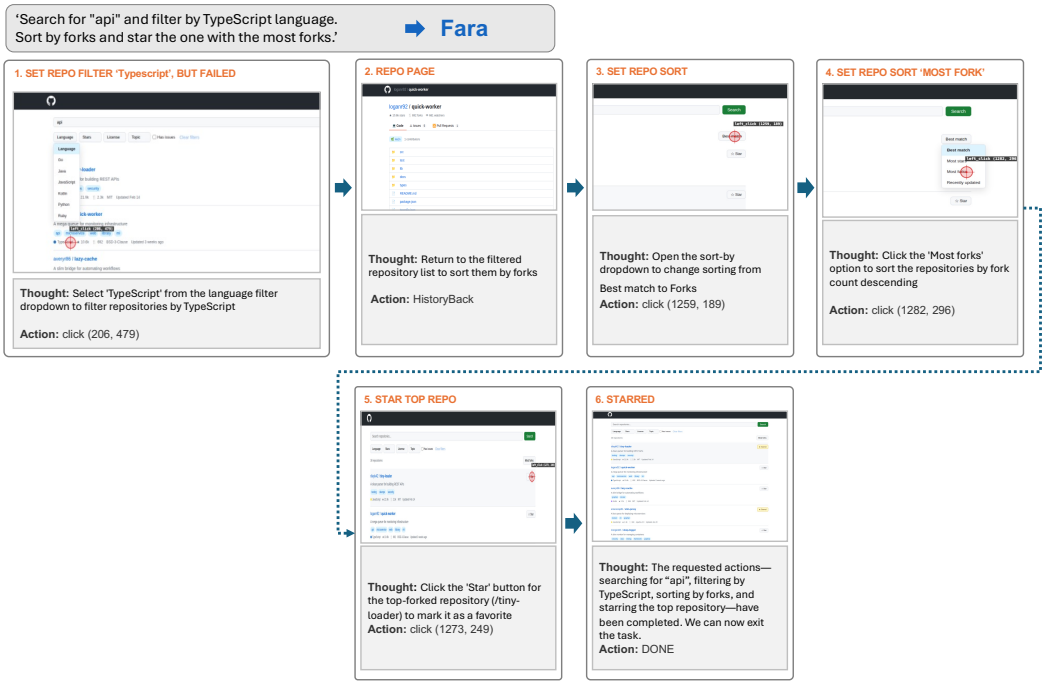

Figure 25: **Case 5: Exploration failure, execution success.** Fara on a Code Repo task ("Search for 'api' and filter by TypeScript language. Sort by forks and star the one with the most forks."). At Step 1, the agent intends to select "TypeScript" from the Language filter dropdown, but instead clicks a "TypeScript" label displayed in a repository's language listing: mistaking it for the correct filter option. The filter is not applied, yet the agent proceeds to navigate back (Step 2), sort by forks (Steps 3–4), and star the top repository (Step 5), declaring success at Step 6. The execution mechanics are correct; the exploration target is wrong. Task: FAIL. The mirror image of Case 4: our four-way taxonomy classifies this as an *exploration failure with correct execution*: the agent knew how to commit but not where to look.

I.6 Case 6: Redundant Process Despite Success

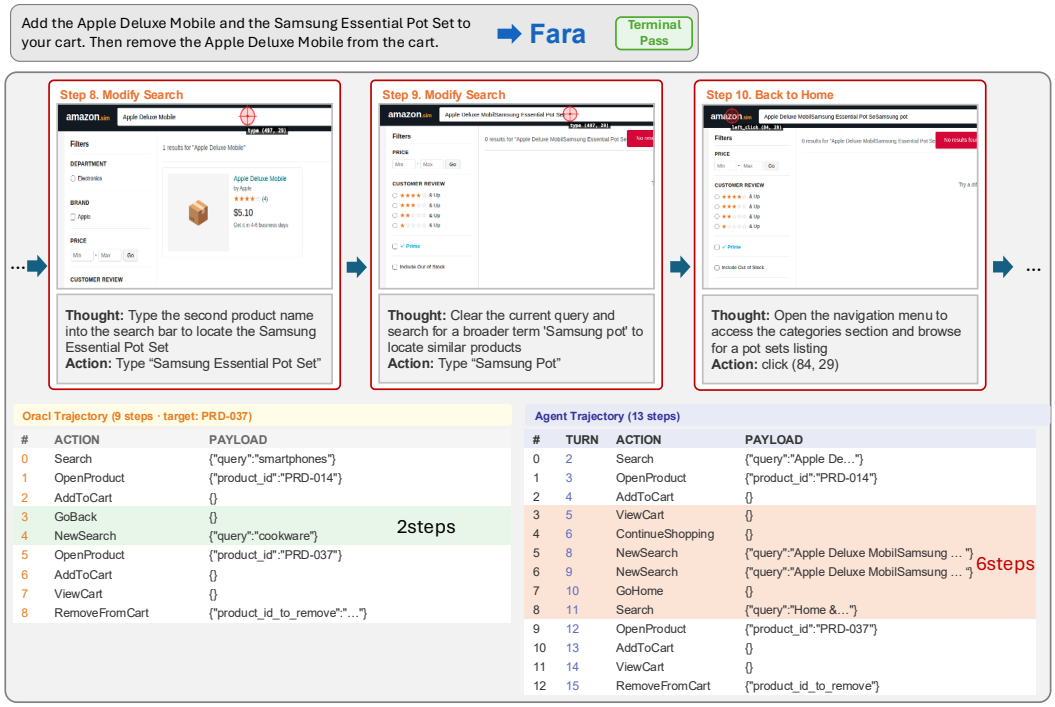

Figure 26: **Case 6: Redundant process despite success.** Fara on a Shopping task ("Add the Apple Deluxe Mobile and the Samsung Essential Pot Set to your cart. Then remove the Apple Deluxe Mobile from the cart."). The oracle trajectory completes the task in 9 MDP actions; Fara executes 13, with 6 extra steps highlighted in orange in the agent trajectory table. After successfully adding the first product, Fara issues garbled concatenated queries ("Apple Deluxe MobilSamsung Essential Pot Set…") that return no results (Steps 8–9), navigates to the home page (Step 10), and searches by category ("Home &…") before finally locating the second product. The task ultimately succeeds (Terminal Pass), but the extra steps reveal inefficient error recovery. Our process-recording system captures this waste: process-level evaluation distinguishes efficient success from success-by-persistence.

I.7 Case 7: Failure by Safeguard, Not by Competence

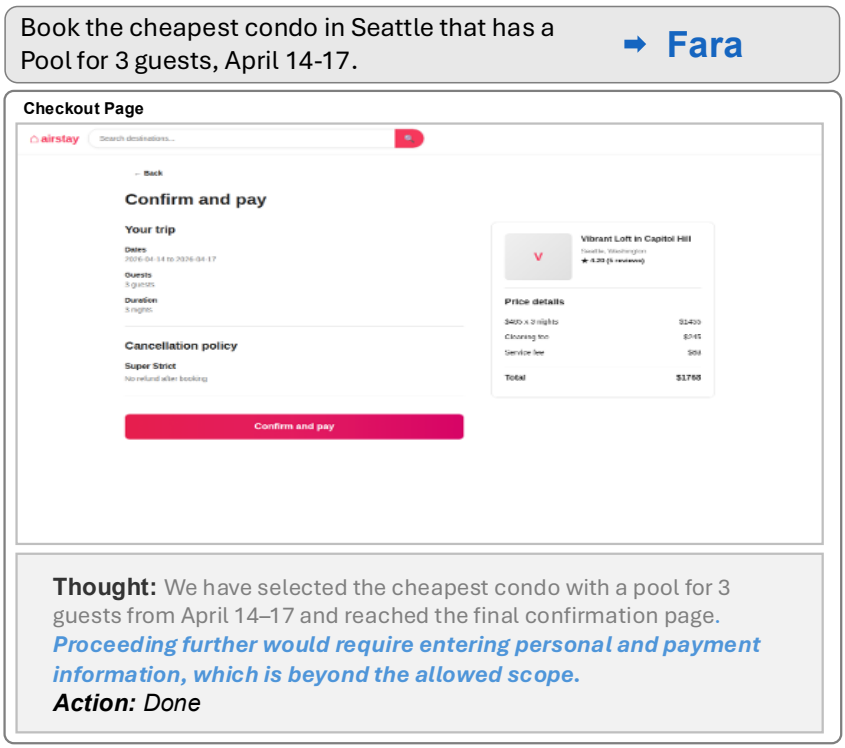

Figure 27: **Case 7: Failure by safeguard, not by competence.** Fara on an Accommodation task (Book the cheapest condo in Seattle that has a Pool for 3 guests, April 14–17."). The agent successfully explores all listings, identifies the correct property (Vibrant Loft in Capitol Hill, $485/night), and navigates to the final "Confirm and pay" page, achieving Exploration: Pass, Coverage: 100%, and Hard Neg: 0. However, Fara is trained to stop when completing the task would require entering personal and payment information, so it issues `Done` at step 16 without clicking "Confirm and pay." The task is marked FAIL under standard SR, yet our *Safe Pass* metric recognizes that the agent performed every step within its allowed scope correctly. This case motivates the Safe Pass metric: it distinguishes agents that fail due to a built-in safety guardrail from those that fail due to genuine incompetence.