HxAgent: Iterative LLM Agent Planning for Generating Replayable Web Test Scripts from Natural Language

Anonymous authors
Paper under double-blind review

Abstract

Turning natural-language functionality descriptions into executable test scripts is a labor-intensive step of web UI testing. We present HxAgent, an iterative LLM-based planning agent that executes a described task in a browser and emits a replayable, locator-based test script that runs without any LLM at replay time. After every action, a dedicated state evaluator reassesses progress and can stop a diverging trajectory, and the next action is chosen only among executable elements, conditioned on (1) the current observation, (2) a short-term memory of prior state–action pairs, and (3) examples and textual rules accumulated in context from HxAgent's own earlier successful and failed attempts; no model parameters are updated. HxAgent reaches 97.4% Exact-Match on MiniWoB++ (10.5% above a state-of-the-art WALT), 83.8% on 350 real-world tasks (13.4% above WALT), and 50.7% on OnlineMind2Web (13.5% above WALT). Counting only scripts that are generated and successfully replayed, it succeeds end-to-end on 78.9% of real-world and 38.2% of OnlineMind2Web tasks.

1 Introduction

In web UI testing, much of the process remains manual and error-prone. Testers execute textual task descriptions using tools such as Selenium (Selenium) or Katalon (Katalon), which record user interactions to form Sequences of Actions (SoAs)—automated test scripts. While effective, this manual recording is time-consuming and susceptible to human error, motivating automation.

Recent LLM-based approaches (Humphreys et al., 2022; Jia et al., 2018; Liu et al., 2018; Drouin et al.; Liu et al., 2023a; Sumers et al.) advance automated interaction but cannot directly generate executable test scripts. Existing UI agents face four main limitations: (1) Models such as WebArena (Zhou et al., 2023) and WebLINX (Lù et al., 2024) prioritize task completion over capturing executable SoAs, focusing on end-user automation rather than test generation. (2) Visual and multimodal planners (e.g., SeeClick (Cheng et al., 2024), UI-TARS (Qin et al., 2025), SeeAct (Zheng et al., 2024), CogAgent (Hong et al., 2024b)) represent actions as absolute screen coordinates or textual descriptions—fragile under UI changes and difficult to translate into executable scripts. (3) Pattern-based agents (e.g., MindAct (Deng et al., 2024), RCI (Kim et al., 2024), AdaPlanner (Sun et al., 2024), Synapse (Zheng et al., 2023)) depend heavily on human demonstrations, limiting scalability. (4) Li *et al.* (Li et al., 2023) avoid demonstrations via self-reflection, yet often fall into unguided trial-and-error loops lacking reasoning over prior state–action pairs.

We introduce HxAgent, an iterative LLM-based planning approach for automatically generating executable SoAs from task goals. Inspired by ReAct (Yao et al., 2023), HxAgent continuously reasons over (1) the current page, (2) feasible actions, (3) short-term memory of prior state–action pairs, and (4) long-term experience with rule abstractions extracted from past correct and incorrect trajectories. Each action is executed and its outcome informs the next step until completion. By anchoring reasoning in textual and visual evidence, HxAgent mitigates hallucinations and ensures decisions remain grounded in real screen context.

Unlike reactive reflection methods, HxAgent has *proactive correction*, which means that a state evaluator, separate from action selection, checks every intermediate state and stops diverging runs, while actions are restricted to executable elements and conditioned on rules from earlier failures.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ReAct (Yao et al., 2023) replans stepwise but has no separate evaluator, action constraint, or cross-episode memory; Reflexion (Shinn et al., 2024) corrects only after a trial fails.

Empirically, HxAgent outperforms baselines by 13.2%, 5.7%, and 11.4% in `Exact-Match` accuracy on OnlineMind2Web, MiniWoB++, and Real-world datasets, respectively, and achieves 77.3%–100.0% success in test script generation. On challenging tasks, it attains 81.6% Exact-Match, while others largely fail. In brief, this paper provides the following contributions:

1. HxAgent: an LLM-based approach that converts a natural-language task description into the sequence of actions that accomplishes it on a web application, and turns that sequence as executable, replayable scripts for automated testing.

2. An iterative LLM-based action-sequence generation with short- and long-term experience.

3. An empirical evaluation to show its effectiveness over the baselines and **a dataset** (HxAgent) containing tasks and actual SoAs, which can serve as a benchmark for future studies.

2 KEY IDEAS

We design HxAgent with the following key ideas:

Key Idea 1 [Iterative LLM-based Action-Sequence Generation]. We design an iterative LLM-based planner that generates the next action based on the current page, feasible actions, previous state–action pairs, and past (in)correct experiences. Each generated action is executed in the testing environment, and the resulting state guides the next generation step until the task is completed. Actions are explicitly represented as tuples of web-element operations with optional input values, forming an executable test script. Unlike self-reflection approaches that retry faulty plans, HxAgent performs *proactive correction*, reassessing and refining the action sequence after each step to avoid irrelevant or hallucinated actions.

Key Idea 2 [Sequence Action Memory as Short-Term Memory]. We introduce a short-term memory mechanism that stores previous actions and the resulting webpage states. HxAgent uses this structured history to maintain context, avoid previously failed decisions, and guide future reasoning.

Key Idea 3 [Combining action memory with experience reinforcement]. Building on LLM reflection mechanisms (Shinn et al., 2024; Li et al., 2023), we propose a two-phase strategy with a training and an evaluation phase. In training, the LLM gathers state–action sequences with success or failure feedback, which are aggregated to guide performance in evaluation. *Short-term memory* complements this process by retaining recent actions and outcomes, such as clicks and inputs, providing continuity, while accumulated experience enables the model to generalize from past patterns and apply effective strategies in new tasks.

Key Idea 4 [Vision enhancement]. The HTML of real-world websites can be extensive, risking exceeding the LLM's context length. Selecting elements is also challenging when complex layouts obscure an element's context: in a date picker, for instance, an agent may detect "Day 1" as interactable but fail to distinguish between months, yielding ambiguous options. To address this, we exploit the multi-modal capabilities of LLMs, providing the screenshot alongside the actionable elements of the current webpage to support the choice of next action.

3 HXAGENT

3.1 FORMULATION

We model web-based task execution as a Partially Observable Markov Decision Process (POMDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T})$ where $s_t \in \mathcal{S}$ denotes the latent web environment state at time t, $a_t \in \mathcal{A}$ is an executable action, $o_t \in \mathcal{O}$ is the observation, and $\mathcal{T}$ is the transition function. We omit the reward $\mathcal{R}$ and discount γ because HxAgent optimizes no return: actions are chosen by a frozen, prompted LLM, and task success is a binary judgment by the state evaluator G (Eq. 6), not a learning signal.

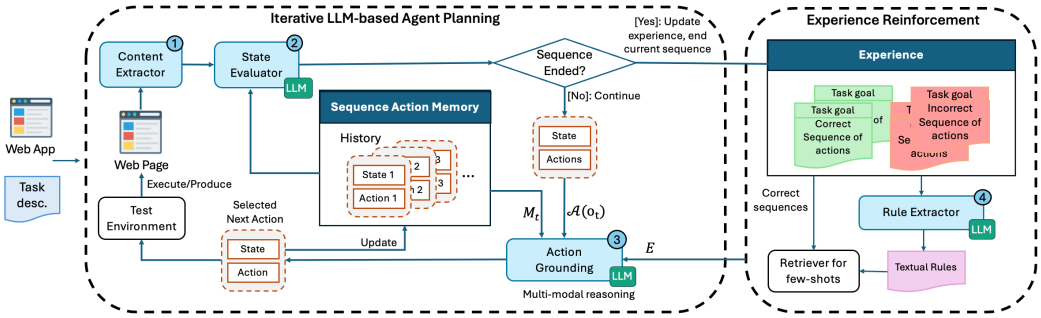


Figure 1: HxAgent Architecture

Multi-modal observation. Since the agent can only partially observe s_t, it receives a multi-modal observation $o_t = \left(o_t^{txt}, o_t^{img}, o_t^{elm}\right)$, where o_t^{txt} contains textual DOM information, o_t^{img} is a webpage screenshot, and o_t^{elm} denotes the set of interactable elements.

The feasible action set is defined as $\mathcal{A}(o_t) = \{(op, e, v) \mid e \in o_t^{elm},\ op \in \mathcal{O},\ v \in \mathcal{V} \cup \{\varnothing\}\}$ where op denotes an operation (e.g., click or input), e a web element, and v an optional input value.

Sequence of actions. An action sequence to perform a task up to time t is defined as

$$\tau_t = \left((s_0, a_0), (s_1, a_1), \ldots, (s_t, a_t)\right) \tag{1}$$

where the state transitions $s_{t+1} \sim \mathcal{T}(s_t, a_t)$ and a_t is determined using the grounding function (5).

Sequence action memory (SAM). This short-term memory at step t is defined as

$$M_t = \{(\hat{s}_{t-k}, a_{t-k}), \ldots, (\hat{s}_{t-1}, a_{t-1})\} \tag{2}$$

where $\hat{s}_t$ denotes the observed state representation and k is the memory window size. This memory maintains recent interaction context to guide subsequent decisions to accomplish the current task.

Long-term experience memory. This memory stores historical interaction trajectories from multiple tries for each task, their outcomes, and behavioral rules:

$$E = \{(\tau^{(i)}, y^{(i)})\}_{i=1}^{N} \cup \mathcal{K}^{(j)} \tag{3}$$

where $\tau^{(i)}$ denotes a complete SoA and $y^{(i)} \in \{0, 1\}$ indicates task success or failure. Abstracted behavioral rules can be derived as

$$\mathcal{K}^{(j+1)} = \phi(\tau^{(i)}, E) \tag{4}$$

where ϕ denotes a rule-based knowledge extraction function. This function is implemented by prompting the LLM with a completed trajectory and the current rule set, asking it to abstract a concise behavioral rule (e.g., "verify the label before clicking") only when the candidate rule is not already entailed by an existing one. Rules are stored as plain-text heuristics alongside their source SoA. Implementation details are given in Appendix C.

These memory components complement each other: the short-term memory maintains immediate context and the current sequence of actions required to accomplish the task, while the long-term experience captures learned strategies distilled from many SoAs explored during training.

Action grounding. Inspired by ReAct (Yao et al., 2023), which enables LLMs to interleave reasoning and acting by pausing after each decision to observe the environment, we adopt an iterative action grounding scheme that conditions decision making on the current observation o_t, short-term memory M_t, and accumulated experience E. At a time step t, the agent selects an action according to

$$a_t \sim \pi(. \mid o_t, \mathcal{A}(o_t), M_t, E) \tag{5}$$

where the grounding function π is implemented by an LLM and constrained to the feasible action set $\mathcal{A}(o_t)$. This formulation enables iterative, memory-aware, and experience-driven decision making to support long-horizon task execution in dynamic web environments.

State transition. After grounding an action, the agent executes the action on the test environment and transitions to the next state $s_{t+1} \sim \mathcal{T}(s_t, a_t)$.

Algorithm 1 Iterative LLM-based Agent Planning

[1] Input: $task$ to complete, url of the target webpage, and Experience of previous action sequences E
$M_t \leftarrow \varnothing$ IterativeAgentPlanning$task, url$ $s_t \leftarrow TestEnvironment.load(url)$ $t < MAX_t$ $\hat{s}_t, A(o_t) \leftarrow ContentExtractor(s_t)$
$success, stop \leftarrow \mathcal{G}_{\mathcal{LLM}}(\tau_t, M_t, E)$
$success = true$ training $E \leftarrow E \cup (M_t, 1)$ **break**
$stop = true$ //early termination training $E \leftarrow E \cup (M_t, 0)$ **break**
$a_t \leftarrow \pi_{\mathcal{LLM}}(\hat{s}_t, \mathcal{A}(o_t), M_t, E)$
count(a_t) > 1 // a_t is duplicated $a_t \leftarrow \pi_{\mathcal{LLM}}(o_t^{img}, a_t, M_t, E)$
a_t requires input $inputvalue \leftarrow LLM(a_t, M_t, E)$ $a_t \leftarrow a_t \cup inputvalue$
$M_t \leftarrow M_t \cup (\hat{s}_t, a_t)$ $s_t \leftarrow \mathcal{T}(s_t, a_t)$ $t \leftarrow t + 1$
$t = MAX_t$ training $E \leftarrow E \cup (M_t, 0)$
$E \leftarrow E \cup \phi_{\mathcal{LLM}}(\tau_t, E)$ // update rules

State evaluation. At each new state, the agent evaluates the state to determine whether the task has been successfully completed or whether execution should terminate early due to failure. The evaluation is defined as follows:

$$\{success, stop\} \sim \mathcal{G}(\tau_t, M_t, E) \tag{6}$$

If the task is completed, the outcome $y^{(i)}$ of the experience memory E is updated automatically when human judgment is not involved in determining task success.

3.2 Architecture

Fig. 1 shows HxAgent's architecture with LLM indicating LLM-based agents. It takes as input the web application and the task description, and produces the SoA to complete the task. HxAgent operates with two key components: 1) Iterative LLM-based Agent Planning to produce action sequences, and 2) Experience Reinforcement to support the decision-making.

Given a target web application and task description, HxAgent initiates its iterative process from the specified starting page URL. The process begins with Content Extractor (1), which operates on the current webpage state s_t to capture the content $\hat{s}_t$ and enumerate feasible actions $\mathcal{A}(o_t)$. State Evaluator (2) implements the evaluation function $\mathcal{G}$ by employing the LLM to evaluate the current state of the action sequence using the action sequence τ_t, sequence action memory M_t, and experience memory E. Within a single prompt, the agent (i) summarizes previous interactions, (ii) describes the current state, (iii) determines whether to stop execution early due to significant divergence from the expected state, (iv) decides whether the task is complete, and (v) evaluates whether the intended goal has been successfully achieved.

At each iteration t, Action Grounding (3) implements action grounding $\pi(.)$ to determine the most suitable next action a_t. In our implementation presented in Algorithm 1, the agent first uses the LLM with textual presentation of the current page $\hat{s}_t$, and if there exists two or more applicable actions, the agent uses the image or visual observation o_t^{img}.

The Rule Extractor (4) component implements the long-term experience memory E by extracting rules from previous action sequences.

3.3 Test Case Generation

HxAgent produces test scripts by converting an action sequence that successfully completes a task into a test case, where the task description serves as the test case name. Each step in the test case corresponds to an action executed on a given state. An action is defined by its type (e.g., click, text entry), value (e.g., input text), target web element, and the state of a webpage. The test cases can be automatically replayed in the target environment during regression testing.

4 Experimental Methodology

We seek to evaluate (1) the effectiveness of HxAgent in generating action sequences (Sequence Action Effectiveness), (2) its effectiveness in generating test cases (Test Generation Effectiveness),

(3) the impact of long-term experience memory, (4) the cost of LLM usage, and (5) the effect of each HxAgent component (Ablation Study).

Datasets. We use three datasets. First, Real-world dataset (Table 7), includes 350 tasks from seven popular websites (50 each) generated from task templates (e.g., "Subscribe to the channel"). Second, MiniWoB++ (Liu et al., 2018), provides 44 sampled tasks with 25 instances each (1,100 total). Third, OnlineMind2Web (Xue et al., 2025b), extends Mind2Web (Deng et al., 2024) with 300 tasks from 136 websites across multiple domains. After removing 15 inaccessible ones, 285 tasks from 132 sites were used.

Baselines. We restrict our comparison to agents that can emit *executable*, *replayable* action sequences, since this is a prerequisite for test-script generation rather than one-time task execution. Therefore, we chose the following three primary baselines: Li *et al.*, SeeAct, and WALT (Prabhu et al., 2026). For SeeAct, we adopt $SeeAct_{choice}$, which grounds actions through textual choices. The details on the rationales for selecting these benchmarks and baselines are in the Related Work section.

In addition, Table 8 includes several other baselines, namely WebNT5, HTML-T5-XL, WebGUM, WGE, CC-Net, RCI, AdaPlanner, and Synapse. These methods are evaluated only on MiniWoB++, as they depend on human demonstrations that are unavailable for the other two datasets.

Each approach uses its own action-space extraction mechanism. All approaches run on the same applications and environment, so the set of actions available on each page is identical, but the candidate sets used by each approach differ. We treat extraction as part of each method's design, analogous to the grounding component of other agents. To isolate the contribution of planning and memory from that of extraction, the ablation in Section 5.4 keeps HxAgent's extractor fixed across all variants.

Procedure. We set up Selenium test environment via ChromeDriver. The task in the text string is used as input to HxAgent and baselines. We use *GPT–4o* in our experiments, as it serves as the primary LLM in most baselines and is widely regarded as a state-of-the-art model.

For MiniWoB++, we first run the training phase for 20 task instances, to obtain the maximum of 8 few-shot examples and a set of textual rules to be used as experience. As in Li *et al.*, we evaluated each task on 50 instances. For the real-world dataset, we evaluate on 50 instances for each task.

Evaluation Metrics. We define an action as correct if both the selected element (its content) and the predicted operation are correct. For sequence action generation, we use:

Exact Match Rate (`Exact-Match`*)*: We consider the generated SoA for a task as correct only if all actions are correct. `Exact-Match` is defined as the ratio of correct results to the total number of results.

Prefix Match Rate (`Prefix-Match`*)*: `Prefix-Accuracy` is the accuracy of the prefix of a generated SoA for a task instance, which is the ratio of continuous correct actions from the start to the first error, divided by the total number of actions. `Prefix-Match` is the average in all instances.

5 Empirical Results

5.1 Sequence Action Effectiveness

Performance on the Real-world dataset: As seen in Table 1, HxAgent achieves the highest performance on the Real-world dataset, with 83.8% Exact-Match and 91.8% Prefix-Match, outperforming Li *et al.* by 56.8%, WALT by 13.4%, and SeeAct by 11.4% on Exact-Match. Owing to its iterative planning mechanism (ReAct) and integration of short-term memory, HxAgent effectively adapts to interface variations and dynamic webpage states. In contrast, Li *et al.* employ a static, screen-based planning strategy with full HTML representations, which frequently exceed context limitations and fail to generalize when interfaces change. SeeAct, while demonstrating strong performance on structured platforms such as YouTube, LinkedIn, and Amazon, exhibits a substantial decline on highly dynamic websites like Expedia. Overall, HxAgent demonstrates superior adaptability and generalization capabilities, maintaining high accuracy even in complex, multi-screen tasks that require sequential reasoning and action execution.

Performance on OnlineMind2Web: Table 2 summarizes results on Online-Mind2Web dataset. HxAgent achieves 50.7% Exact-Match and 62.4% Prefix-Match over 285 tasks from 132 websites,

Table 1: Performance of Sequence Action Generation on Real-world dataset

Website	Exact-Match (%)				Prefix-Match (%)			
	Ours	**Li *et al.***	**SeeAct**	**WALT**	**Ours**	**Li *et al.***	**SeeAct**	**WALT**
YouTube	100.0	20.0	100.0	100	100.0	20.0	100.0	100
Linkedin	92.0	46.0	100.0	80	98.4	71.0	100.0	90
Facebook	93.3	33.3	66.7	66.7	96.7	39.0	79.2	75
Google	96.0	0.0	100.0	60	98.6	0.0	100.0	78
Amazon	93.4	54.0	100.0	80	94.4	56.0	100.0	96
Stackover.	80.0	36.0	40.0	80	90.0	45.4	63.3	80
Expedia	32.0	0.0	0.0	20	64.4	9.4	44.1	32.7
Tot./Avg	**83.8**	27.0	72.4	69.5	**91.8**	34.4	83.8	78.8

Note: **Ours** denotes our proposed framework, HxAgent.

Table 2: Performance of Sequence Action Generation on OnlineMind2Web

Level	No.	Exact-Match (%)				Prefix-Match (%)			
		Ours	Li *et al.*	SeeAct	WALT	Ours	Li *et al.*	SeeAct	WALT
Easy	80	81.3	40.0	63.8	56.3	90.8	54.0	75.8	76.6
Medium	135	40.7	14.8	30.4	36.3	55.0	26.2	42.4	68
Hard	70	30.0	5.7	10.0	20	41.2	13.8	20.2	62.6
Tot./Avg	285	**50.7**	20.2	34.7	37.5	**62.4**	31.3	46.1	69

consistently outperforming Li *et al.* and SeeAct across all difficulty levels. Its accuracy reaches 81.3% on easy, 40.7% on medium, and 30.0% on hard tasks.

Compared to the Real-world dataset, Online-Mind2Web includes more complex UI elements such as dropdowns and calendar pickers, where HxAgent's iterative planning proves effective. It outperforms Li *et al.* by 30.5%, SeeAct by approximately 16%, and WALT by 4.6%, particularly on tasks requiring multi-step reasoning and adaptation to dynamic state changes.

Despite these gains, HxAgent still struggles with large interactive elements such as datetime pickers, enter-to-submit fields, and large interactive components that may exceed LLM context limits.

Performance on MiniWoB++: As seen in Table 3, over 975 task instances from 39 tasks, HxAgent achieves 97.4% Exact-Match and 99% Prefix-Match: 97–100% on tasks with fewer than three screens or actions, and 93% on more complex ones. Without human demonstrations, it surpasses WebGUM (401K demos) and CC-Net (BC+RL, 2.4M demos), and matches Synapse (97.3%), which relies on 154 handcrafted exemplars and task-specific prompts that risk overfitting. Among demonstration-free methods, HxAgent outperforms Li *et al.* (97.4% vs. 91.7%), SeeAct by 27.5 points, and WALT by 10.5 points. On 125 challenging instances averaging five screens and seven actions (Table 4), HxAgent reaches 81.6% Exact-Match, versus 9.6% for Li *et al.*, 8% for SeeAct (successful only on `flight.AA`), and 0% for WALT. SeeAct's failures stem from datetime pickers, dropdowns, and icon-only buttons, which HxAgent handles robustly.

Accuracy by Task Complexity: We evaluate HxAgent by task complexity, measured through action sequence length and actions per screen.

Next-step prediction accuracy by SoA's length. As seen in Fig. 2a, HxAgent achieves 100% first-step accuracy on MiniWoB++ and Real-world datasets, with a gradual decline as sequence length increases. A similar trend occurs in Online-Mind2Web, where `Exact-Match` drops from 81.3% (easy) to 40.7% (medium) and 30.0% (hard).

Exact-Match sequence accuracy by number of actions per screen. In Fig. 2b, HxAgent maintains high `Exact-Match` even with up to 500 actions per screen. Performance decreases on long or dynamic tasks (e.g., pop-ups, date pickers, complex forms), with some outliers below 70%, mainly due to hidden options or invalid data generation.

5.2 Test Generation Effectiveness

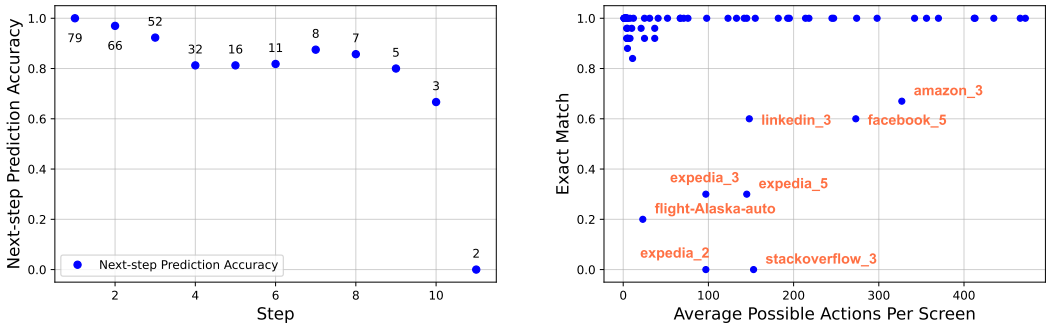
We used HxAgent to generate regression test scripts for the 1,493 tasks with correct SoAs and executed them on the target web applications. A script is considered valid if it (1) executes all steps without errors and (2) successfully reaches the intended goal. As shown in Table 5, all successful SoAs in MiniWoB++ produced valid scripts, while 90.8% of successful SoAs from real-world applications

Table 3: Action Sequence Generation on MiniWoB++. Exact-match (%), Correct runs (975 total).

Metrics	Li (4o)	SeeAct	Ours	WALT
Exact-Match (%)	91.7%	69.9%	97.4%	86.9%

Table 4: Performance on 125 Challenging Task Instances (MiniWoB++).

Tasks	Exact-Match (%)			
	Ours	Li *et al.*	SeeAct	WALT
choose-date	92.0	24.0	0.0	0.0
book-flight	100.0	24.0	0.0	0.0
flight.AA	100.0	0.0	40.0	0.0
flight.Alaska	96.0	0.0	0.0	0.0
flight.Alaska-auto	20.0	0.0	0.0	0.0
Total/Avg.	81.6 (±34.6)	9.6 (±13.1)	8.0 (±18)	0.0



(a) Next-step Prediction.

(b) Accuracy by Actions/Screens.

Figure 2: Performance by Task Complexity. Numbers at points in (a) give the tasks reaching that step.

executed correctly. The higher reliability in MiniWoB++ stems from its static, controlled environment, whereas real-world sites introduce dynamic content and runtime variability—for example, layout changes or fluctuating search results may invalidate XPath locators. On LinkedIn, the "Jobs" button may yield different XPaths across layouts, and on Expedia, network delays occasionally caused actions to execute before elements fully loaded, resulting in failures.

On the OnlineMind2Web dataset, our approach achieves a 77.3% success rate. Although many tasks resemble those in the Real-world dataset, the broader variety of website types introduces failures on pages with dynamic banners, dropdowns, or frequently changing navigation paths. For example, when searching "SAM2" on GitHub, the generated script may succeed during execution but later return a *429 Too Many Requests* error due to dynamic request limits.

Over all tasks, including those without a correct action sequence, HxAgent produces a script that replays successfully for 78.9% of Real-world and 38.2% of OnlineMind2Web tasks.

5.3 Experience Analysis

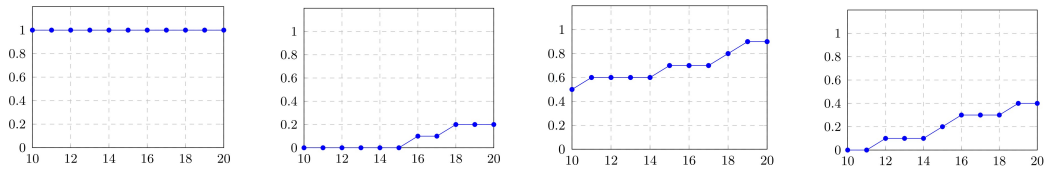
Fig. 3 shows the moving average for Experience, i.e., the average correct SoA over a window of 10 recent runs during training. Comparing Figures 3c and 3d (without experience rules), the moving average starts at 0.5 (5 correct sequences over 10 runs) with rules but at 0 without. Consequently, after 20 runs, the agent *without experience rules* reaches only 0.4, versus 0.9 (*9 correct sequences over the last 10 runs*) when rules are included. This rising trend indicates that HxAgent keeps acquiring correct knowledge during training. For `click-button`, with the best possible initial performance (Fig. 3a), retention stays consistent at a 100% moving average, whereas for a complex task (`Alaska-auto`) it is lower, as the correct sequence is harder to obtain. Tracking the moving average monitors the quality of knowledge in experience memory and helps select the best few-shot examples and textual rules for prediction: we can halt training once it exceeds a threshold (Fig. 3c) and capture the knowledge at that point in the Experience prompts for subsequent predictions.

Table 5: Success Rates of Generated Test Scripts

Test scripts for	Valid / Succ. Trial	End-to-end (Valid / #tasks)
MiniWoB++	1048/1048 = 100.0%	1048/1100 = 95.3%
Real-world App	276/304 = 90.8%	276/350 = 78.9%
OnlineMind2Web	109/141 = 77.3%	109/285 = 38.2%

Table 6: Contribution of Components

Module	EM (%)	PM (%)
HxAgent (4o)	**97**	**99**
w/o experience	88	93
w/o (SAM + experience)	47	67
w/o iterative planning	75	85



(a) click-button (b) Alaska-auto (c) search-engine (d) search w/o rules

Figure 3: Moving average of selected tasks in training

5.4 Ablation Study

Ablation Study on Key Components. We removed HxAgent's key components and compared their performance. For each component, we removed the corresponding section in the main prompt. However, for the variant *w/o iterative planning*, we changed the main prompt to ask LLM only once to generate a complete SoA without the loop for planning. As seen in Table 6, all variants have lower accuracies than HxAgent, indicating the contributions of each component. Without sequence actions memory, the variant `w/o (SAM + experience)` has the most drop in `Exact-Match` (47 vs. 97). Thus, SAM is most important with its retaining the history of previous steps and evolving states to decide the next action. We found that `w/o (SAM + experience)` kept repeating some action multiple times until the step limit. HxAgent *w/o iterative planning* has a decrease of 23% in `Exact-Match` where it struggles on tasks with more screens. The variant *w/o experience* has the smallest drop of 9% in `Exact-Match`. This confirms our idea that experience with reflection in Li *et al.* has less impact than the combination of `SAM+experience` and iterative planning.

Ablation Study on Varying Sizes of Short-term Memory. Our results show that restricting memory to *only 1–3 most recent actions* reduces `Exact-Match` by 6% and 3%, respectively. Full access to short-term memory proves beneficial for tasks where the current state does not fully reflect the completion of prior steps. For example, in the `click-tab-2` task, full memory access enables HxAgent to track all visited tabs, preventing redundant clicks on the same tab, an issue with limited memory.

5.5 Cost Analysis

Figure 10 shows the token distributions in prompts and outputs for each HxAgent component: *content tokens* (Content Extractor), *agent tokens* (Action Grounding), and *rule tokens* (Rule Extractor). These components are invoked repeatedly during the iterative loop, and prompt token counts vary with the complexity of the web application and the action-sequence length. Action Grounding consumes the most tokens, followed by Content Extractor, as both manage the web state and are critical to the iterative process. Figure 11 presents token consumption for input generation in Action Grounding, executed once per input field in the task. Overall, the average cost across all tasks is 80k input and 22k output tokens for both training and evaluation.

Compared with WALT, HxAgent consumes about one-third of WALT's total tokens; for output tokens, the gap is larger still, at only 8.9% of WALT's count (see Section G.5).

6 Related Work

Benchmarks for Web Task Completion and UI Agents. The rapid progress of web UI agents has been supported by realistic benchmarks for navigation and interaction, such as WebArena (Zhou et al., 2023), WebLINX (Lù et al., 2024), WebGames (Thomas et al., 2025), OSUniverse (Davydova et al., 2025), BEARCUBS (Song et al., 2025a), and EconWebArena (Liu & Quan, 2025) BrowseComp (Wei et al., 2025). Since these benchmarks are defined in natural language rather than executable form, we collected our own real-world dataset (Table 7) and adopted MiniWoB++ (Liu et al., 2018) and

OnlineMind2Web (Xue et al., 2025a), which better align with test generation using available or **executable** sequences of actions.

Web Task Completion and Web UI Agents. Several approaches have been proposed to build task completion and UI agents including Browsing (Song et al., 2025b), ShowUI (Lin et al., 2025), OpenAI's Operator (Anonymous, 2025), SeeClick (Cheng et al., 2024), CogAgent (Hong et al., 2024b), InfiGUIAgent (Liu et al., 2025), UI-TARS (Qin et al., 2025), UI-AGILE (Lian et al., 2026), UI-Venus (Gu et al., 2025).

Computer-Using Agents. Beyond the web, agents have been developed for mobile (Guardian (Ran et al., 2024), GPTDroid (Liu et al., 2023b), AutoDroid (Wen et al., 2023), DroidAgent (Yoon et al., 2023)) and desktop use (TaskMatrix.AI (Liang et al., 2024), Computer Use (Gur et al., 2023b), Claude 3.5 Sonnet (Anthropic, 2025)).

However, most of these agents are designed to perform computer-using tasks with LLMs that represent actions as absolute screen coordinates or natural-language descriptions. Such representations are brittle for regression testing and, as designed, may not be executed properly without invoking an LLM at replay time. In contrast, our approach generates scripts that run regression tests without any subsequent LLM involvement. We therefore exclude these agents from our comparison: their outputs (textual descriptions or absolute coordinates) would first need to be converted into executable actions for test generation, a process that could introduce errors and lead to unfair comparisons. Instead, we adopt SeeAct (Zheng et al., 2024) and WALT (Prabhu et al., 2026) as our two baselines, since *both can prompt LLMs to output executable SoAs* and are recent state-of-the-art approaches.

Relation to reflection-based agents. HxAgent shares ingredients with ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2024) but differs in its control structure. ReAct interleaves reasoning and acting without persistent cross-episode memory and Reflexion repairs reactively, revising only after a full trajectory fails. HxAgent instead (i) separates **state evaluation** from **action grounding** via a dedicated evaluator $\mathcal{G}_{LLM}$ that decides completion or early termination after every step, enabling **proactive** correction rather than end-of-trajectory repair; (ii) extracts structured behavioral rules incrementally from both successful and failed trajectories during training; and (iii) conditions each subsequent action on this accumulated, abstracted rule set rather than on raw trajectory replay alone. The result is a self-improving control loop with persistent experience abstraction, and the ablation in Section 5.4 quantifies the payoff: removing experience drops Exact-Match by 9 points, and removing short-term memory with it drops by 50 points, confirming that the combination improves performance.

7 Conclusion

We present HxAgent, a multi-modal, iterative Web testing agent that uses vision-capable LLMs to generate test scripts for GUI tasks described in natural language. We showed that long-term experience memory, with rules extracted from past experience, helps the agent make better decisions, and that grounding the model's reasoning in the actual text and visual context at each step reduces hallucinations. Moreover, HxAgent's proactive correction reassesses the state after each step to adjust its plan, in contrast to state-of-the-art self-reflection approaches, whose reactive repair attempts corrections only after an entire trajectory has failed. Our evaluation on three datasets shows superior performance over the baselines.

This work suggests promising directions for test generation and autonomous end-to-end testing. As LLMs and multimodal reasoning advance, future frameworks may not only generate reliable action sequences but also autonomously perform tests, reducing human effort. Research may move toward fully autonomous, self-reflective agents that continuously learn to test and verify diverse applications without human intervention—pointing to a future in which AI-driven testing agents act as trustworthy collaborators, improving software reliability while reducing human effort.

Data Availability. Data/code are available on our website (HxAgent).

REFERENCES

Anonymous. Operator: A vision-enhanced gpt agent for autonomous web interaction. *Technical Report*, 2025.

Anthropic. Computer use and ai agents: A new paradigm for screen interaction. *Technical Blog*, 2025.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.505. URL https://aclanthology.org/2024.acl-long.505/.

Mariya Davydova, Daniel Jeffries, Patrick Barker, Arturo Márquez Flores, and Sinéad Ryan. Osuniverse: Benchmark for multimodal gui-navigation ai agents. *arXiv preprint*, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36, 2024.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are web agents at solving common knowledge work tasks? In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Zhangxuan Gu et al. Ui-venus technical report: Building high-performance ui agents with rft. *arXiv preprint arXiv:2508.10833*, 2025.

Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. In *The Twelfth International Conference on Learning Representations*, 2023a.

Izzeddin Gur, Xinyun Zhang, Xinyang Chen, Shuyan Zhou, Adams Wei Yu, Kevin Lin, and Karthik Narasimhan. Computer use agents: Benchmarking autonomous computer use by language models. *arXiv preprint arXiv:2307.09288*, 2023b.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024a.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2024b. URL https://arxiv.org/abs/2312.08914.

Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*, pp. 9466–9482. PMLR, 2022.

HxAgent. HxAgent. https://anonymous.4open.science/r/hxagent.

Sheng Jia, Jamie Ryan Kiros, and Jimmy Ba. Dom-q-net: Grounded rl on structured language. In *International Conference on Learning Representations*, 2018.

Katalon. Katalon. https://katalon.com/.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. Robula+: An algorithm for generating robust xpath locators for web testing. *Journal of Software: Evolution and Process*, 28 (3):177–204, 2016.

Tao Li, Gang Li, Zhiwei Deng, Bryan Wang, and Yang Li. A zero-shot language agent for computer control with structured reflection. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Shuquan Lian, Yuhang Wu, Jia Ma, Yifan Ding, Zihan Song, Bingqi Chen, Xiawu Zheng, Hui Li, and Rongrong Ji. Ui-agile: Advancing gui agents with effective reinforcement learning and precise inference-time grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8575–8584, 2026.

Yuxi Liang, Zhuo Chen, Shuyan Zhou, Adams Wei Yu, Chenfei Wu, Xing Xie, and Jiang Bian. Taskmatrix.ai: Towards building generalist agents for billions of tasks. *arXiv preprint arXiv:2303.16434*, 2024.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 19498–19508, 2025.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations (ICLR)*, 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 *The Twelfth International Conference on Learning Representations*, 2023a.

Yuhang Liu et al. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. *arXiv preprint arXiv:2501.04575*, 2025.

Zefang Liu and Yinzhu Quan. Econwebarena: Benchmarking autonomous agents on economic tasks in realistic web environments. *arXiv preprint*, 2025.

Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. Chatting with gpt-3 for zero-shot human-like mobile automated gui testing, 2023b. URL https://arxiv.org/abs/2305.09434.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue, 2024.

Michel Nass, Emil Alégroth, Robert Feldt, Maurizio Leotta, and Filippo Ricca. Similarity-based web element localization for robust test automation. *ACM Transactions on Software Engineering and Methodology*, 32(3):1–30, 2023.

Vu Nguyen, Thanh To, and Gia-Han Diep. Generating and selecting resilient and maintainable locators for web automated testing. *Software Testing, Verification and Reliability*, 31(3):e1760, 2021.

Viraj Prabhu, Yutong Dai, Matthew Fernandez, Jing Gu, Krithika Ramakrishnan, Yanqi Luo, Silvio Savarese, Caiming Xiong, Junnan Li, Zeyuan Chen, and Ran Xu. Walt: Web agents that learn tools, 2026. URL https://arxiv.org/abs/2510.01524.

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, 2025. URL https://arxiv.org/abs/2501.12326.

Dezhi Ran, Hao Wang, Zihe Song, Mengzhou Wu, Yuan Cao, Ying Zhang, Wei Yang, and Tao Xie. Guardian: A runtime framework for llm-based ui exploration. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 958–970, 2024.

Selenium. Selenium. https://www.selenium.dev/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Yixiao Song, Katherine Thai, Chau Minh Pham, Yapei Chang, Mazin Nadaf, and Mohit Iyyer. Bearcubs: A benchmark for computer-using web agents. *arXiv preprint*, 2025a.

Yueqi Song, Frank F Xu, Shuyan Zhou, and Graham Neubig. Beyond browsing: Api-based web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11066–11085, 2025b.

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in Neural Information Processing Systems*, 36, 2024.

George Thomas, Alex J. Chan, Jikun Kang, Wenqi Wu, Filippos Christianos, Fraser Greenlee, Andy Toulis, and Marvin Purtorab. Webgames: Challenging general-purpose web-browsing ai agents. *arXiv preprint*, 2025.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2023. URL https://api.semanticscholar.org/CorpusID:261277501.

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, 2025a. URL https://arxiv.org/abs/2504.01382.

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. 2025b. URL https://arxiv.org/abs/2504.01382.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Juyeon Yoon, Robert Feldt, and Shin Yoo. Autonomous large language model agents enabling intent-driven mobile gui testing, 2023. URL https://arxiv.org/abs/2311.08649.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 61349–61385. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zheng24e.html.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*, 2023.

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. *arXiv preprint arXiv:2307.13854*, 2023. URL https://webarena.dev.

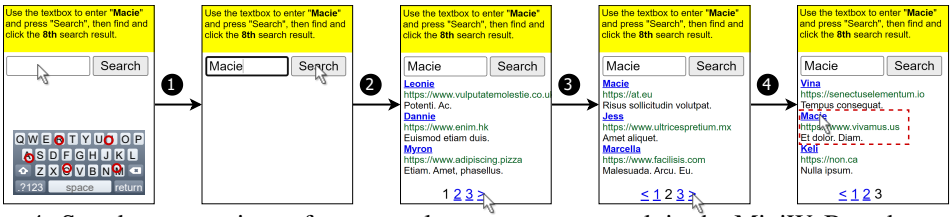


Figure 4: Step-by-step actions of an example `search-engine` task in the MiniWoB++ dataset Liu et al. (2018).

A Limitations

Scope of testing. HxAgent generates the action steps of a test script, focusing on verifying the UI workflow expected in the description. While it verifies each action, it does not generate assertions or test oracles. We do not evaluate fault detection, code coverage, or robustness across application versions, all of which a complete testing pipeline requires.

Script robustness. Generated scripts locate elements by XPath, which can break under layout changes, dynamic content, or slow loading. More robust locators are complementary to our approach, using exist approaches such as (Leotta et al., 2016), (Nguyen et al., 2021), and s (Nass et al., 2023).

Black-box information only. HxAgent uses only what is observable through the browser: the rendered DOM and text, extracted interactable elements, its own action history, accumulated experience, and screenshots. It does not use JavaScript source, instrumentation, execution traces, network traffic, or DOM mutation events. This makes it applicable without source access, but it cannot exploit these signals to reach hidden states or explain failures.

Action extraction. Elements missed by the extractor cannot be selected; a full end-to-end rerun of all baselines with a shared extractor remains future work.

B Motivating Example

An important step in web testing is generating a sequence of actions on the web pages from a natural language description of a task. Fig. 4 displays the task with the description: *"Use the textbox to enter 'Macie' and press 'Search', then find and click on the 8th search result."* The tester is expected to follow this description and perform the described actions on the web pages under test. The tester uses a testing environment (Selenium or Katalon) to execute the application. First, the tester must type "Macie" into the textbox and click the "Search" button. They then need to click the "≥" hyperlink twice before selecting the hyperlink associated with the entry "Macie" as shown in Step 4. The process of the tester's performing a SoA (e.g., clicks, inputs) to interact with the environment (browser), ultimately changes the application's states (e.g., GUI, HTML, or screenshot).

C State-of-the-art Approaches

First, several LLM-based solutions have emerged to automate the manual process Gur et al. (2023a). First, the **UI agents for Web task execution** such as UI-TARS Qin et al. (2025), SeeClick Cheng et al., SeeAct Zheng et al. (2024), CogAgent Hong et al. (2024a), and the models used in WebArena Zhou et al. (2023) and WebLinx Lù et al. (2024), focus on completing the task of "clicking on the 8th search result", i.e., the link with the word "Macie" (Fig. 4). They *do not explicitly generate the executable actions that are needed for test script generation for Web applications*. For example, the WebArena benchmark would check only whether a model finally clicks on the "Macie" link.

Second, in contrast, UI-TARS Qin et al. (2025), SeeClick Cheng et al., and CogAgent Hong et al. (2024a) generate the sequences of actions (SoAs) where an action is expressed as an operation on a screen area defined by **absolute coordinates**, e.g., a "click" on [0,25,75,90] for the "Macie" link. *This is an unreliable locator approach for regression testing*, since even small UI modifications (e.g., shifting, resizing, or adding elements) can break coordinate-based references. It is not trivial to

transform an action description with absolute coordinates into executable actions that are suitable for regression test script generation.

Third, another prominent line of approaches describe **the actions or SoA in natural language**. In our example, at step 4, SeeAct Zheng et al. (2024) could describe the action as *"click on the word "Macie""*. However, there are two appearances of that word on the screen after step 4, which might make the model confused. In fact, it was reported that SeeAct suffers much lower accuracy in SoA generation when outputting executable actions Zheng et al. (2024), which are crucial for Web test script generation.

Fourth, several **prompt-based approaches with demonstrations** such as MindAct Deng et al. (2024), RCI Kim et al. (2024), AdaPlanner Sun et al. (2024), and Synapse Zheng et al. (2023) leverage *few-shot or many-shot prompting* by providing exemplars to the LLMs. As shown in Fig. 4, these demonstrations involve sequences like text entry and result navigation. However, they are limited by the heavy manual effort required for curation and the risk of overfitting to specific patterns.

Fifth, the prompt-based approaches without demonstrations leverage the **self-reflection** capability of LLMs to identify and revise the first critical mistake after a failed attempt, and retry the task. However, *they operate in a reactive repairing strategy, where corrections are attempted only after an entire action trajectory has failed.* Moreover, without maintaining a record of prior state-action pairs, for instance, at step 3 in Fig. 4, the model might mistakenly click the hyperlink labeled "1," returning to the previous screen. A heuristic based on textual similarity (e.g., matching the word "Macie") fails, as multiple links share that label in steps 3 and 4. Relying on DOM indices is also unreliable, as not all search results are visible on a single page due to pagination.

D Long-term Experience Reinforcement

Rule extraction and storage. The Rule Extractor receives (i) the just-completed trajectory with its success/failure label and (ii) the full set of existing rules, and is prompted to extract a new textual rule that generalizes the observed behavior. Because all current rules are provided as part of this prompt, the LLM produces a rule that differs from those already stored, so the set is not duplicated and grows only when a genuinely new heuristic emerges. The extracted rules act as textual heuristics and, together with the correct SoAs serving as few-shot examples, populate the experience section of the main prompt (Fig. 9), which is updated dynamically during the training phase. We do not yet perform proactive consolidation or pruning of stored experience; for larger or longer-lived stores, consolidation of similar trajectories into canonical rules, utility- or age-based pruning, and periodic revalidation of stored rules against the current UI are natural extensions, which we leave to future work.

Training phase.

Initially, the experience is set to empty. During the training phase, the experience is updated once the task is successful. For update, the (in)correctly generated sequences of actions are saved as experience. For an incorrect sequence, we use Rule Extractor 4 to extract a textual rule from the (in)correct sequence, ensuring it differs from the current rules, and update it to the current set of rules. These rules act as textual heuristics, along with the correct SoA serving as few-shot examples embedded in the experience section of the main prompt (Fig. 8). This part of the main prompt is dynamically updated in the training phase. We also track the average of the correct SoA generated over a number of recent episodes in training (called *moving average*). Consider Experience Reinforcement with task `click-tab-2`. The first episode initializes the task: "*Switch between the tabs to find and click the link 'et'.*" HxAgent uses Iterative LLM-based Agent Planning to generate and execute the action "*click the link 'et'*," completing the task. The correct sequence is then stored in the Experience (Algorithm 1, lines 8–13). In the second episode, a similar task is initialized. The Experience guides the agent, but this time the generated sequence is incorrect. It is saved as an incorrect sequence, and Rule Extractor creates the rules. In the third episode, with updated Experience (correct sequences+rules), it *switches tabs* before clicking the link, creating a correct sequence. This is added to Experience (Algorithm 1, lines 14–18). Specifically, if the task remains unfinished at $t = MAX_t$ (the maximum number of steps), it is categorized as an incorrect execution (Algorithm 1, lines 32–35).

Evaluation phase.

We select the optimal experience from training to be used as input for the evaluation phase. An optimal experience is the combination of the correct SoA and rules that exhibits an upward trend and stable pattern from the training, decided by the moving average. The iterative planning is the same, except for the fixed optimal experience $E_{optimal}$. Hence, with experience, the reasoning for the next action a_{t+1} at the iteration t become $a_{t+1} \sim \pi(. \mid o_t, \mathcal{A}(o_t), M_t, E_{optimal})$.

E Iterative Agent Planning

E.1 Implementation Details

Algorithm 1 displays the pseudo-code of our iterative LLM-based Agent Planning algorithm. It takes as input a natural-language task description $task$ and a webpage url for the process. First, the Sequence Actions Memory (M_t) is also initialized for the given task with an empty state-action history (line 2). The workflow begins by extracting either the DOM or screen images from the active webpage in the test environment (line 4). Next, an iterative planning phase is executed (lines 6–31), starting with the Content Extractor, which parses the webpage to derive the initial state and a set of feasible actions (line 6).

At line 7, the LLM is invoked to reason whether the current trajectory has achieved the goal ($success = true$) or an early stopping condition has been met ($stop = true$). If either case occurs, the algorithm logs the corresponding outcome in the Experience E—$(M_t, 1)$ for success or $(M_t, 0)$ for failure—during training mode, and then terminates the loop (lines 8–19). This process allows both successful and failed trajectories to contribute to improving the Experience for future planning.

If the task is still ongoing, the LLM is queried to infer the next action a_t given the current observations $(\hat{s}_t, \mathcal{A}(o_t))$, the Sequence Actions Memory M_t, and the Experience E (line 20). In cases where the generated action is duplicated (i.e., multiple identical candidates appear), the LLM is invoked again using the visual observation o_t^{img}—a screenshot of the current webpage—to select the correct element (lines 21–23). When the chosen action corresponds to an input operation, the LLM is prompted once more to generate the appropriate content for the target input field (lines 24–26). The completed action is then appended to the Sequence Actions Memory together with the current state (line 28).

Next, the Test Environment executes the selected action on the webpage to update the interface and retrieve the new state s_t (lines 29–30). The loop counter t is incremented, and the process continues until the task is completed, an early stop condition is reached, or the iteration limit MAX_t is exceeded. If the maximum number of iterations is reached without completion, the trajectory is recorded as a failed episode $(M_t, 0)$ when in training mode (lines 32–35). Finally, the Experience E is updated through the function $\phi_{LLM}(\tau_t, E)$ to refine future action planning (line 37).

Let us take the iteration #4 of Fig. 4 as an example. The HTML content of the webpage and its action transition arrow represent a state-action pair. At the end of iteration 3, M_t currently stores the state-action pairs of the history including the ones at 1) the iteration t=1 (No search results displayed, input text: “Macie”), 2) the iteration t=2 (No search results displayed, input text: “Macie”, clicking “Search”), and 3) the iteration t=3 (three links displayed: “Leonie”, “Dannie”, “Myron”, pagination links, and clicking the “$\leq$” hyperlink). Then, an action a_3 is performed to transit the 3^{rd} page to the 4^{th}. Next, we extract the current state s_4 and feasible actions A_4 (including the input search field, clicking Search button, clicking the “Macie”, “Jess”, and “Marcella” hyperlinks, clicking the “$\leq$” hyperlink, etc.). At the iteration $t = 4$, the LLM reasons and decides the next action a_4 to be "clicking the “$\geq$” hyperlink" based on the observations of current webpage state s_4 and its corresponding actions A_4, the sequence actions memory M_t up to iteration t_3, and the experience E. Then the procedure at the end of iteration $t = 4$ continues and moves on to $t = 5$ until the task is finished. Due to maintaining the contents/states of the steps, it does not stop and click at the “Macie” hyperlink at the 4th search result, and continues to “Macie” link in the next iteration.

```
The search results are displayed in a div with the id 'page-content'.
The first three search results are: 'Leonie', 'Dannie', and 'Myron' with additional links and descriptions.
There is also a pagination section with links to navigate through the pages.
```

Figure 5: State in summarized natural language description format extracted from 3^{rd} webpage of Figure 4.

```
ITERATIVE_ACTION_REASONING_PROMPT = '''You are a web assistant... You will complete the task
      by taking a series of steps. Each step is a description of the action you take and the
      specific item, entity, or element on the website that the action is applied.
{experience}
# Here is the actual task.
{sequence_actions_memory}
After completing the above steps, you reach a state: {state} where the following feasible
     steps exist:
{feasible_actions}
POSSIBLE NEXT ACTION #1: {action_1} ...
Your job is to choose the most possible next steps to help you complete the task....
# The JSON response must strictly follow these rules:
    "chosen_action": ... (the index of the potential action that you choose)
    "action_description": ... (a string describing the action you choose)
    "reason": (describing why you choose the action)...
```

Figure 6: Iterative Action Reasoning Prompt (Main Prompt).

E.2 Task Description & Content Extractor

The input task is described in natural language. This description is recorded in the Sequence Actions Memory (line 1, Algorithm 1). HxAgent utilizes Content Extractor 1 (lines 6, Algorithm 1) to extract information from the initial webpage and update the information after the agent performs actions (lines 28, Algorithm 1). The webpage is provided to the LLM either in DOM format or as screen images. This module has two functions: (1) feasible action extraction and (2) state extraction.

First, the LLM receives the DOM of the page and generates possible actions described as JSON objects. We extract only the visible and interactable HTML elements attached to specific user events using the DevTools API. We also extract relevant surrounding contexts. For example, the context for a button and a container could be its inner content, while an input field's context could be its label. If necessary, we traverse up the DOM tree to gather more information from parent nodes and bind accordingly. After that, each interactable element is mapped with its operation, resulting in a list of possible actions on the current webpage in a JSON file.

Second, HxAgent extracts and represents the state of a webpage in one of two formats, based on the DOM size. For smaller DOMs, it generates a simplified HTML containing interactable elements and a mapping to the original HTML file. For large DOMs that exceed the LLM's context, it produces a text summary generated by the LLM from the webpage screenshot using vision capability (Fig. 5).

F Datasets

This appendix complements Section 4 by presenting detailed descriptions and statistics of the datasets employed in our experiences. The first, the Real-world dataset (Table 7), was collected from a set of well-known websites including YouTube, LinkedIn, Facebook, Google, Amazon, StackOverflow, and Expedia. For each website url, we designed a set of task templates (e.g., *"Subscribe to the {} channel"*, *"Login to YouTube with username {} and password {}"*). Each template was then instantiated into specific tasks (e.g., *"Subscribe to the 'TEDx Talks' channel"*). The final dataset consists of 50 task instances per website across 7 websites, yielding a total of 350 task instances.

The second dataset is from the **MiniWoB++** benchmark Liu et al. (2018). Each environment or application is defined for a task, having a description (e.g., *"Choose an item from a list"*), a set of utterances (e.g., *"Bobine"*, *"Betty"*) that were randomly chosen for each episode to form a complete natural language task instance (e.g., *"Select Betty from the list..."*). We sampled 44 tasks from the dataset, each with 25 instances, yielding a total of 1,100 task instances.

The third dataset is **OnlineMind2Web** Xue et al. (2025b), a recent and actively maintained benchmark derived from the original Mind2Web dataset Deng et al. (2024). It contains 300 tasks collected from 136 real-world websites across domains such as clothing, food, housing, and transportation. The

```
SEQUENCE_ACTIONS_MEMORY_PROMPT = '''
You are visiting the website title:
    {title}
You are asked to complete the following
    task: {task}
You have completed the following steps:
> STATE #1: {state_1}
> STEP #1: {action_1}
> STATE #2: {state_2}
> STEP #2: {action_2} ...
```

Figure 7: Sequence Actions Memory Section of Main Prompt.

```
EXPERIENCE_PROMPT = '''
# Here are the history of your trials
SUCCESS TRIAL #1: Task: {task_1}
STEP #1: {task_1_action_1}
STEP #2: {task_1_action_2}
...
SUCCESS TRIAL #8: Task: {task_8}
STEP #1: {task_8_action_1}
STEP #2: {task_8_action_2}
...
# Rules extracted from past attempts,
    use to evaluate your policy:
RULE #1: {rule_1}
RULE #2: {rule_2}...
```

Figure 8: Experience Section of Main Prompt.

Table 7: Real-world dataset statistics.

Application	#Actions/Screens	#Screens	SoAs
YouTube	291 ± 144	2 ± 1	3 ± 1
LinkedIn	146 ± 36	3 ± 1	4 ± 1
Facebook	260 ± 110	3 ± 1	3 ± 1
Google	91 ± 189	3 ± 1	4 ± 1
Amazon	330 ± 82	2 ± 1	3 ± 1
StackOverflow	296 ± 151	3 ± 1	5 ± 5
Expedia	135 ± 22	4 ± 1	9 ± 4
Total/Avg.	221	3	4

Table 8: Baseline approaches

Approach	Model	demos	Feedback
WebNT5	T5+Fine-tuning	✓	-
HTML-T5-XL	T5+Fine-tuning	✓	-
WebGUM	ViT+T5+Fine-tuning	✓	-
WGE	RL	✓	-
CC-Net	RL + SL	✓	-
RCI	LLM Prompting	✓	✓
AdaPlanner	LLM Prompting	✓	✓
Synapse	LLM Prompting	✓	-
Li et al.	LLM Prompting	-	✓
SeeAct	LLM Prompting	-	✓

dataset classifies tasks into three difficulty levels: *easy* for those requiring up to 5 steps to complete (83 tasks), *medium* for those requiring 6–10 steps (143 tasks), and *hard* for those involving 11 steps or more (74 tasks). In our experiments, 15 tasks were excluded because four websites were inaccessible, resulting in a final set of 285 tasks from 132 websites used for evaluation.

G Result Analysis

G.1 Analysis of the Real-world dataset

As seen in Table 1, HxAgent outperforms Li *et al.* by 56.8%, with the gap being most significant in tasks that require navigating more than two screens and completing at least two actions. In this real-world dataset, websites are more dynamic than those in MiniWoB++, particularly when dealing with dropdown options (e.g., search tasks in `Google` or filling in locations to book flights on `Expedia`, Table 1). These dynamic features pose a challenge for the staged planning approach of Li *et al.*, which plans all actions on a screen at once. The reflection alone in Li *et al.* learns from the entire

passing/failing sequences, thus, less flexible than its combination with ReAct and short-term memory as in HxAgent. Large state changes by an action often lead to the failure of subsequent actions in the plan. Moreover, Li *et al.* use the entire HTML for state representation, resulting in exceeding LLM context or selecting invalid actions.

In contrast, HxAgent performs well on the tasks in this dataset. These tasks involve state changes after an action, making the iterative planning in ReAct effective in adapting to changes. These tasks involve navigating up to four screens, performing up to five actions, processing large HTML files.

However, some tasks still pose greater challenges due to highly dynamic layouts and varying action sequences across runs. For instance, ordering a gift card may require navigating to the product detail page and filling out a "Gift Card Details" form, whereas ordering a smartphone might only require clicking "Add to Cart" after searching. These variations in layout/actions make planning with prior experience more difficult, leading to occasional inaccuracies.

For `Expedia` tasks, it achieves the lowest `Exact-Match` of 32% and `Prefix-Match` score of 64.4%. It struggles to handle real-world date pickers to select departure and return dates. HxAgent shows a performance with 80% `Exact-Match` and 90% `Prefix-Match` in `Stackoverflow` tasks. There is one task that requires filling a long form with constraints, resulting an incorrect actions.

We also evaluated the best-performing variant of SeeAct Zheng et al. (2024), $\texttt{SeeAct}_{choice}$, on the Real-world dataset. This model, which grounds actions using textual choices, was used to assess how a strong web-task model could be adapted to a new dataset. SeeAct achieved an `Exact-Match` rate of 72.4% and a `Prefix-Match` rate of 83.8%. It can handle most tasks on platforms such as `YouTube`, `Linked In`, `Google`, and `Amazon`. However, its performance drops to 66.7% on `Facebook`, and it fails to complete tasks on `Expedia` due to difficulties handling real-world date pickers for selecting departure and return dates.

G.2 **Analysis of the OnlineMind2Web**

Table 2 shows the results obtained from running three approaches on the Online-Mind2Web dataset. HxAgent consistently outperforms the baselines across tasks in Exact-Match at all difficulty levels. However, its performance varies by difficulty, with 81.3% on easy tasks, 40.7% on medium tasks, and 30.0% on hard tasks. In this dataset, the websites are more complex than those in the Real-world dataset, particularly due to dropdown menus, calendar pickers, and similar components. Some UI elements are clickable while others are disabled, and task completion often depends on executing the correct sequence of clicks, further complicated by errors and formatting artifacts in the UIs. These dynamic features present significant challenges for the staged planning approaches of Li *et al.* and SeeAct. For Li *et al.*, using the DOM to represent the state of a webpage leads to token limitations in the LLM. In contrast, SeeAct suffers from incomplete descriptions of UI elements, which often result in suboptimal action choices.

For easy tasks, HxAgent has relative improvements of 2.03X over Li *et al.* and 27.4% over SeeAct, with the substantial gains occurring in tasks that involve complex interface elements such as duplicate web elements, calendar pickers, and dropdown menus. For example, in the task *"Find Florida internship programs in the Mayo Clinic College of Medicine and Science,"* Li *et al.* was unable to reach the target page, whereas SeeAct failed as it attempted to resolve duplicate "Search" buttons.

For medium tasks, HxAgent achieves relative improvements of 2.75X over Li *et al.* and 33.9% over SeeAct. These tasks require managing state changes triggered by user actions, where HxAgent's iterative planning is particularly effective for adapting to such dynamic interactions. For example, when a user enters a zip code into a search box, the system must wait for the resulting state change and the appearance of a dropdown list before selecting the appropriate item. In the task *"Browse pediatricians near zip code 90028 who specialize in Internal Medicine and have a rating of at least 4 stars,"* after the user inputs "90028" into the Location field, the system must wait for the state update and then correctly select *"Los Angeles, CA 90028"* from the dropdown.

For hard task, HxAgent achieves improvements of 5.3X over Li *et al.* and 3X over SeeAct. These tasks involve complex UI elements such as dropdowns and require multiple steps in the execution process. They also combine characteristics of both easy and medium tasks, as they mix relatively straightforward lookup operations with more intricate interactions that demand careful sequencing of actions. For instance, in the task *"Browse dermatologists within 10 miles of zip code 10019 and*

Table 9: Action Sequence Generation on MiniWoB++. Exact-match (%), Correct runs (975 total).

Approach	Exact-Match (%)	# Correct runs
WebNT5	59.1%	532
HTML-T5	88.3%	861
WebGUM	89.9%	877
WGE	64.3%	627
CC-Net	93.8%	915
RCI	89.5%	926
A.Plan.	88.0%	853
Synapse	97.3%	958
Li (4o)	91.7%	949
SeeAct	69.9%	682
HxAgent	97.4%	950
WALT	86.9%	847

filter by only those who accept Blue Medicare Advantage," the agent must first enter the zip code "10019" into the Location field, wait for the interface to update, and then correctly select *"New York, NY 10019"* from the dropdown before applying the remaining filters. Satisfying the task requirements thus depends on completing the full sequence of steps. Such tasks often unfold as relatively long action chains, typically requiring three to four sub-actions per screen before progressing further.

Although HxAgent demonstrates substantial improvements over Li *et al.* and SeeAct, HxAgent still struggles with complex components such as datetime pickers, enter-to-submit search boxes, and highly intricate websites that may cause LLM context overflows.

G.3 Analysis of the MiniWoB++

As seen in Table 3, which reports 975 task instances from 39 tasks, HxAgent achieves a 97.4% average `Exact-Match` across the shared set of tasks among the baselines. This set of tasks varies in screens and actions, with screens ranging from 1 to 6, and a maximum of 6 actions needed to complete the task. Tasks with fewer screens and actions tend to have a higher average `Exact-Match`, with 97% to 100% reported for tasks with fewer than 3 screens and actions. This percentage drops marginally to 93% for tasks containing more than 3 screens and actions, with a maximum of 4 screens and 9 actions. With `Prefix-Match` of 99%, in many cases where HxAgent does not fully succeed in a task, the generated sequences contain correct prefixes of action sequences, reducing correction effort.

Although HxAgent achieves comparable `Exact-Match` with the best baselines in each category, note that it *eliminates the need for human demonstrations* (see Table 9). Specifically, it matches the performance of WebGUM, despite WebGUM utilizing 401K demonstrations with web screenshots to jointly fine-tune the vision encoder ViT and T5. HxAgent performed better than CC-Net (BC + RL) in 10 task instances, with CC-Net employing 2.4M human's demonstrations. Synapse, with perfect results, relies heavily on 154 quality hand-crafted exemplars, some of which further contain task-specific filter prompts to convert raw HTML states into clean observations. Their solution may result in overfitting MiniWoB++, as they have a high number of few-shot exemplars and the reducing of the difficulty of tasks requiring exploration in multiple states with customized and obvious observation.

Among the approaches that do not rely on human demonstrations, HxAgent performs slightly better than Li *et al.* (97% vs. 94%) and substantially outperforms SeeAct by 20% in `Exact-Match` accuracy. We further stratified results on 125 challenging task instances drawn from five MiniWoB++ tasks (Table 4). These tasks are considered challenging because they require navigation across an average of five screens and the execution of approximately seven actions to complete. As seen in Table 4, HxAgent shows the effectiveness (**81.6% `Exact-Match`**) in more challenging tasks over Li *et al.* and SeeAct, which have only 10% and 8% `Exact-Match` in these complex actions.

As reported in Table 3, SeeAct achieves an `Exact-Match` rate of 69.9%, which underperforms Li *et al.* On the 125 most challenging tasks, its `Exact-Match` rate drops to 8%, with successful completion limited to the `flight.AA` tasks (Table 4). Our further analysis reveals that web components, such as datetime pickers and dropdowns, remain significant obstacles for SeeAct. This is also true for other

non-descriptive elements like buttons with only icons, which require action-oriented descriptions to be processed effectively. In brief, *HxAgent achieves higher accuracies on the challenging tasks, while the baselines struggle with such intricate actions.*

G.4 Accuracy by Task Complexity

We evaluate the performance of HxAgent with respect to task complexity, measured by the length of action sequences and the number of actions on webpages.

Next-step prediction accuracy by action sequence length. In Fig. 2a, HxAgent achieves 100% accuracy on the first step for all tasks in the MiniWoB++ and Real-world datasets. However, as a task requires more action steps, accuracy slightly decreases. (For step 7, a slight increase is observed due to the smaller number of tasks reaching this stage, resulting in a relatively higher value). For two tasks reaching Step 11, it did not predict the correct action, despite the successes of the previous 10 steps. A similar trend is observed in the OnlineMind2Web dataset, where `Exact-Match` drops from 81.3% on easy tasks to 40.7% on medium tasks and 30.0% on hard tasks.

Exact-Match sequence accuracy by number of actions per screen. The action space may contain a large number of selectable actions per screen. As shown in Fig. 2b, HxAgent maintains high `Exact-Match` accuracy even when the action space is large, with up to 500 possible actions per screen.

There are eight outliers (shown in orange): two tasks with sequence lengths of up to 11 steps, where `Exact-Match` falls below 70%. For tasks involving dynamic content, such as confirmation popups in `linkedin-3` and story posts in `facebook-5`, HxAgent performs reasonably well but with limited consistency. Tasks requiring the selection of hidden options (`flight-Alaska-auto`) or actions that span multiple UI elements (e.g., date-pickers in `expedia-3` and `expedia-5`) achieve only 30% `Exact-Match`. Similarly, for tasks involving complex forms (e.g., `stackoverflow-3`), the system generated invalid data during form filling.

G.5 Cost Analysis

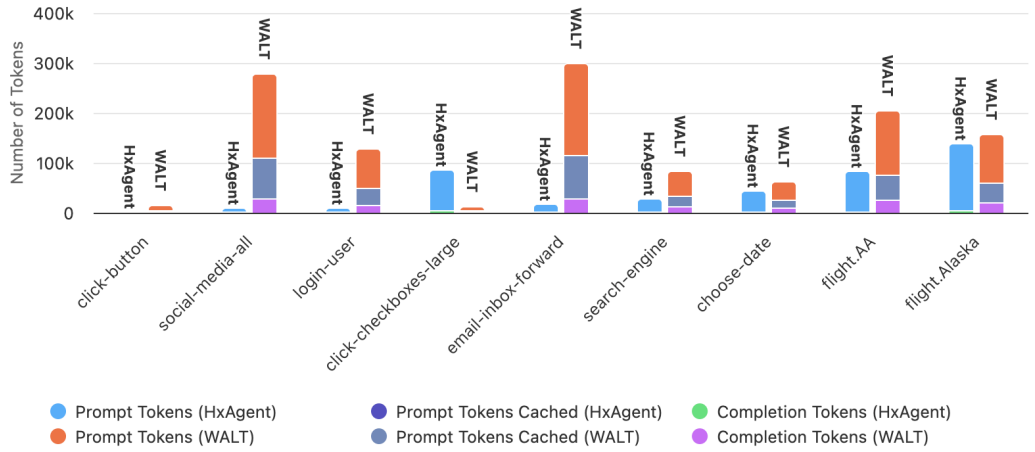


Figure 9: Token Consumption Comparison by Dataset and Tool.

We conduct a cost analysis on representative MiniWob++ tasks, ranging from simple tasks such as ‘click-button‘ and ‘login-user‘ to more complex ones including ‘flight.AA‘ and ‘flight.Alaska‘. Figure 9 shows the total token consumption of each method.

The results show that HxAgent achieves lower and more stable token usage than the baselines, especially on tasks with complex interfaces or long interaction sequences. In contrast, WALT consistently consumes substantially more tokens, particularly on ‘social-media-all‘, ‘email-inbox-forward‘, and ‘flight.AA‘, indicating higher inference costs for handling complex UI interactions.

Tasks involving dynamic forms, date pickers, and multi-step navigation generally increase token consumption across all methods. Nevertheless, HxAgent remains more cost-efficient while achieving

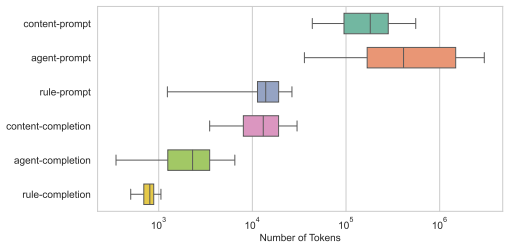


Figure 10: #tokens for each component's input/output.

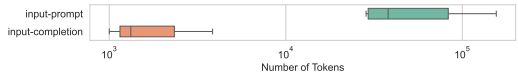


Figure 11: #tokens for input generator (log scale).

competitive or superior performance. Compared with WALT, HxAgent uses only about one-third of WALT's total tokens, while its output tokens account for just 8.9% of WALT's output token count.

Figures 10 and 11 provide a detailed breakdown of token consumption across different components. Figure 10 shows that the agent module consumes the largest number of tokens, particularly in the prompt stage, indicating that reasoning and action planning dominate the inference cost. In contrast, the rule-completion component requires relatively few tokens.

Figure 11 further shows that, within the input generator, prompt tokens account for most of the cost, while completion tokens remain comparatively small. Overall, these results suggest that contextual grounding and webpage representation are the primary sources of token consumption in the framework.