

# WEBXSKILL: Skill Learning for Autonomous Web Agents

Zhaoyang Wang<sup>1</sup>, Qianhui Wu<sup>2,\*</sup>, Xuchao Zhang<sup>2</sup>, Chaoyun Zhang<sup>2</sup>, Wenlin Yao<sup>2</sup>,  
Fazle Elahi Faisal<sup>2</sup>, Baolin Peng<sup>2</sup>, Si Qin<sup>2</sup>, Suman Nath<sup>2</sup>, Qingwei Lin<sup>2</sup>, Chetan Bansal<sup>2</sup>,  
Dongmei Zhang<sup>2</sup>, Saravan Rajmohan<sup>2</sup>, Jianfeng Gao<sup>2</sup>, Huaxiu Yao<sup>1,\*</sup>

<sup>1</sup>University of North Carolina at Chapel Hill      <sup>2</sup>Microsoft  
{zhaoyang, huaxiu}@cs.unc.edu    {qianhuiwu, xuchaozhang}@microsoft.com

## Abstract

Autonomous web agents powered by large language models (LLMs) remain brittle on long-horizon browser workflows. A key bottleneck is a grounding gap in existing skill formulations: textual workflow skills provide natural language guidance but cannot be directly executed, while code-based skills execute without giving the agent step-level guidance for adaptation or recovery. We introduce WEBXSKILL, a framework that bridges this gap with executable skills, each pairing a parameterized action program with step-level natural-language guidance. WEBXSKILL operates in three stages: skill extraction mines reusable action subsequences from readily available synthetic agent trajectories and abstracts them into parameterized skills, skill organization indexes them into a URL-based graph for context-aware retrieval, and skill deployment exposes two complementary modes, grounded mode for fully automated execution and guided mode where skills serve as step-by-step instructions the agent follows with its native planning. WEBXSKILL demonstrates consistent improvements on WebArena, WebVoyager, and Online-Mind2Web. We further find that better skill deployment mode depends on a model’s plan and execution capability. The code is available at <https://github.com/aiming-lab/WebXSkill>.

## 1 Introduction

Large language models (LLMs) have enabled autonomous web agents that interact with real websites through browser actions such as clicking, typing, and navigating (Agashe et al., 2025; Murty et al., 2024; Ning et al., 2025; Zhou et al., 2024; He et al., 2024b). Yet these agents remain brittle on complex multi-page workflows because they rarely retain and reuse procedural interaction knowledge, forcing them to re-plan long action sequences from scratch even for recurring routines (Xue et al.,

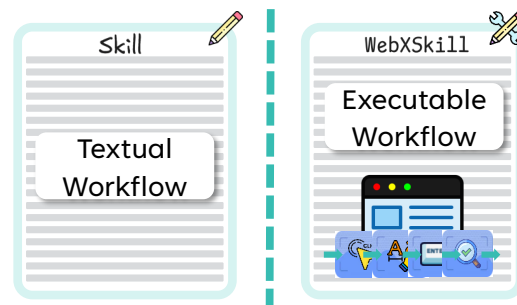


Figure 1: WEBXSKILL equips web agents with executable skills that pair an action program with step-level natural-language guidance for planning assistance.

2025; Li et al., 2025). For example, when an agent completes a complex task, the procedural guidance embedded in the trajectory is often discarded, and the next time the agent encounters a similar task it must re-derive the entire action sequence, increasing the risk of planning and execution errors.

Recent work has begun to address this gap through skills, reusable knowledge units that encode common interaction patterns (Sodhi et al., 2023; Wang et al., 2024; Prabhu et al., 2026; Zheng et al., 2025; Wang et al., 2025). However, existing skill formulations for web agents suffer from a grounding gap. Workflow-based approaches such as AWM (Wang et al., 2024) represent skills as natural language instructions (e.g., “search for the product, then add it to cart”) that guide planning but cannot be directly executed, requiring the agent to translate each instruction into concrete browser actions and reintroducing grounding errors. Code-based approaches like SkillWeaver (Zheng et al., 2025) and WALT (Prabhu et al., 2026) ground skills in executable code or action scripts, but deploy them as opaque black-box calls without step-level guidance, making the agent unable to understand, adapt, or recover when execution fails mid-way. Note that detailed code scripts are often not provided to the agent, making it impossible for the agent to learn from the skill’s internal logic or to

\*Corresponding authors.

flexibly adapt to actual situations.

In this paper, we propose WEBXSKILL, a framework that bridges the grounding gap by introducing *executable skills* that pair parameterized action programs with step-level natural-language guidance, as shown in Figure 1. Each skill carries both a concrete sequence of browser operations and semantic annotations (name, description, typed parameters, and per-step guidance), making it simultaneously executable by the runtime and readable by the agent. This dual nature enables two complementary deployment modes: a *grounded mode* where the agent invokes a skill as an atomic tool call and the runtime auto-executes the underlying action sequence, and a *guided mode* where skills are surfaced as step-by-step instructions the agent follows using its native browser actions, preserving autonomy for adaptation. In contrast to prior work that hides the internal logic of skills, our two modes offer the agent readable details of the selected skill, providing agent autonomy to adapt in unexpected states. WEBXSKILL obtains such skills by: (1) skill extraction, which mines reusable action subsequences from abundant synthetic agent trajectories and abstracts them into parameterized skills via LLM-based generalization; (2) skill organization, which indexes skills into a URL-based skill graph that maps web pages to their applicable skills, enabling context-aware retrieval; and (3) skill deployment, which selects between grounded and guided execution depending on the task and model capabilities. Our contributions are:

- We identify a grounding gap between textual and code-based skill formulations for web agents, and propose executable skills that pair parameterized action programs with step-level natural-language guidance, transparent to the agent while maintaining efficiency.
- We instantiate this as WEBXSKILL, which extracts skills from low-cost synthetic trajectories (avoiding test-time leakage), organizes them in a skill graph for context-aware retrieval, and deploys them in grounded or guided mode for skill transfer across websites.
- WEBXSKILL improves task success over strong baseline methods on WebArena, WebVoyager, and Online-Mind2Web. We further show that our deployment mode can be adaptively selected based on model capabilities.

| Method                           | Exec. | Guid. | Acq.  | Ctx. |
|----------------------------------|-------|-------|-------|------|
| AWM (Wang et al., 2024)          | ✗     | ✓     | Test  | ✗    |
| SkillWeaver (Zheng et al., 2025) | ✓     | ✗     | Expl. | ✗    |
| ASI (Wang et al., 2025)          | ✓     | ✗     | Test  | ✗    |
| WALT (Prabhu et al., 2026)       | ✓     | ✗     | Expl. | ✗    |
| <b>WEBXSKILL</b>                 | ✓     | ✓     | Traj. | ✓    |

Table 1: Design axes of skill-based web agent methods. Textual skills guide but cannot execute; code-based skills execute but lack step-level guidance.

## 2 Related Work

**Web Agents.** Web agents have rapidly evolved with LLMs and vision-language models that autonomously interact with browsers to complete complex tasks (Nakano et al., 2021; Ning et al., 2025; He et al., 2024b; Li et al., 2025). Early work established the ReAct paradigm (Yao et al., 2023), interleaving chain-of-thought reasoning with browser actions, and benchmarks such as Mind2Web (Deng et al., 2023), WebVoyager (He et al., 2024a), and WebArena (Zhou et al., 2024) evaluate agents on live or self-hosted sites. Complementary directions include planning, agent collaboration, and memory (Gao et al., 2023; Hong et al., 2024; Wu et al., 2024; Chhikara et al., 2025; Yang et al., 2026a), data synthesis (Sun et al., 2025; Pahuja et al., 2025; Wang et al., 2026), and reinforcement learning (Wei et al., 2025; Lu et al., 2025; Yang et al., 2026b). These agents operate over low-level browser actions, which makes long-horizon tasks fragile; tool-use and skill learning have emerged to provide reusable abstractions.

**Tool-Use and Skill Learning.** Hybrid computer-use agents improve efficiency through multi-action prediction and coding-oriented skills (Yang et al., 2025; Song et al., 2025). Building on this trend, a growing body of work equips web agents with reusable higher-level knowledge (Wang et al., 2024; Zhou et al., 2025; Zheng et al., 2025; Wang et al., 2025; Prabhu et al., 2026; Sodhi et al., 2023; Sarch et al., 2024). Table 1 summarizes how these methods differ along four axes: executability (**Exec.**), step-level guidance (**Guid.**), acquisition strategy (**Acq.**), and context-aware skill retrieval (**Ctx.**). Textual workflow methods such as AWM (Wang et al., 2024), StepP (Sodhi et al., 2023), and ICAL (Sarch et al., 2024) represent skills as natural language instructions that guide planning but cannot be directly executed, leaving a gap between “what to do” and “how to execute it”. Executable-

skill methods close one side of this gap by making skills directly invocable but sacrifice step-level guidance: SkillWeaver (Zheng et al., 2025) discovers patterns through autonomous website exploration (**Expl.**) and compiles them into Python APIs whose internal logic is opaque to the agent; ASI (Wang et al., 2025) induces programmatic skills from successful episodes but acquires them from test-time task trajectories (**Test**), introducing the risk of evaluation leakage; WALT (Prabhu et al., 2026) reverse-engineers built-in website functionality into deterministic tools with validated input schemas, again as black-box calls. The internal logic of these code-based skills is hidden from the agent, making it unable to understand, adapt, or recover when execution fails unexpectedly. WEBXSKILL bridges this grounding gap along all axes. Every skill couples an executable program with step-level natural-language guidance, enabling both grounded execution for efficiency and guided step-by-step use for agent autonomy. Both textual guidance and action programs are transparent and readable to the agent, allowing it to learn from the skill’s internal logic and flexibly adapt to actual situations. Skills are extracted from independently synthesized trajectories (**Traj.**), avoiding costly exploration and the risk of test-data leakage. Extracted skills are then organized into a URL-based graph for context-aware retrieval rather than a flat library surfaced regardless of agent’s current page state and execution context.

### 3 Method

**Problem Setup.** Given a user task  $q$ , web agent  $\pi$  repeatedly generates browser actions over webpage observations. At each step  $t$  the agent receives observation  $o_t$  that includes both a textual description of the webpage (e.g., accessibility tree) and a screenshot, and emits an action  $a_t$  from a primitive action space  $\mathcal{A}_{\text{prim}} = \{\text{click}, \text{input}, \text{scroll}, \text{navigate}, \dots\}$ , until the agent signals completion or a step budget is reached. In this web agent setting, the agent often reasons from scratch at every step, with no mechanism to reuse procedural knowledge from prior interactions, even when facing recurring patterns such as searching for a product or filling forms.

**Overview.** To mitigate this limitation, we propose WEBXSKILL, which augments the web agent with a library of executable skills that encode frequently recurring interaction patterns as reusable

and readable action programs. Figure 2 shows the three stages of our method: skill extraction, skill organization, and skill deployment.

#### 3.1 Skill Extraction

The goal of skill extraction is to distill agent trajectories into a compact library of parameterized skills. We use readily public available agent trajectories as the data source and exclude any trajectories that are derived from the evaluation benchmarks, avoiding the need for website exploration and the risk of test-data leakage.

**Trajectory Abstraction.** Given a corpus  $\mathcal{T} = \{\tau_1, \dots, \tau_N\}$ , where each trajectory records the step-by-step actions an agent took to complete (or attempt) a task, we use an LLM to identify reusable subsequences and abstract them into parameterized skills. Specifically, for every  $\tau_i$ , we construct a structured representation that includes the task description, page URL at each step, the action taken (action type, target element and parameters), and the agent’s reasoning. The LLM is prompted to: (1) identify action subsequences that represent a coherent, reusable operation (e.g., “search for a product by keyword”), (2) abstract concrete action values into typed parameters (e.g., replacing a specific search query with a query: `str` parameter), and (3) annotate each action step with natural language guidance describing its purpose and reasoning.

**Skill Curation and Verification.** To keep the skill library compact without sacrificing coverage, we first use an online deduplication strategy to compare each candidate skill against the existing library before insertion, combining rule-based similarity and embedding-based semantic similarity approaches. The LLM then decides whether to add, update, or skip the candidate skill. We further verify each skill’s action sequence by executing it in a test environment and discard any skill that fails to run, which is crucial for effective skill deployment, as studied in our later ablation (Section 5.3): reliability is the largest contributor to final accuracy.

#### 3.2 Skill Organization

Extracted skills should be efficiently retrievable at inference time, otherwise massive skill descriptions will overwhelm the agent context. Unlike existing methods that often treat skills as a flat library or API set to be selected or ranked at every step, we exploit the page-level structure of web browsing: a

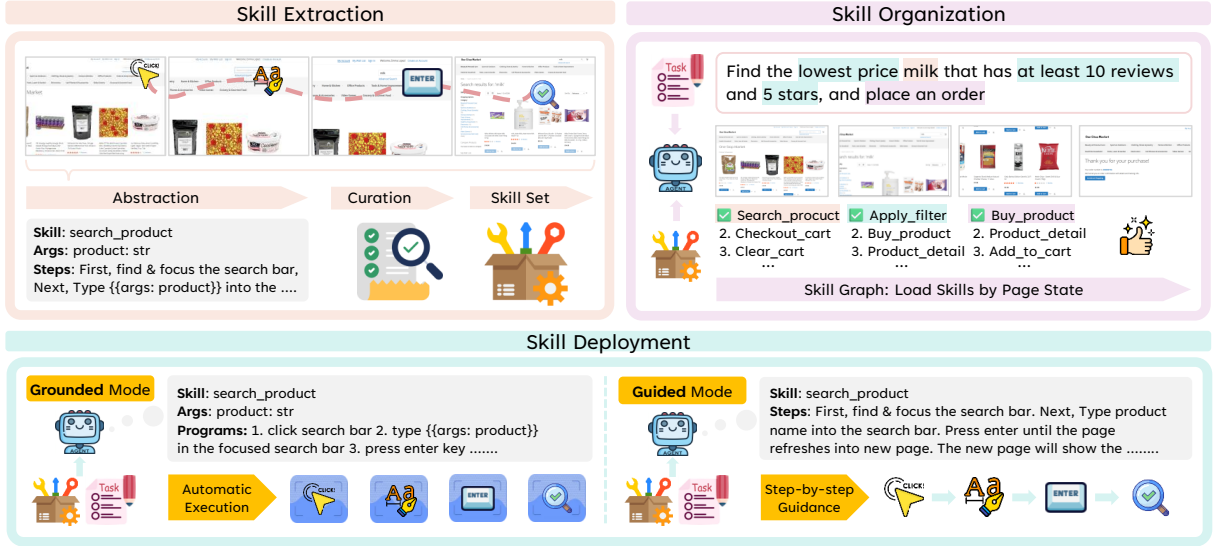


Figure 2: Overview of WEBXSKILL. (1) *Skill Extraction* abstracts low-level interaction trajectories into reusable skills, followed by curation and verification to improve quality. (2) *Skill Organization* structures skills into a URL-based graph and retrieves state-relevant candidates. (3) *Skill Deployment* supports *grounded mode* (invoke with auto-executed sequence for efficiency) and *guided mode* (surface step-by-step instructions for agent autonomy).

“search product” skill is only applicable on a page with a search bar, not on a checkout page.

**Skill Graph.** We organize skills into a graph  $\mathcal{G} = \{(u_j, \mathcal{S}_j)\}_{j=1}^M$ , where  $u_j$  is a generalized URL pattern (e.g., shopping/catalogsearch/\*) and  $\mathcal{S}_j$  contains the skills applicable to matching pages. URL routes are inexpensive signals of page identity and function, while variable path segments can be wildcarded. Skills sharing the same generalized URL are grouped into the same node. Using this graph will not block skill transfer across sites, since we can still use basic semantic embedding retrieval, as we studied in Section 4.2.

**Context-Aware Retrieval.** At inference time, the agent can efficiently match against graph nodes using the page URL: all matched nodes are retrieved and their associated skills are surfaced as candidates for current step. Since every skill is bound to specific browser elements, we further filter by checking element presence on the page, ensuring the surfaced skills are executable. For skill transfer across sites, we use embedding-based retrieval over skill descriptions and guidance to retrieve relevant skills for current task and page context.

### 3.3 Skill Deployment

As discussed in Section 2, existing executable-skill methods deploy skills as black-box calls where the agent has no visibility into the internal logic of the skill, making it unable to adapt when execution

fails unexpectedly. WEBXSKILL instead exposes both step-level guidance and code-based action sequences to the agent, enabling it to understand the skill logic and flexibly adapt to unexpected states or recover from execution errors. This motivates to decouple the “what” (the skill’s textual guidance and code program) from the “how” (the execution strategy): *grounded mode* for automated execution and *guided mode* for agent-driven execution.

**Grounded Mode.** In grounded mode, each skill is exposed as a callable tool in the agent’s action space:  $\mathcal{A} = \mathcal{A}_{\text{prim}} \cup \mathcal{A}_{\text{skill}}$ . The accompanying natural-language guidance remains visible to the agent as a planning aid. When the agent invokes a skill (e.g., `search_product(query="laptop")`), the runtime executes the corresponding action sequence by matching referenced elements against the current DOM and dispatching the low-level actions in order. This mode maximizes efficiency by compressing multi-step procedures into a single call, but it places greater demands on the agent’s reasoning and error recovery: execution can be interrupted by unexpected page changes, and the skill may not always align with the agent’s intent.

**Guided Mode.** For agents with weaker reasoning capabilities or skills must transfer across sites, guided mode surfaces skills as high-level guidance that the agent follows step-by-step using its own actions, so it can adapt when states differ from expectation. When activating a skill, the agent re-

ceives step-level textual and code guidance (e.g., “to search a product, first click the search input field, then type the query and press Enter”), while the automated action execution is disabled. If a step fails due to a changed page layout, the agent can recognize and re-plan toward the same sub-goal through alternative steps.

The two modes can be chosen per agent backbone and scenarios. Grounded mode is more efficient for stronger models that can reliably execute skills and recover from unexpected states; guided mode offers more robustness for weaker models that benefit from explicit procedural guidance.

## 4 Experiments

### 4.1 Experimental Setup

**Benchmarks.** We evaluate on three popular web agent benchmarks. WebArena (Zhou et al., 2024) provides five self-hosted websites; following recent work (Liu et al., 2025; Yang et al., 2026b), we use its cleaned 154 task subset. Though it is costly for web agent evaluation on live websites due to internet connectivity and reCAPTCHA issues, we also evaluate on two live-site benchmarks. For WebVoyager (He et al., 2024a), we retain 11 live websites and exclude Allrecipes, Booking, Google Flights, and Google Search because site changes or reCAPTCHA make their tasks invalid. Online-Mind2Web (Xue et al., 2025) adds 300 tasks across more than 100 live websites. We mainly use WebArena for controlled ablations, while the other two test live-site and cross-site skill transfer.

**Models & Baselines.** We focus on two strong multimodal LLMs: GPT-5 (Singh et al., 2025) and Qwen-3.5-122B-A10B-Int4 (Qwen Team, 2026) because skills are often used in the context of advanced models in the community. We also supplement results for Qwen3.6-35B-A3B-FP8 and Qwen3.5-9B to broaden the comparison though pilot experiments showed that sub-30B models could not reliably follow skill schemas. We compare against (1) *Vanilla*, a ReAct (Yao et al., 2023) agent without skills; (2) *MAP* (Multi-Action Prediction), which generates up to 3 browser actions per step; (3) *SkillWeaver* (Zheng et al., 2025); and (4) *WALT* (Prabhu et al., 2026). Note that skill acquisition is part of each method: SkillWeaver explores websites, WALT reverse-engineers site functions, and WEBXSKILL uses synthetic trajectories. To isolate skill organization and deployment effectiveness from this source difference, we also re-deploy

SkillWeaver and WALT skills in our framework (WEBXSKILL + *SkillWeaver/WALT*).

**Implementation.** All methods use a 30 interaction step budget. Skills are extracted by SynthAgent (Wang et al., 2026) on synthetic tasks for each benchmark website. At inference, we retrieve up to 20 candidate skills per page via skill graph. The agent framework is implemented with browser-use (browser-use, 2025). For benchmark evaluation, we follow the default settings by each benchmark and use GPT-4.1 if required for LLM judge. More details are in Appendix A.

### 4.2 Main Results

**WebArena.** Table 2 present results on WebArena. We first find that both deployment modes of WEBXSKILL outperform all baselines on the overall metric for both backbones, confirming that executable skills reduce planning errors and improve task completion. SkillWeaver and WALT in their original implementations are inferior to MAP. However, when their skills are re-deployed inside WEBXSKILL, performance recovers and even surpasses MAP. This isolates the contribution of the agent harness from the skill quality and ensures best-effort comparison. GPT-5 slightly favors grounded mode, whereas Qwen3.5-122B favors guided mode. Manual inspection confirms that Qwen is less reliable at recovering from execution faults, and guided mode helps by giving agent full autonomy to adapt. We further study the mode selection preference later within this section.

**WebVoyager and Online-Mind2Web.** Table 3 and Table 4 present results on WebVoyager and Online-Mind2Web respectively. For these two live websites benchmarks, we mainly compare against Vanilla and MAP methods, because SkillWeaver and WALT do not release skills for live websites and their acquisition pipelines rely on heavy exploration or website reverse-engineering that is not feasible for online services. Grounded mode reaches 86.1%, improving over Vanilla by 14.2 points and MAP by 11.7 on WebVoyager. For Online-Mind2Web, grounded mode is consistently the best method under all three judges. The guided mode also improves over MAP by 4.8 points.

**Skill Transfer across Websites.** Another advantage for WEBXSKILL is that guided mode allows skill transfer across sites, while prior code-based methods rely on fixed action scripts that are tightly

| Model | Method                           | Shop. | CMS  | Reddit | GitLab | Map  | Overall     |
|-------|----------------------------------|-------|------|--------|--------|------|-------------|
| GPT-5 | Vanilla                          | 50.0  | 65.7 | 79.0   | 63.3   | 50.0 | 59.7        |
|       | Vanilla + MAP                    | 52.3  | 80.0 | 84.2   | 60.0   | 46.2 | 63.0        |
|       | SkillWeaver (Zheng et al., 2025) | 28.3  | 42.9 | 30.2   | 44.4   | 41.3 | 37.4        |
|       | WALT (Prabhu et al., 2026)       | 30.9  | 47.4 | 57.7   | 51.8   | 28.1 | 42.9        |
|       | WEBXSKILL + SkillWeaver          | 56.8  | 57.1 | 68.4   | 66.7   | 53.9 | 59.7        |
|       | WEBXSKILL + WALT                 | 52.3  | 77.1 | 84.2   | 56.7   | 50.0 | 62.3        |
|       | WEBXSKILL (Grounded)             | 65.9  | 65.7 | 100.0  | 70.0   | 57.7 | <b>69.5</b> |
|       | WEBXSKILL (Guided)               | 59.1  | 80.0 | 89.5   | 63.3   | 61.5 | <u>68.8</u> |
| Qwen  | Vanilla                          | 40.9  | 48.6 | 57.9   | 43.3   | 42.3 | 45.5        |
|       | Vanilla + MAP                    | 38.6  | 47.1 | 73.7   | 43.3   | 46.2 | 47.1        |
|       | SkillWeaver (Zheng et al., 2025) | 37.8  | 38.7 | 66.7   | 40.0   | 46.2 | 43.8        |
|       | WALT (Prabhu et al., 2026)       | 31.8  | 45.7 | 57.9   | 50.0   | 46.2 | 44.2        |
|       | WEBXSKILL + SkillWeaver          | 36.4  | 42.9 | 63.2   | 53.3   | 30.8 | 43.5        |
|       | WEBXSKILL + WALT                 | 31.8  | 45.7 | 57.9   | 50.0   | 46.2 | 44.2        |
|       | WEBXSKILL (Grounded)             | 47.7  | 60.0 | 47.4   | 46.7   | 38.5 | <u>48.7</u> |
|       | WEBXSKILL (Guided)               | 47.7  | 57.1 | 63.2   | 50.0   | 57.7 | <b>53.9</b> |

Table 2: Results of GPT-5 and Qwen-3.5-122B on WebArena. WEBXSKILL + SkillWeaver/WALT deploys the respective method’s within our framework. Best and second-best overall results are **bolded** and underlined.

| Method                  | Amzn | Apple | ArXiv | BBC  | Camb. | Cour. | ESPN | GitHub | GMap | HF   | Wolf. | Overall     |
|-------------------------|------|-------|-------|------|-------|-------|------|--------|------|------|-------|-------------|
| Vanilla                 | 90.2 | 65.1  | 76.7  | 54.8 | 74.4  | 73.8  | 59.1 | 85.4   | 73.2 | 74.4 | 65.2  | 71.9        |
| Vanilla + MAP           | 90.2 | 72.1  | 81.4  | 35.7 | 81.4  | 69.1  | 56.8 | 92.7   | 80.5 | 81.4 | 78.3  | 74.4        |
| WEBXSKILL (Grounded)    | 95.1 | 74.4  | 86.1  | 81.0 | 90.7  | 83.3  | 79.6 | 95.1   | 90.2 | 83.7 | 89.1  | <b>86.1</b> |
| WEBXSKILL (Guided)      | 82.9 | 74.4  | 81.4  | 71.4 | 90.7  | 73.8  | 79.6 | 95.1   | 95.1 | 74.4 | 91.3  | 82.7        |
| WEBXSKILL (Guided + WA) | 97.6 | 79.1  | 76.7  | 71.4 | 90.7  | 78.6  | 93.2 | 87.8   | 92.7 | 83.7 | 84.8  | <u>85.1</u> |

Table 3: Results on WebVoyager with GPT-5. Guided + WA uses skills extracted from WebArena.

| Method      | Judge        |              |              |
|-------------|--------------|--------------|--------------|
|             | GPT-4.1      | GPT-5.1      | WebJudge     |
| Vanilla     | 59.00        | 53.33        | 61.33        |
| MAP         | 60.33        | 57.33        | 68.67        |
| Grounded    | <b>68.67</b> | <b>67.00</b> | <b>72.00</b> |
| Guided      | 67.33        | 63.33        | 70.00        |
| Guided + WA | 67.33        | 66.00        | 68.00        |

Table 4: Result on Online-Mind2Web with GPT-5 with three different judge models.

coupled with specific websites and cannot transfer. We test it with only WebArena skills (Guided + WA) and it reaches 85.1% without target-site programs, a 10.7 point gain over MAP on WebVoyager, while it has 8.1% relative improvement over MAP on Online-Mind2Web. The transfer works precisely because step-level guidance survives website change while fixed action scripts do not, directly addressing the concern that URL-based retrieval and grounded scripts cannot generalize across sites.

**Deployment Mode Preference.** A central value of WEBXSKILL is that a skill does not commit the agent to a single execution interface. The

| Backbone        | WebArena     |              | WebVoyager   |              |
|-----------------|--------------|--------------|--------------|--------------|
|                 | Grounded     | Guided       | Grounded     | Guided       |
| Qwen3.6-35B-A3B | <b>59.09</b> | 57.79        | <b>89.98</b> | 85.93        |
| Qwen3.5-9B      | 44.16        | <b>49.35</b> | 71.86        | <b>84.86</b> |

Table 5: Skill deployment mode study with broader open-source models.

same learned procedure can be invoked as an executable tool or read as a step-by-step plan, allowing each backbone to use procedural knowledge in the form that best matches its capabilities. Table 2 shows that GPT-5 favors grounded mode, whereas Qwen3.5-122B performs better with guided mode. The comparison in Table 5 extends this pattern: Qwen3.6-35B benefits from grounded execution, while Qwen3.5-9B benefits from guided use. These results point to a general design principle for agent skills: reusable knowledge and execution control should be decoupled. Models that can carry out a complete procedure as a tool gain from grounded execution; models that benefit from reasoning over intermediate steps can use the same skill as guidance while retaining control of their actions.

| Method               | Avg. Steps ↓ |      |      |      |      |            | Skill Usage % |             |       |      |       |      |
|----------------------|--------------|------|------|------|------|------------|---------------|-------------|-------|------|-------|------|
|                      | Shop.        | CMS  | Red. | Git. | Map  | All        | All           |             | Succ. |      | Fail. |      |
|                      |              |      |      |      |      |            | IR            | UR          | IR    | UR   | IR    | UR   |
| Vanilla              | 7.8          | 11.9 | 7.5  | 11.6 | 13.3 | 10.4       | –             | –           | –     | –    | –     | –    |
| Vanilla + MAP        | 6.1          | 11.1 | 13.1 | 11.8 | 8.4  | 9.8        | –             | –           | –     | –    | –     | –    |
| SkillWeaver          | 6.8          | 10.2 | 7.4  | 11.7 | 11.4 | 9.4        | 6.6           | 37.7        | 7.5   | 39.1 | 5.5   | 35.5 |
| WALT                 | 6.5          | 11.0 | 7.3  | 11.5 | 10.0 | <b>9.2</b> | 9.3           | 33.1        | 9.3   | 32.3 | 9.3   | 34.5 |
| WEBXSKILL (Grounded) | 6.1          | 10.5 | 6.2  | 13.0 | 11.1 | <u>9.3</u> | <b>16.5</b>   | <b>70.8</b> | 19.5  | 73.8 | 12.4  | 63.8 |
| WEBXSKILL (Guided)   | 7.5          | 11.5 | 8.6  | 14.5 | 11.1 | 10.5       | <u>10.1</u>   | <u>68.8</u> | 11.4  | 68.9 | 8.3   | 68.8 |

Table 6: Efficiency and skill usage analysis on WebArena with GPT-5. IR (Invocation Rate): percentage of agent steps that invoke a skill. UR (Usage Rate): percentage of tasks where at least one skill is invoked. Succ. and Fail. denote successful and failed tasks, respectively.

| Method                 | #Sk.  | #Op. | Sk./Nd. | SR   | Util. |
|------------------------|-------|------|---------|------|-------|
| SkillWeaver            | 87.6  | 1.6  | –       | 84.1 | 8.2   |
| WALT                   | 8.2   | 4.5  | –       | 67.2 | 22.0  |
| WEBXSKILL              | 118.2 | 3.8  | 4.2     | 77.1 | 12.9  |
| WEBXSKILL <sup>†</sup> | 67.6  | 3.5  | 3.5     | 85.0 | 27.8  |

Table 7: Skill quality and coverage. <sup>†</sup>: skills from test-set trajectories. #Sk.: skills/site; #Op.: write actions/skill; Sk./Nd.: skills/page node; SR: execution success (%); Util.: skills invoked (%).

| Corpus Size | # Skills | TSR         | UR          |
|-------------|----------|-------------|-------------|
| 25%         | 105      | 62.8        | 26.6        |
| 50%         | 222      | 66.2        | 39.0        |
| 75%         | 474      | 68.8        | 51.9        |
| 100%        | 591      | <b>69.5</b> | <b>70.8</b> |

Table 8: Skill scaling on WebArena by varying the corpus size for skill extraction. TSR: task success rate (%); UR: tasks invoking at least one skill (%).

**Efficiency & Skill Usage.** Table 6 studies step efficiency and skill usage. WEBXSKILL achieves substantially higher skill adoption than SkillWeaver and WALT: grounded mode reaches Usage Rate (UR) 70.8% and Invocation Rate (IR) 16.5%, roughly doubling WALT and SkillWeaver. This higher adoption reflects not only the usefulness of our extracted skills but also the effectiveness of our skill organization. Splitting by outcome, grounded mode shows notably higher IR and UR on successful tasks, suggesting that effective skill invocation contributes to task completion. In contrast, Guided mode shows nearly identical UR across success and failure (68.9% vs. 68.8%), indicating that guided skills are uniformly available but success depends on the agent’s own execution quality rather than skill availability. For step efficiency, Grounded mode achieves 9.3 average steps with higher success rate, fewer steps than Vanilla. Guided mode uses slightly more steps as the agent needs to take action by action, trading efficiency for adaptability.

## 5 Analysis

### 5.1 Skill Quality and Coverage

Table 7 and Figure 3 compare skill libraries across methods. WEBXSKILL builds the largest library

(100+ skills per site) from low-cost synthetic trajectories, spread across all ten functional categories with no single category exceeding 24%. In contrast, SkillWeaver concentrates over 60% in retrieval (data-extraction routines), and WALT provides only 41 skills in total, resulting in sparse coverage across most categories. The category distribution of our synthetic skills closely matches the test-set oracle variant, indicating that our extraction captures the natural distribution of reusable skills without access to evaluation data. Our skills are also more procedural, averaging 3.8 operations per skill compared to SkillWeaver’s 1.6, since many of SkillWeaver’s skills are value-extraction routines that only read page content. Beyond library size, our skill graph keeps each page node compact (4.2 skills per node), which helps reduce retrieval noise. Utilization (Util.) highlights the difference between breadth and effective coverage: although WEBXSKILL has 10× more skills than WALT, its 12.9% utilization corresponds to ~15 unique skills invoked per site, while WALT’s 22.0% maps to only about 1.8 unique skills per site, far short of the procedural coverage long-horizon tasks demand, consistent with the results in Table 6. For execution reliability, the execution success rate (SR) of our curated and verified skills reaches 77%, close to the 85% achieved by the test-set oracle variant.

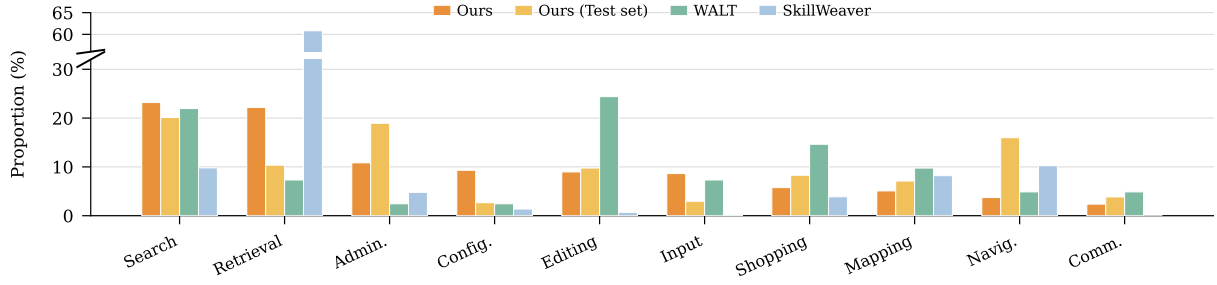


Figure 3: Skill category distribution. WEBXSKILL covers all ten categories; SkillWeaver concentrates over 60% in Retrieval, and WALT has 41 skills in total.

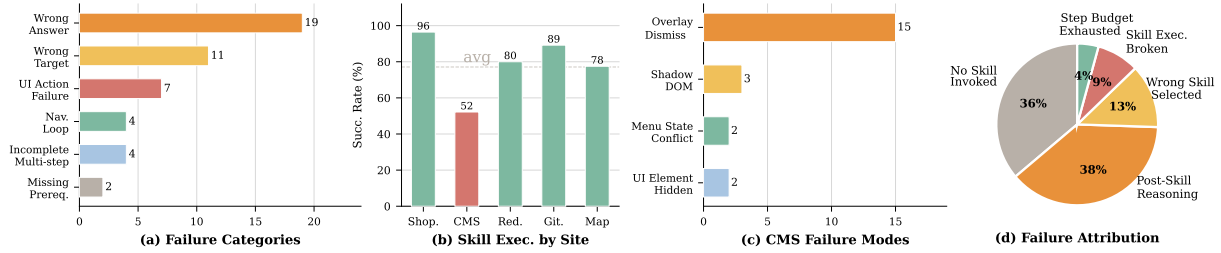


Figure 4: Failure analysis of WEBXSKILL (grounded mode) on WebArena. (a) Failure categories for failed tasks from trajectory inspection. (b) Per-site skill execution success rate. (c) Root causes of CMS skill execution failures. (d) Failure attribution by skill usage role.

| Corpus     | Organization | # Skills | TSR         |
|------------|--------------|----------|-------------|
| Full       | Skill Graph  | 591      | <b>69.5</b> |
| +Filtering | Skill Graph  | 548      | 68.8        |
| Full       | DOM matching | 591      | 65.6        |
| Full       | Flat Library | 591      | 59.1        |

Table 9: Study on corpus filtering and skill retrieval on WebArena with GPT-5. Filtered corpus removes sources similar to test tasks. DOM matching retrieves skills based on DOM structure similarity, while Flat library searches the full library.

## 5.2 Skill Scaling and Retrieval

Table 8 traces how the size of the 2,500-trajectory corpus changes the verified skill library and its use. Expanding from 25% of the corpus to the full set grows the library from 105 to 591 skills, raises task success from 62.8% to 69.5%, and increases UR from 26.6% to 70.8%. All three measures rise monotonically, but coverage continues to grow most strongly in the long tail: the final 25% of sources adds 117 skills and raises UR by 18.9 points. These later skills support increasingly specialized workflows, broadening the set of tasks for which the agent can retrieve a relevant procedure.

Table 9 studies the generality of the extracted procedures with a similarity-filtered corpus. We retrieve the closest WebArena test task for each

synthetic source, classify workflow and goal similarity with GPT-5, and remove 196 near-matching sources before rebuilding the library. The filtered corpus retains 548 verified skills and reaches 68.8% task success, close to the full corpus at 69.5%. The extracted procedures therefore remain useful across varied task instructions that share reusable browser operations. We also hold the full skill library fixed and validate how skill candidates are selected. A flat-library search reaches 59.1%, DOM matching reaches 65.6%, and the proposed URL-based skill graph reaches 69.5%. DOM matching gains 6.5 points by conditioning retrieval on the current page structure; the graph adds another 3.9 points by first narrowing candidates to the page’s functional context and then filtering them against visible elements. This layered retrieval keeps the candidate set compact while preserving the coverage of full library.

## 5.3 Ablation Study

Table 10 isolates each component, and the contributions align with the four axes of the grounding gap (Table 1). We have the following observations: (1) Broad experience produces strong procedural coverage. The synthetic library reaches 69.5 task success, comparable to 68.2 from an oracle library built with test-set trajectories, showing that diverse synthetic experience can recover the workflows

| Variant                | Task Success Rate (%) |             |            |             |             |             |            | Skill Usage (%) |             |             |             |             |
|------------------------|-----------------------|-------------|------------|-------------|-------------|-------------|------------|-----------------|-------------|-------------|-------------|-------------|
|                        | Shop.                 | CMS         | Red.       | Git.        | Map         | All         | Steps      | All             |             | Succ.       |             | Fail.       |
|                        |                       |             |            |             |             |             |            | IR              | UR          | IR          | UR          | UR          |
| WEBXSKILL (Grounded)   | <b>65.9</b>           | 65.7        | <b>100</b> | 70.0        | 57.7        | <b>69.5</b> | 9.3        | 16.5            | 70.8        | 19.5        | 73.8        | 63.8        |
| w/ Test-set Skills     | 59.1                  | <b>71.4</b> | 84.2       | <b>86.7</b> | 46.2        | 68.2        | <b>8.1</b> | <b>28.2</b>     | <b>86.4</b> | <b>29.6</b> | <b>83.8</b> | <b>91.8</b> |
| w/ Mix Mode            | 59.1                  | 68.6        | 84.2       | 70.0        | <b>57.7</b> | 66.2        | 8.6        | 17.9            | 70.1        | 19.3        | 72.5        | 65.4        |
| w/o Skill Verification | 40.9                  | 60.0        | 78.9       | 60.0        | 50.0        | 55.2        | 10.4       | 21.2            | 74.0        | 19.1        | 72.9        | 75.4        |
| w/o Skill Graph        | 52.3                  | 51.4        | 94.7       | 56.7        | 57.7        | 59.1        | 10.9       | 10.1            | 56.5        | 11.0        | 53.8        | 60.3        |
| w/o Step Guidance      | 52.3                  | 62.9        | 84.2       | 63.3        | 50.0        | 60.4        | 10.0       | 17.6            | 73.4        | 19.1        | 75.3        | 70.5        |

Table 10: Ablations on WebArena with GPT-5. IR: percentage of steps invoking a skill; UR: percentage of tasks invoking at least one skill.

| Backbone     | Mode     | TSR  | UR   | Gr. choices |
|--------------|----------|------|------|-------------|
| GPT-5        | Grounded | 69.5 | 70.8 | 100.0       |
|              | Guided   | 68.8 | 68.8 | 0.0         |
|              | Mix      | 66.2 | 70.1 | 97.1        |
| Qwen3.5-122B | Grounded | 48.7 | 48.4 | 100.0       |
|              | Guided   | 53.9 | 21.4 | 0.0         |
|              | Mix      | 42.2 | 40.3 | 96.2        |

Table 11: Mix-mode behavior on WebArena. TSR/UR and grounded (Gr.) choices are percentages.

needed at evaluation time. (2) Skill value comes from the whole lifecycle, not extraction alone. Verification makes the largest contribution, while contextual organization and step-level guidance each provide substantial gains. A useful skill must be reliable in execution, surfaced in the right context, and understandable to the agent when adaptation is needed. (3) The framework improves skills beyond those it learns itself. Organizing and deploying SkillWeaver and WALT skills with WEBXSKILL raises success by 22.3 and 19.4 points, respectively, showing that our organization and deployment provide a reusable interface for existing skill collections. (4) Skill deployment should match how a model uses procedural knowledge. Each model exhibits a different preference between grounded and guided use, while mix mode largely converges to grounded execution. The useful flexibility is therefore at the model level: the same skill can be exposed as an executable tool or a readable plan according to the model’s capabilities.

#### 5.4 Failure Analysis

We manually inspect all failed trajectories in Figure 4 and find: (1) Failures fall into six categories, with wrong answer extraction as the dominant mode: the agent completes the workflow correctly but reports an incorrect final answer. (2) Execution reliability varies sharply by site, from 96% on Shop-

ping down to 52% on CMS, the most structurally complex environment in WebArena. (3) The dominant CMS execution failure, “overlay dismiss”, is usually benign: these skills end with a click on a non-interactive element intended to dismiss a navigation overlay. The preceding steps succeed and 80% of affected tasks still complete despite the trailing error, so strict per-step success can understate end-task utility. (4) Failure attribution shows that post-skill reasoning (38%) is the largest failure category: skills executed successfully but the agent failed at subsequent steps. No skill invoked (36%) means the agent bypassed available skills entirely and failed using only native browser actions. Overall, most failures originate from agent-level decision making rather than skill design flaws, suggesting that further gains may be obtained by improving the reasoning and planning of LLM and the context management of agent harness.

## 6 Conclusion

We introduce WEBXSKILL, a skill-learning framework for autonomous web agents that bridges the grounding gap in existing skill formulations. By making action programs with step-level natural language guidance visible and readable to the agent, WEBXSKILL enables both efficient grounded execution and adaptable guided use, while skill extraction from synthetic trajectories and organization in a URL-based graph ensure a rich and retrievable skill library. Across a series of benchmark tasks, WEBXSKILL consistently outperforms strong baselines. Extensive analysis reveals a model-dependent preference for deployment mode: stronger models benefit from grounded execution, while weaker models benefit from guided use. Overall, WEBXSKILL offers a practical approach for equipping web agents with reusable procedural knowledge through readable and executable skills.

## Limitations

While WEBXSKILL advances skill learning for web agents, it has several limitations that may be further improved in future work:

- Our skill extraction relies on a corpus of synthetic agent trajectories, while this is viewed as a strength for its scalability and avoidance of test-data leakage, it also inherits the blind spots of the upstream synthesis. Sites whose interactions are poorly covered by synthetic trajectories will yield a sparser skill library.
- The URL-based skill graph is primarily designed for in-site retrieval, which may raise concerns about limiting cross-site transfer. However, this limitation stems from the nature of code-based web skills rather than the retrieval mechanism itself: such skills, including ours, are tightly coupled to site-specific DOM structures and are therefore inherently difficult to transfer across sites. We introduce guided mode to partially relax this constraint by exposing skills as procedural guidance rather than fixed scripts.

## Use of AI Assistants

We used coding agents (e.g., Claude Code) to assist with code implementation and debugging. And we also used AI assistants (e.g., ChatGPT) to help polish the writing of the paper. The authors have reviewed and edited the content as needed.

## References

- Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*.
- browser-use. 2025. [browser-use: Make websites accessible for ai agents. automate tasks online with ease.](#) GitHub repository, accessed 2025-10-10.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. [Mind2web: Towards a generalist agent for the web.](#) *Preprint*, arXiv:2306.06070.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. [Pal: Program-aided language models.](#) *Preprint*, arXiv:2211.10435.
- Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024a. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890.
- Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. 2024b. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. *arXiv preprint arXiv:2410.19609*.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*.
- Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, and 1 others. 2025. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*.
- Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, and 1 others. 2025. Scalecua: Scaling open-source computer use agents with cross-platform data. *arXiv preprint arXiv:2509.15221*.
- Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanqing Xiong, and Hongsheng Li. 2025. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*.
- Shikhar Murty, Hao Zhu, Dzmitry Bahdanau, and Christopher D Manning. 2024. Nnetnav: Unsupervised learning of browser agents through environment interaction in the wild. *arXiv preprint arXiv:2410.02907*.
- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.
- Liangbo Ning, Ziran Liang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Wenqi Fan, Xiaoyong Wei, Shanru Lin, Hui Liu, Philip S. Yu, and Qing Li. 2025. [A survey of webagents: Towards next-generation ai agents for web automation with large foundation models.](#) *Preprint*, arXiv:2503.23350.
- Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Awadallah. 2025. [Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents.](#) *Preprint*, arXiv:2502.11357.

- Viraj Prabhu, Yutong Dai, Matthew Fernandez, Krithika Ramakrishnan, Jing Gu, Yanqi Luo, Silvio Savarese, Caiming Xiong, Junnan Li, Zeyuan Chen, and Ran Xu. 2026. [WALT: Web agents that learn tools](#). In *The Fourteenth International Conference on Learning Representations*.
- Qwen Team. 2026. [Qwen3.5: Towards native multi-modal agents](#).
- Gabriel Sarch, Lawrence Jang, Michael J Tarr, William W Cohen, Kenneth Marino, and Katerina Fragkiadaki. 2024. Vlm agents generate their own memories: Distilling experience into embodied programs of thought. *Advances in Neural Information Processing Systems*, 37:75942–75985.
- Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 32 others. 2025. [Openai gpt-5 system card](#). *Preprint*, arXiv:2601.03267.
- Paloma Sodhi, SRK Branavan, Yoav Artzi, and Ryan McDonald. 2023. Step: Stacked llm policies for web actions. *arXiv preprint arXiv:2310.03720*.
- Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, and 1 others. 2025. Coact-1: Computer-using agents with coding as actions. *arXiv preprint arXiv:2508.03923*.
- Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. 2025. [Os-genesis: Automating gui agent trajectory construction via reverse task synthesis](#). *Preprint*, arXiv:2412.19723.
- Zhaoyang Wang, Yiming Liang, Xuchao Zhang, Qianhui Wu, Siwei Han, Anson Bastos, Rujia Wang, Chetan Bansal, Baolin Peng, Jianfeng Gao, Saravan Rajmohan, and Huaxiu Yao. 2026. [SynthAgent: Adapting web agents with synthetic supervision](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15730–15752, San Diego, California, United States. Association for Computational Linguistics.
- Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025. [Inducing programmatic skills for agentic tasks](#). In *Second Conference on Language Modeling*.
- Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.
- Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, and 1 others. 2025. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. *arXiv preprint arXiv:2505.16421*.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.
- Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. [An illusion of progress? assessing the current state of web agents](#). In *Second Conference on Language Modeling*.
- Ke Yang, Zixi Chen, Xuan He, Jize Jiang, Michel Galley, Chenglong Wang, Jianfeng Gao, Jiawei Han, and ChengXiang Zhai. 2026a. [Plugmem: A task-agnostic plugin memory module for llm agents](#). *Preprint*, arXiv:2603.03296.
- Rui Yang, Qianhui Wu, Zhaoyang Wang, Hanyang Chen, Ke Yang, Hao Cheng, Huaxiu Yao, Baolin Peng, Huan Zhang, Jianfeng Gao, and 1 others. 2026b. [Gui-libra: Training native gui agents to reason and act with action-aware supervision and partially verifiable rl](#). *arXiv preprint arXiv:2602.22190*.
- Yuhao Yang, Zhen Yang, Zi-Yi Dou, Anh Nguyen, Keen You, Omar Attia, Andrew Szot, Michael Feng, Ram Ramrakhya, Alexander Toshev, and 1 others. 2025. [Ultracua: A foundation model for computer use agents with hybrid action](#). *arXiv preprint arXiv:2510.17790*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [React: Synergizing reasoning and acting in language models](#). *Preprint*, arXiv:2210.03629.
- Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. 2025. [Skillweaver: Web agents can self-improve by discovering and honing skills](#). *Preprint*, arXiv:2504.07079.
- Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. [Webarena: A realistic web environment for building autonomous agents](#). *Preprint*, arXiv:2307.13854.
- Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Li Erran Li. 2025. Proposer-agent-evaluator (pae): Autonomous skill discovery for foundation model internet agents. In *Forty-second International Conference on Machine Learning*.

## A Implementation Details

### A.1 Synthetic Trajectory Collection

Our skills are extracted from trajectories collected by SynthAgent (Wang et al., 2026), which synthesizes diverse web tasks through categorized exploration of target websites. We use 2,500 and 600 synthetic tasks for WebArena (Zhou et al., 2024) and WebVoyager (He et al., 2024a), respectively, covering a wide range of user intents and website functionalities. Our extraction pipeline is designed to mine reusable action subsequences from both successful and failed trajectories, since a failed trajectory may still contain useful skills to finish a subtask before the failure point. For Online-Mind2Web (Xue et al., 2025), we generate independent tasks from site names and home URLs with SynthAgent. Benchmark instructions, labels, outcomes, and execution trajectories are not supplied to task synthesis or skill extraction.

### A.2 Skill Extraction Details

**Trajectory Formatting.** Each trajectory is converted to a structured textual representation before being sent to the LLM. For each step, we record the page URL, the agent’s reasoning (thinking and next goal), and each action with its type, target element description (tag name, visible text, and key HTML attributes such as id, name, aria-label), and action parameters. The same extraction prompt and settings are used for every benchmark. The output must follow the JSON schema in Table 13; each candidate contains 3–6 meaningful actions and guidance for every action.

**Online Deduplication.** When processing the trajectory corpus sequentially, each newly proposed skill is compared against the existing library using a three-level cascade: (1) exact name deduplication to filter identical skill names; (2) same-site near-duplicate filtering based on Jaccard similarity between tokens in skill function names; (3) library-wide matching using a similarity score with OpenAI’s text-embedding-3-small model. For each candidate, the top- $k$  ( $k=20$ ) most similar existing skills are included in the extraction prompt to help the LLM decide whether to add, update, or skip. The full extraction prompt is shown in Tables 12 and 13.

### A.3 Skill Format

As shown in Table 14, each executable skill is represented as a JSON object containing a semantic sig-

#### Skill Extraction Prompt (1/2): Task & Deduplication

```
You are a web automation expert. Analyze the trajectory
and
extract reusable skill abstractions.

## Minimum Steps
Each skill MUST have at least {step_threshold} meaningful
action
steps (click, input, send_keys, navigate, etc.).

## Duplicate & Overlap Detection (Quality Over Quantity)
Before creating any new skill, carefully check the
existing
library below. Each skill has a similarity_score.
You MUST "skip" or "update" (NOT "new") if ANY of these:
1. Same user goal: An existing skill achieves the same
outcome.
2. Large semantic overlap: >=70% overlapping action steps.
3. Differ only in final step(s).
[...]

## Existing Skill Library
{existing_skills_section}

## Trajectory
{trajectory_text}

## Your Task
1. Identify reusable action sequences ({step_threshold}-6
actions) that could be abstracted into skills.
2. Skills should be generic (parameterized), atomic (
single
logical operation), and reusable across different tasks.

3. Check the existing library first. For each candidate
skill:
- similarity_score > 0.4 -> very likely overlaps
- Same user goal -> "skip"
- Better version in trajectory -> "update"
- No similar existing skill -> "new"

## meta_url Rules
- start_url: The EXACT URL from the trajectory
- meta_url: A generalized URL pattern
- Use * for variable parts: "gitlab/*/-/issues/*"
```

Table 12: Skill extraction prompt (1/2): task description, deduplication rules, and skill abstraction instructions. The prompt receives the formatted trajectory and existing skill library with similarity scores.

nature (name, description, typed parameters) and a sequence of action steps. Every action step includes a guidance field providing natural language guidance, an action\_type, an element\_ref describing the target UI element via tag name and HTML attributes, and step-specific params. Parameters use the {{param\_name}} syntax for value abstraction.

### A.4 Skill Graph and Retrieval

The skill graph uses generalized URL patterns (meta\_url) as node identifiers, where variable path segments are replaced with wildcards (e.g., gitlab/\*/-/issues/\*). At inference time, URL matching follows a priority-based aggregation: (1) exact match on the normalized URL (high-

### Skill Extraction Prompt (2/2): Output Format

```
## Output Format
{
  "extractions": [
    {
      "action": "new",
      "skill": {
        "name": "skill_name_in_snake_case",
        "description": "What this skill does",
        "start_url": "exact URL from trajectory",
        "meta_url": "generalized URL pattern",
        "parameters": [
          { "name": "p", "type": "str",
            "description": "...", "required": true }
        ],
        "action_steps": [
          {
            "guidance": "Why this action is needed",
            "action_type": "click|input|scroll|...",
            "element_ref": {
              "tag_name": "button|input|a|div|...",
              "text_content": "visible text",
              "attributes": { "id": "...", "class": "..." }
            },
            "params": { "text": "{{param_name}}" }
          }
        ]
      }
    },
    { "action": "skip", "existing_id": "skill_42",
      "reason": "Same as search_product" },
    { "action": "update", "existing_id": "skill_42",
      "reason": "Shorter: 3 steps vs 4",
      "skill": { ... complete updated skill ... } }
  ]
}

## Action Types Reference
1. click - element_ref: REQUIRED
2. input - element_ref: REQUIRED, params: { "text", "clear" }
3. select_dropdown - element_ref: REQUIRED, params: { "text" }
4. scroll - element_ref: null, params: { "direction", "pages" }
5. send_keys - element_ref: null, params: { "keys" }
6. navigate - element_ref: null, params: { "url", "new_tab" }
[...]

## Important Rules
- guidance is REQUIRED for every action step
- Use {{param_name}} syntax for parameterized values
- Even failed trajectories may contain useful action sequences
```

Table 13: Skill extraction prompt (2/2): output JSON format, action types reference, and important rules. The LLM outputs extraction decisions (new, skip, or update) with complete skill definitions.

est priority); (2) wildcard pattern matching via fnmatch on all graph nodes, ranked by specificity (number of non-wildcard path segments). Skills from all matching nodes are merged, with more specific matches taking priority and duplicates (by skill name) removed. Up to 20 skills are surfaced per page. During runtime, we also use element-based heuristics to filter skills whose target elements are not present on the current page, improving relevance and reducing noise for the agent.

### Example of an Extracted Executable Skill (JSON)

```
{
  "name": "search_products_from_homepage",
  "description": "From the shopping homepage, perform a product search using the main search box and submit it.",
  "meta_url": "shopping",
  "parameters": [
    { "name": "query", "type": "str",
      "description": "Product search query to enter.",
      "required": true }
  ],
  "action_steps": [
    {
      "guidance": "Focus the main site search input.",
      "action_type": "click",
      "element_ref": {
        "tag_name": "input",
        "attributes": { "id": "search", "name": "q",
          "placeholder": "Search entire store here..." }
      },
      "params": {}
    },
    {
      "guidance": "Type the search query, clearing existing text.",
      "action_type": "input",
      "element_ref": {
        "tag_name": "input",
        "attributes": { "id": "search", "name": "q" }
      },
      "params": { "text": "{{query}}", "clear": true }
    },
    {
      "guidance": "Submit the search by clicking the Search button.",
      "action_type": "click",
      "element_ref": {
        "tag_name": "button", "text_content": "Search",
        "attributes": { "type": "submit", "aria-label": "Search" }
      },
      "params": {}
    }
  ]
}
```

Table 14: Example of an extracted executable skill in JSON format. Each skill contains a semantic signature (name, description, typed parameters) and a sequence of action steps with element references and natural language step-level guidance.

## A.5 Skill Deployment Prompts

In grounded mode (§3.3), skills are registered as callable tools (prefixed with fg\_ to differ from native browser actions like click) in the agent’s action space. The agent system prompt includes action rules explaining the availability and recommended usage of these pre-built skills (Table 15). Each registered tool carries a description derived from the skill’s metadata, including action step guidance and parameter specifications (Table 16).

In guided mode, skills are surfaced as step-by-step workflow instructions rather than auto-executed tools. The agent system prompt includes a skill mode section explaining how to activate

### Grounded Mode: Action Rules

```
<action_rules>
- In some pages, you are equipped with pre-built skills ( fg_*).
  Each skill is a sequence of low-level actions (click,
  input,
  scroll, wait, etc.) that will be sequentially executed
  when
  you call the skill.
- Prefer fg_* skills when they match your current goal
  because
  they are efficient and well-tested.
- If an fg_* skill fails or does not meet your
  expectations,
  do not repeat the same failing fg_* call; continue using
  native browser actions to achieve your goal.
</action_rules>

<reasoning_rules>
[...]
- If current page provides pre-built skills (fg_*), you
  need
  to provide reasoning about whether to use them or not.
</reasoning_rules>
```

Table 15: Grounded mode action rules appended to the agent system prompt, instructing the agent on how to use pre-built skill tools (fg\_\*).

### Grounded Mode: Example Registered Skill Tool

```
Tool name: fg_search_products_from_homepage
Description: From the shopping homepage, perform a product
search using the main search box and submit it.

Action Steps:
Step 1: Focus the main site search input. ->
  click <input id="search" placeholder="Search entire
  store here...">
Step 2: Type the search query into the main search field.

  -> input <input id="search"> with text="{{query}}"
Step 3: Submit the search by clicking the Search button.
  -> click <button type="submit" aria-label="Search">

Parameters (pass as kwargs_json):
- query: str (required) - Product search query to enter.
```

Table 16: Example of a registered skill tool in grounded mode. The tool description includes the skill’s action step guidance and parameter specifications.

and follow skills (Table 17). At runtime, available skills are listed via an `<available_skills>` block. Upon activation, step-level guidance is injected via `<activated_skill_guidance>` (Table 18).

## A.6 Web Agent Evaluation

Evaluating web agents can be challenging due to the time-consuming nature of browser actions and costly live site interactions. Across the web agent literature (Prabhu et al., 2026; Zheng et al., 2025; Wang et al., 2025), it is common practice to evaluate on a single benchmark such as WebArena (Zhou et al., 2024) because it provides high-quality verification and reproducible self-hosted environments,

### Guided Mode: System Prompt

```
## SKILL MODE

You have access to a library of Skills -- pre-defined
step-by-step workflow guides for common tasks on web pages.

### How Skills Work:
- Each step, you'll see <available_skills> listing skills
- Call use_skill(skill_name="...") to activate a skill
- On your NEXT step, <activated_skill_guidance> will
  appear
  with detailed step-by-step instructions
- Follow the guidance using your normal browser actions
- Skills are guides, not automatic executors -- YOU
  perform
  each step
- Call clear_skill() when finished

### When to Use Skills:
- When an available skill matches your current sub-goal
- Skills save you from figuring out the UI workflow

### When NOT to Use Skills:
- When no skill matches your goal
- When the task is simple enough

### After Activating a Skill:
1. Read the step-by-step guidance carefully
2. Observe the page to identify the elements described
3. Execute each step using your browser actions
4. If a step fails, adapt and continue
5. Call clear_skill() when done
6. IMPORTANT: skill is a general guide, not a strict
  script;
  use your judgment to adapt as needed.
```

Table 17: Guided mode system prompt appended to the agent, explaining how to activate and follow skill guidance using native browser actions.

allowing for more reliable attribution of performance differences to the agent design rather than environmental noise. To further test the generalization of WEBXSKILL, we also evaluate on WebVoyager (He et al., 2024a), which includes 10+ live websites. We mitigate reliability concerns by manually inspecting the trajectories to exclude unstable websites due to reCAPTCHA or task inaccessibility issues. For Online-Mind2Web, we use GPT-4.1, GPT-5.1 and WebJudge-7B to evaluate the 300 tasks with the benchmark’s key-point extraction, screenshot selection, and final-decision pipeline with a threshold of 3 for task success.

## B Case Study

We present two examples from WebArena using GPT-5 that illustrate how WEBXSKILL skills assist the agent in both grounded and guided modes. For each case, we show the task description, selected trajectory steps with browser screenshots, and the agent’s actions. Steps where a skill is invoked or activated are highlighted with a green background.

| Guided Mode: Runtime Skill Injection                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> --- Injected into agent's input message each step ---  &lt;available_skills&gt; Skills are pre-defined step-by-step workflow guides. Call use_skill(skill_name="...") to activate a skill. Call clear_skill() when finished.  1. search_products_from_homepage: Perform a product search    using the main search box. 2. navigate_to_category: Navigate to a specific product    category from the navigation menu. 3. sort_search_results: Sort search results by a criterion.  [...] &lt;/available_skills&gt;  --- After calling use_skill("search_products...") ---  &lt;activated_skill_guidance&gt; Skill: "search_products_from_homepage" Description: Perform a product search using the main search box and submit it with the Search button.  Follow these steps using your browser actions:  Step 1: Focus the main site search input. Example: click on &lt;input&gt; placeholder="Search entire store here..." id="search" Step 2: Type the search query, clearing existing text. Example: input "{{query}}" into &lt;input&gt; id="search" Step 3: Submit the search by clicking the Search button. Example: click on &lt;button&gt; text="Search" type="submit"  IMPORTANT: - Use native browser actions to execute each step - If a step fails, adapt and continue - Call clear_skill() when finished &lt;/activated_skill_guidance&gt; </pre> |

Table 18: Guided mode runtime injection. Available skills are listed via <available\_skills>; upon activation, step-by-step guidance with action examples is injected via <activated\_skill\_guidance>.

### B.1 Grounded Mode

Figure 5 shows the agent completing a Reddit forum posting task in only 3 steps using grounded mode. The task requires creating a post about a Harry Potter virtual meetup in the books subreddit. In Step 0, the agent identifies the available skill `submit_text_post_with_forum_selection` and invokes it with the post title, body, and target forum as parameters. The first attempt fails due to a JSON formatting error. In Step 1, the agent retries the same skill with corrected arguments, and the runtime automatically executes the entire multi-step workflow: navigating to the submission form, selecting the “books” forum, filling in the title and body, and submitting. In Step 2, the agent confirms the post is visible and calls done. Without the skill, this task would require the agent to independently navigate to the correct forum,



Figure 5: Grounded mode case study: the agent completes a Reddit forum posting task in 3 steps by invoking a skill that auto-executes the multi-step post submission workflow.

find the submission form, fill in multiple fields, and submit, a sequence that typically takes 8–10 steps.

### B.2 Guided Mode

Figure 6 shows the agent modifying an order’s shipping address in the CMS admin panel using guided mode (earlier navigation steps via the `open_orders_grid_from_any_admin_page` skill are omitted for brevity). In Step 6, the agent activates the skill `edit_shipping_address_for_order`, which provides 5-step guidance for editing the address form. Following the guidance, the agent fills in the street, city, and zip code fields (Step 8) and saves the form (Step 9). In the omitted final steps, the agent discovers that the State/Province field was not updated because the skill guidance did not cover dropdown fields. The agent independently re-opens the edit form and selects “New York” from the State/Province dropdown, demonstrating skill adaptation: the ability to go beyond the skill’s

