Inducing Programmatic Skills for Agentic Tasks

Zora Zhiruo Wang **Apurva Gandhi** **Graham Neubig** **Daniel Fried**
Carnegie Mellon University
{zhiruow,apurvag,gneubig,dfried}@cs.cmu.edu

Abstract

To succeed in common digital tasks such as web navigation, agents must carry out a variety of specialized tasks such as *searching for products* or *planning a travel route*. To tackle these tasks, agents can bootstrap themselves by learning task-specific skills online through interaction with the web environment. In this work, we demonstrate that *programs* are an effective representation for skills. We propose agent skill induction (ASI), which allows agents to adapt themselves by inducing, verifying, and utilizing program-based skills on the fly. We start with an evaluation on the WebArena agent benchmark and show that ASI outperforms the static baseline agent and its text-skill counterpart by 23.5% and 11.3% in success rate, mainly thanks to the programmatic verification guarantee during the induction phase. ASI also improves efficiency by reducing 10.7–15.3% of the steps over baselines, by composing primitive actions (e.g., `click`) into higher-level skills (e.g., `search_product`). We then highlight the efficacy of ASI in remaining efficient and accurate under scaled-up web activities. Finally, we examine the generalizability of induced skills when transferring between websites, and find that ASI can effectively reuse common skills, while also updating incompatible skills to versatile website changes.[1]

1 Introduction

To achieve success in common digital tasks such as web navigation, it is essential for agents to be able to perform a variety of specialized tasks such as searching for products on a shopping website (Yao et al., 2022; Deng et al., 2024) or finding a driving route on the map (Zhou et al., 2024a; Xie et al., 2024a). While one source for agents to learn such tasks is demonstrations annotated by humans (Deng et al., 2024) or synthesized with large language models (LMs) on websites of interest (Murty et al., 2024b;a), this can be a challenging offline learning procedure given the broad range of website domains and functionalities, especially for the collected demonstrations to match or cover the distribution of tasks queried at inference time (Zhou et al., 2024b); not to mention the limitations in resources to collect abundant high-quality data at ease (Pan et al., 2024).

Instead of learning from demonstrations offline, an alternative way is to learn these tasks directly online from test queries to prevent potential distribution mismatch between demonstration and downstream tasks (Levine et al., 2020). Some works propose to have agents induce casual abstractions (Majumder et al., 2024), single-state guidelines (Fu et al., 2024), or multi-step procedural workflows (Sarch et al., 2024; Wang et al., 2024b) as a form of intermediate knowledge to augment agent memory via non-parametric approaches (Brown et al., 2020). Nonetheless, most existing approaches represent this knowledge in text, offering limited quality and verification guarantees. In this work, we propose that *executable programs* are effective representations for intermediate skill acquisition, given their verifiability and composability advantages (Setlur et al., 2025).

We present ASI, namely agent skill induction (§2), that induces and applies programmatic skills along the process of solving user web navigation queries. More concretely, given a natural language (NL) query, the agent first generates an action trajectory attempting to solve the task using built-in, primitive actions such as `click` and `scroll`. The agent then

[1] https://github.com/zorazrw/agent-skill-induction

induces higher-level skills (e.g., `search_product(name)`) that wrap primitive actions or prior skills as executable programs, accompanied with corresponding test trajectories to verify their quality. Verified skills are then incorporated into the agent action space and can be directly called to solve future tasks with similar procedures, as depicted in Figure 1 (bottom).

We first evaluate ASI on the WebArena benchmark (Zhou et al., 2024a) (§3) and demonstrate that our online, adaptive ASI surpasses its static agent baseline without adaptive components by 23.5% in success rate. To validate the advantage of using programmatic representations for skills, we further compare to an adaptive agent, AWM (Wang et al., 2024b), that represents skills in memory as non-executable texts (Figure 1 top); we find ASI scores 11.3% higher success rate by employing verifiable, programmatic skills (Figure 1 bottom). Beyond the correctness aspect, the task-solving procedures by ASI-supported agents are 10.7–15.3% more efficient than the baseline approaches, mainly because of the action space abstraction and composition enabled by the programmatic skill representation.

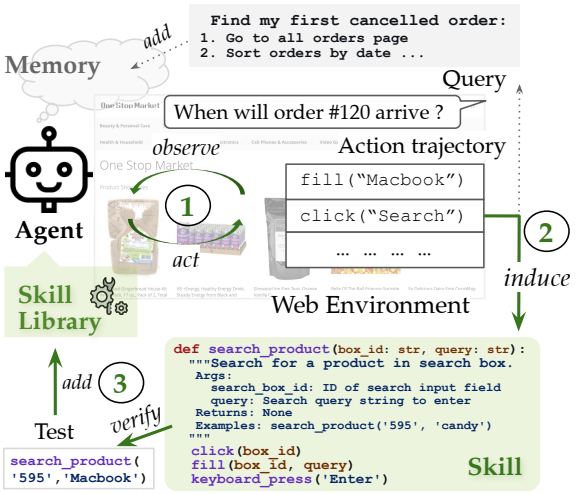

Figure 1: Online adaptive agent that induces and reuses programmatic skills as actions (bottom), as opposed to adding textual skills in memory (top).

We further stress test ASI on scaled-up scenarios (§4) that require substantially longer-horizon trajectories to complete the task. Across various domains such as shopping and social forums, we find the efficiency advantage offered by ASI is more pronounced, reducing action trajectories by 9.5 and 5.6 average steps compared to static and text-form agents. Facilitated by this improved procedural efficiency and planning, we find that ASI agent adheres to the optimal trajectory better and achieves tasks 20.7–38.9% more correctly.

Finally, we study agent behaviors on generalizing induced skills to other websites (§5), particularly from sandboxed, simulated websites to real-world websites of similar domains. While ASI agents effectively transfer common skills (e.g., `search_product`) to new websites, some skills may be incompatible with alternative website designs hence less useful. Nonetheless, ASI can quickly refine its prior skills or create new skills on new websites, indicating it allows agents to adapt online while maintaining verifiability via programs.

In short, ASI enhances web agent success and efficiency by inducing and applying verifiable programmatic skills, in general and longer-horizon tasks, even across varied websites.

2 Agent Skill Induction

In this section, we first lay out the web agent problem setup (§2.1) and introduce online, self-adaptive agents (§2.2). We then describe the core component of ASI— programmatic skill induction and verification (§2.3).

2.1 Problem Statement: Online Adaptive Agent

For the scope of this work, we focus on language model (LM) based agents, where each agent policy consists of an LM backbone $\mathcal{L}$, a memory $\mathcal{M}$, and a skill library $\mathcal{A}$, as illustrated in Figure 1 top and bottom. In the implementation, the memory $\mathcal{M}$ and the skill library $\mathcal{A}$ are provided as input context to the LM backbone. We denote the agent policy as $\pi_{\mathcal{L}}(\cdot|\mathcal{M}, \mathcal{A})$ and $\pi_{\mathcal{L}}$ for short. We focus on the web browser environment defined by a transition function $\mathcal{T}(s'|s, a)$ that models the change in the webpage after an action.

We focus on an online adaptation scenario where we have access to a sequence of NL queries $Q = \{q_1, q_2, \cdots, q_N\}$ specifying the tasks, and no other information such as demonstration

trajectories or ground-truth rewards are available (Wang et al., 2024a;b). For each task specified by a natural language (NL) query q, the agent generates a trajectory of actions $\tau = (s_0, a_0, s_1, a_1, \cdots, s_{H-1}, a_{H-1}, s_H)$ for a finite number of H steps. At each time step h in the horizon, the agent receives observation o_h from the current state s_h, and generates an action $a_h \in \mathcal{A}$ based on the observations and actions so far, via $\pi_{\mathcal{L}}(o_{0:h}, a_{0:h-1}; \mathcal{M}, \mathcal{A}) \rightarrow a_h$. The generated action will be executed on the environment and incurs a state change $\mathcal{T}(s_h, a_h) \rightarrow s_{h+1}$. This observe–act loop continues for H steps until the task reaches a task-terminating condition, such as the agent generating a termination action (e.g., `send_msg_to_user`) or the horizon reaches a pre-determined maximum number of steps $h = H_{max}$. We denote each pair of query and trajectory $(q, \tau) := e$ as an episode e. Agents can update the content in $\mathcal{M}$ and $\mathcal{A}$ and reuse them across episodes.

2.2 Inducing Reusable Skills

To realize online adaptive agents, one common approach is to induce skills from correct trajectories to update the agent (Wang et al., 2024b). But since ground-truth rewards are unavailable, an LLM-based evaluator $V_{\mathcal{L}}$ is often used to judge the correctness of episodes. Formally, from the total of N episodes throughout the online process $\{e^1, \cdots, e^N\} := \mathcal{E}$, we employ an LM-based evaluator $V_{\mathcal{L}}(e) \rightarrow 0/1$ to filter out the episodes predicted as correct $\mathcal{E}_V = \{e_i \in \mathcal{E} | V_{\mathcal{L}}(e_i) = 1, i \in \{1, \cdots, N\}\}$ and perform skill induction only on $\mathcal{E}_V$.

Central to our adaptive agents is an induction component I that enables the adaptivity of agents, which can be rule-based (Ellis et al., 2023; Grand et al., 2024) or instantiated by an LM $I(\cdot|LM)$ (Wang et al., 2024b); we follow the latter for its better performance and use I to represent the module for simplicity. For online adaptive agents $\pi_{\mathcal{L}}$, to induce skills, I is instructed to take in one filtered episode e and output one or more pieces of desired skills $D = \{d\}$, denoted as $I(e) \rightarrow \mathcal{D}$. Following AWM (Wang et al., 2024b), we update the agent in non-parametric ways that incorporate the induction outcome $I(e_t) \rightarrow d_t$ into the agent, instead of updating the parameters of the underlying LM backbone $\mathcal{L}$ for agent policy $\pi_{\mathcal{L}}$.

Unlike AWM which represents skills in free-form text representations and can only augment agent memory via $\mathcal{M}_t \cup \{d_t\} \rightarrow \mathcal{M}_{t+1}$ (Figure 1 top), we introduce ASI that represents skills as executable python programs, and directly integrate skills into the agent action space instead, via $\mathcal{A}_t \cup \{d_t\} \rightarrow \mathcal{A}_{t+1}$ (Figure 1 bottom).

2.3 Inducing and Verifying Programmatic Skills

To improve the induction quality, we propose a change in representation *from free-form text to executable programs*, which offers advantages in correctness and efficiency. For one, the program format enables ready verification on skill correctness by executing them; for another, skill programs abstract multiple lower-level actions into a higher-level function call, thus agents can solve tasks in fewer steps without tackling tricky low-level details.

Inducing Programmatic Skills We first clean the input episodes to ensure the induction quality. We remove all the steps that cause execution errors such as invalid argument format, to keep these invalid actions from distracting agent predictions. Furthermore, noticing the long and possibly redundant thought process generated by agents along with each action, we simplify each thought text paragraph into a short one-sentence description (e.g., "Clicked the directions button to access the route planning feature") using LM, effectively reducing the thought content from 87.9 to 13.4 tokens per step.

Given a clean input episode e, we now prompt the induction module I to produce one or more program functions to represent reusable skills $\mathcal{D} = \{d\}$ as executable programs. As exemplified in Figure 2, given the input episode on the left side, the induction module first produces two skills `open_marketing_reviews()` and `search_reviews(search_box_id, search_button_id, search_term)` in the form of callable program functions.

Skill Verification With the programmatic nature of ASI's skills, we can readily verify their correctness by executing them and checking if tasks can be solved successfully. While a naive way is to query the agent with the same NL query and allow it to use newly induced skill actions, we find agents may not always use new skills due to the large search space of

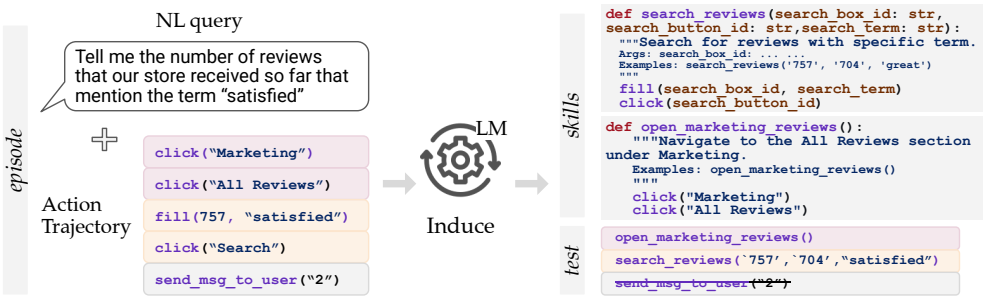

Figure 2: Inducing programmatic skills and rewriting the trajectory from an episode.

possible action trajectories. To have agents more efficiently generate trajectories that test skills in a more targeted way, we curate a rewritten trajectory prefix τ_D to constrain the first few steps executed in the environment, by rewriting and truncating the input action trajectory τ, and subsequently asking the agent to complete the prefix to get a full, checkable trajectory τ_f. Concretely, we first take the original action trajectory in the input episode τ (consisting of primitive actions or previously learned skills), and ask the induction module I to transform it to a skill-using trajectory (Figure 2 bottom right), by replacing sub-trajectories in τ with calls to the newly induced skill programs $\mathcal{D}$, if possible. Zooming into the Figure 2 example, this procedure merges `click('Marketing')` $\rightarrow$ `click('All Reviews')` to an `open_marketing_reviews()` call; transforms `fill(757, 'satisfied')` $\rightarrow$ `click('Search')` to a call of the second skill `search_reviews('satisfied')` with the specified term 'satisfied'; and adopted the last `send_msg_to_user('2')` step directly. Note that we follow Wang et al. (2024b) and induce skills according to each website, so some skills could be tailored to particular webpage contexts such as the 'Marketing' and 'All Reviews' link constants in `open_marketing_reviews`, while other skills apply to more versatile setups such as searching for different reviews in `search_reviews`.

Next, to avoid spurious successes in skill verification, we truncate the trajectory yielded above by removing any trailing primitive actions after the last call to a skill program. Taking Figure 2 as an example, in the original input trajectory, the last `send_msg_to_user('2')` already sends over the correct answer `'2'` to the user. However, if we directly adopt this last step into the skill-using trajectory τ_D, then executing it will always return the correct message to the user, regardless of whether the previous skill calls are valid. We thus remove such trailing actions to make sure verification attends to the induced skills we are testing.

After rewriting and truncation, we get this skill-using trajectory τ_D as the prefix to test skills. We now query the agent π again with the same NL query q and first execute τ_D on the environment. We then allow agents to continue generating up to $H_{max} - |\tau_D|$ actions to finish the task. In the Figure 2 example, to successfully solve query q, we expect the agent to generate another step of `send_msg_to_user(...)` with the correct answer `'2'` in the message. We take the concatenation of the trajectory prefix τ_D and the later additionally produced new steps τ_A (e.g., [`send_msg_to_user('2')`]) as the full trajectory τ_f. We then decide whether to add the induced programs $\mathcal{D}$ into the agent skill library as $\mathcal{A}$ by examining τ_f.

Specifically, we check τ_f from three dimensions: (1) *Correctness*: if executing τ_f successfully solves the task q as judged by the neural model evaluator $V_{\mathcal{L}}$; (2) *Skill Usage*: if the trajectory contains at least one call to at least one new skill in $\mathcal{D}$; and (3) *Skill Validity*: if all skill-calling actions cause environment changes. If all three boxes are checked, we add the skills being called in the trajectory τ_f to the agent skill library $\mathcal{A}_t \cup \mathcal{D}_{called} \rightarrow \mathcal{A}_{t+1}$. By adding $\mathcal{D}_{called}$, the agent can now generate actions that call these skill programs to solve subsequent tasks.

3 General Web Navigation Performance

3.1 Experiment Setup

Benchmark and Evaluation To evaluate ASI on general web navigation scenarios, we adopt the WebArena benchmark (Zhou et al., 2024a) that contains 812 test examples covering

Model	Method	# Steps	SR	Shop	Admin	Reddit	GitLab	Maps	Multi
GPT	Vanilla	-	12.3	13.9	10.4	6.6	15.0	15.6	8.3
	AWM	5.9	35.5	32.1	29.1	**54.7**	**35.0**	42.2	18.8
Claude	Vanilla	5.6	32.7	32.6	36.8	36.8	26.1	38.5	**20.8**
	AWM	5.9	36.3	34.8	39.0	51.9	28.9	39.4	18.8
	ASI (ours)	**5.0**	**40.4**	**40.1**	**44.0**	**54.7**	32.2	**43.1**	**20.8**

Table 1: WebArena success rate by adaptive agents with programmatic skills, in comparison to a static vanilla agent baseline, and a text-skill learning adaptive agent.

five major web activity domains: e-commerce, social forum, software development, content management, and travel. Each example in WebArena has an NL query q for the task, and a program-based evaluator that provides a binary 0/1 score for any given trajectory τ to judge if it successfully solves the task q. This program-based evaluator enables relatively rigorous evaluation based on the functional correctness of the action trajectory. We report the average score across all WebArena examples, if not specified otherwise.

Backbone LM and Agent Architecture We use the top-performing `claude-3.5-sonnet` model as the LM backbone for all components, including the agent policy π, the neural evaluator V, and the skill induction modules I. For experimentation, we use the BrowserGym (Chezelles et al., 2024) framework, which takes the webpage accessibility tree as observation, and instantiates the skill library $\mathcal{A}$ with the WebArena default action space listed in §A.

Baselines We take the vanilla Claude model with the BrowserGym framework (Drouin et al., 2024) as the non-adaptive agent baseline. Additionally, we compare ASI to AWM (Wang et al., 2024b), the current top-performing online adaptive web agent method. Because AWM was originally developed with the `gpt-4o` model, for a fairer comparison, we also experiment with AWM with `claude-3.5-sonnet` model as its LM backbone and also apply the episode cleaning procedure to enhance induction quality. We compare the two baseline methods with our ASI approach. We provide the complete prompts for each agent component: task-solving, episode evaluation, episode cleaning, and skill induction, in §A.

3.2 Results and Analysis

In Table 1, compared to the vanilla static-agent baseline, adaptive agents (AWM and ASI) generally achieve 11.0–23.5% higher success rates overall. Among adaptive agents, our ASI with programmatic skills, achieves another 11.3% success rate gain across websites, compared to its AWM counterpart that induces and uses textual skills. Meanwhile, ASI offers additional efficiency benefits by reducing the number of steps in solutions by 15.3% and 10.6% than vanilla and AWM agents, as one skill-call action can often execute multiple steps written in primitive actions used by vanilla and AWM agents. These advantages in correctness and efficiency are exhibited prominently across different websites and tasks, as shown by the website breakdown on Table 1 (right). Refer to §B for more analysis.

3.3 Why are Programmatic Skills Better?

To more concretely answer why programmatic skills are more effective than textual skills, we take a closer look on the two main differences between AWM and ASI: [1] whether the induction outcome is verified via execution, and [2] whether the induced skills are provided in memory for reference purpose only, or in the action space that allows execution.

Better Induction Quality We take the shopping website as a representative, and analyze the textual and program skills induced by AWM and ASI agents. We group textual and program skills by their functionality and show one representative example in Table 2. Compared to the clear functional boundary and highly-reusable granularity of the `search_product` skill, we find that the textual skills often have (1) more redundant steps, (2) example-specific context: e.g., the last text skill aims to find 'game accessories' while the steps generally applies to any product, and (3) fuzzier boundaries between separable tasks, e.g., the first skill mixes `product-search` and `add-to-wishlist` procedures, thus may not offer optimal guidance when asked to, e.g., search product and add it to cart instead.

Programmatic Skills

```
def search_product(search_box_id: str, query: str):
  """Search for a product using the search box.
   Args:
      search_box_id: ID of the search input field
      query: Search query string to enter
   Returns:
      None

  Examples:
      search_product('595', 'sony bluetooth headphones')
  """
  click(search_box_id)
  fill(search_box_id, query)
  keyboard_press('Enter')
```

Textual Skills

Task: Search for a product and add it to wish list
Action Trajectory:
```fill(621, {product_name})``` # Enter the product name in the search box
```click(478)``` # Click the search button to execute the search
```click({product_link})``` # Click the product to check more details
```click(1769)``` # Click the "Add to Wish List" link

Task: Search for a product's price range in the store
Action Trajectory:
```fill(565, {product_name})``` # Enter the product name in the search box
```click(570)``` # Click the search button to execute the search
```noop(1000)``` # Wait for search results to load
```send_msg_to_user({price_range_info})``` # Analyze and report the price range findings from the search results

---

Task: Search for gaming accessories within a date range
Action Trajectory:
```click(1274)``` # Navigate to the Video Games category
```fill(473, {search_terms})``` # Enter search terms including product name and year
```click(478)``` # Execute the search

Table 2: Example textual and program skills induced on the shopping website.

Verified Induction Improves End Success Rate From qualitative examination of the induction outcomes, we find roughly similar numbers of episodes evaluated as correct and used for induction (70 and 58 examples for AWM and ASI), ASI produced programs pass verification for only 15.6% of the turns, whereas AWM adds new skills for 31.4% of the time (replace or add none otherwise). While skill usage (in memory or as action, [2]) is designated for AWM and ASI, we hypothesize that verification [1] affects induction quality and thus end success. We thus experiment with another setting that induces programs (such that verification is enabled), and only use the induced skills in memory, to study the importance of induction quality. As shown in Table 3, inducing skills with execution-based verification (i.e., *(unverified, text)* $\rightarrow$ *(verified, program)*), while always present skills in memory, improves end success rate by 4.2 points, indicating the importance of higher-quality induction via verification. Yet it is still 3.7 points lower than ASI, suggesting the incompatibility of program format to agent memory. Indeed, we observe many cases where the agent tries to call the skill programs but unsuccessfully, since they are not supported in the action space.

Textual Representations Suit Memory Better To prevent the agent from trying to call these plausible programs, we ablate another setting that transforms program skills to textual format (as Table 2 right) and provide them in agent memory, dubbed *(verified, text)*. This format transformation effectively improves the overall success rate by another 2.6 points, getting a little closer to ASI. Given the different downstream usage, i.e., memory or actuation, textual and program formats may suit individual scenarios better.

Method		SR
Add to Memory	unverified, text	32.6
	verified, program	36.4
	verified, text	39.0
Add as Actions	verified, program	40.1

Table 3: Ablation study on induction verification and format on the shopping website.

Beyond basic web navigation tasks, in the next two sections, we examine agents in two other important scenarios, scaled-up activities (§4) and cross-website generalization (§5).

4 Scaled-Up Browsing Activities

The WebArena benchmark mainly features isolated, single-task scenarios, such as adding a single product to the shopping cart. However, in real-world practices, people need to do a series of such tasks together, such as adding multiple related products (e.g., coffee and
```

mug) to the cart before finally checking out. This browsing request can lead to extremely long-horizon tasks, sometimes with repetitive intermediate procedures. We identify this to be a scenario to further demonstrate the efficacy of program skills, as opposed to textual skills, as programs lend themselves naturally to repeated invocation and composition.

Therefore, we curate several case scenarios where the user asks for action-dense instructions, such as the tasks listed in Figure 3. Because the tasks are long-horizon and involve multiple sub-tasks, we follow Xu et al. (2024) and set up intermediate checkpoints to better track the intermediate progress of agents. Refer to §C.1 to see the full list of tasks and their evaluation checkpoints. We measure the success rate of each example by the percentage of checkpoints achieved by the agent. We report the average success rate of all examples, as well as the average number of steps taken to solve the tasks, in Table 4.

Method	Shopping		Admin		Reddit		GitLab		Map	
	sr ↑	# steps ↓	sr ↑	# steps ↓	sr ↑	# steps ↓	sr ↑	# steps ↓	sr ↑	# steps ↓
VANILLA	41.7	23.5	58.0	20.8	33.3	23.0	33.3	40.0	40.0	15.2
AWM	68.3	21.5	74.0	18.2	40.0	16.8	50.0	33.8	65.0	12.6
ASI (ours)	**100.0**	**16.3**	**91.0**	**14.2**	**55.0**	**12.8**	**55.0**	**25.4**	**100.0**	**6.2**

Table 4: Performance of vanilla, AWM, and ASI agents in scaled-up browsing scenarios. We perform statistical testing between ASI and each baseline and verify all improvements are statistically significant with t-statistics $|t| > 2$ and $p < 0.05$; see §C.3 for more details.

ASI Features Improved Efficiency Across all websites, ASI-produced trajectories have 6.6–14.6 and 4.0–8.4% fewer steps, compared to vanilla and AWM baselines, respectively. As the task horizon continues to grow when involving more intermediate checkpoints, this margin between ASI and baselines will predictably be more prominent.

Subsequent Benefits in Success Rate ASI also achieves higher success rates with more efficient trajectories, outperforming vanilla and AWM baselines by 38.9% and 20.7% on average. From manual analysis, we find this improvement comes from easier, better agent planning when using higher-level skills, without the need to tackle more complex procedures if only low-level primitive actions are available, as with vanilla and AWM agents.

Case Study: Changing Multiple Addresses We present a representative case on the shopping website: changing billing and shipping addresses after moving. As depicted in the top row in Figure 3, the vanilla agent without adaptive skills often roams into some irrelevant exploration steps, instead of sticking to the optimal route to solve the required task. It runs for minutes and exhausts the maximum steps (i.e., 50) before finishing the task.

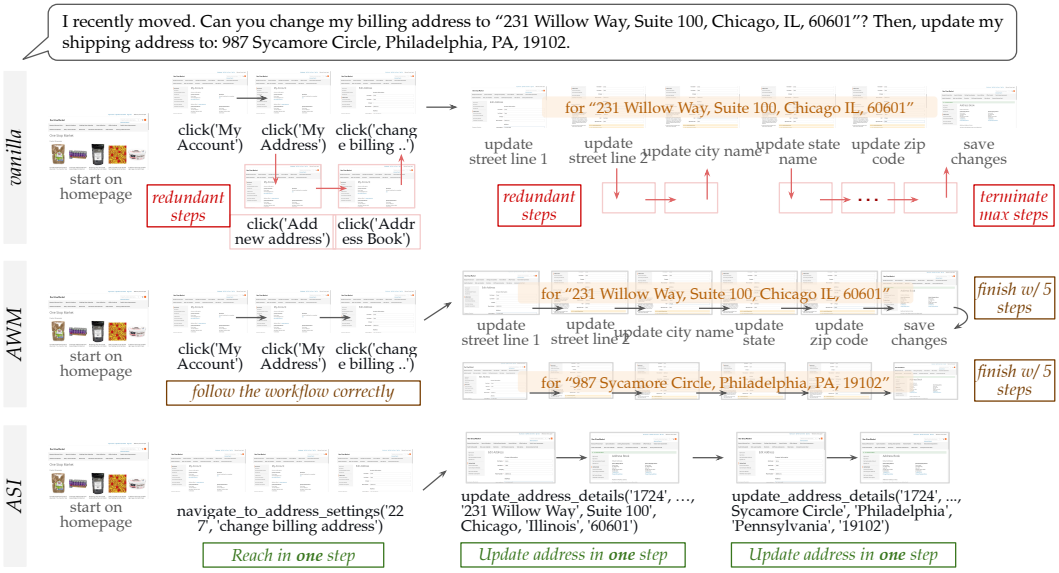

Figure 3: Example scaled-up task of updating multiple addresses on shopping website.

With AWM, adding textual skills in memory provides soft guidelines for agents to follow, the agent thus better sticks to the goal and finishes each part of the task (i.e., navigate to the address page, update billing/shipping address) step by step. Although successful, the trajectory is long, i.e., 27 steps, and still takes a few minutes to finish.

In comparison, ASI (in Figure 3 bottom row) showcases its efficiency by using learned skills to `navigate_to_address_settings` and `update_address_details` can solve each part in one step (vs. the 3–6 steps used by AWM for these parts). Overall, ASI correctly finishes all required actions in only 4 steps, shortening the horizon by 85.2% compared to AWM.

5 Adapting Across Websites

To examine whether agents can generalize with learned skills, we test agents on real-world website counterparts for some of the domains in WebArena as listed in Table 5. [2] This experiment setup can reflect on (1) transfer across different websites of the same domain, and (2) transfer from simulated, sandboxed to real-world websites.

Domain	WebArena Sandboxed	Real-World
shopping	OneStopMarket	Target
online forum	PostMill	Reddit
travel	OpenStreetMap	Google Maps

Table 5: Real-world in-domain website counterparts to each WebArena sandboxed website.

For each sandbox-real website pair, we take ten information-seeking style queries (He et al., 2024) in WebArena that do not involve potential privacy leakage or unrecoverable risky actions, such as making a purchase or changing user password. We provide the task details in §C.2. We compare ASI and AWM with their programmatic and textual skills as learned in §3, as well as comparing to the vanilla static agent baseline.

Transferring Common Skills In Figure 4, we can see how ASI can effectively reuse common skills such as `search_product` in the first step on the Target website.

Figure 4: ASI can generalize the `search_product` skill but face incompatibility when sorting items.

Incompatible Skills One challenge faced by ASI is that some prior skills become incompatible on the new website. For example, the `sort_by_listings()` induced on OneStopMarket selects options from a dropdown menu, yet sorting on the Target website opens a sidebar; despite their semantic similarity, the concrete actions in skill programs are no longer applicable. Still, we find that agents can often spot this incompatibility and rarely attempt to use these deprecated skills.

Adapting Skills to New Environment Although some skills induced on previous websites cannot be directly used on arbitrary new websites, we hypothesize that these skills can still serve as informative references on solving procedurally similar tasks or composing new skills targeted for the new website design.

Method	Shopping		Reddit		Map	
	sr ↑	# steps ↓	sr ↑	# steps ↓	sr ↑	# steps ↓
Vanilla	80.0	5.4	40.0	4.8	63.3	7.4
AWM	80.0	5.0	56.7	4.8	**100.0**	6.2
ASI	**90.0**	3.4	**76.7**	4.4	93.3	4.4
AWM + update	80.0	5.4	63.3	5.8	**100.0**	7.2
ASI + update	**90.0**	**3.2**	**76.7**	**4.0**	93.3	**4.2**

Table 6: Cross-website results. ASI significantly surpasses baselines in *sr* and # *steps* (with $|t| > 2$ and $p < 0.05$) from our analysis in §C.3.

[2] We did not test on administrative and software websites given their more severe safety concerns.

We thus allow agents to induce new skills or update previously acquired skills from experiences on the new website, denoted as *+update* entries in Table 6. We find that enabling skill update in both textual and program formats helps agent performance on new websites. Within the short online learning process (tens of examples), AWM adapts faster to the new websites, while ASI sees a more pronounced improvement in efficiency.

6 Related Work

Adaptive Digital Agents An important thread of agent-improving methods is to build adaptive agents that can autonomously self-improve from experiences. Most works focus on integrating past experiences into agent memory by collecting human annotation (Deng et al., 2024) or LM-based synthesis (Ou et al., 2024; Xu et al., 2025), especially via agent-driven exploration with instruction- (Murty et al., 2024b) or trajectory-driven (Murty et al., 2024a) approaches, offering warm starts on the websites of interest. Other works gather experiences (Wang et al., 2024b) or feedback (Qu et al., 2024) during test time, and augment them into memory through parametric channels such as supervised fine-tuning (Murty et al., 2024a), contrastive learning (Song et al., 2024), or reinforcement learning (Zhou et al., 2024b). Meanwhile, non-parametric approaches can directly augment reflections (Shinn et al., 2023), raw past experiences (Wang et al., 2023; Zheng et al., 2023), or further induced reusable workflows (Wang et al., 2024b). While these adaptive agents learn textual skills stored in memory, our ASI stores skills as verifiable and composable programs in the agent action space (i.e., skill library), thus enabling better quality and efficiency.

Skill Discovery and Learning Learning specialized skills for tasks in programmatic (Shin et al., 2019; Ellis et al., 2023; Cai et al., 2024; Wang et al., 2024a; Grand et al., 2024), embodied (Sharma et al., 2022; Wang et al., 2023; Liang et al., 2023; Sarch et al., 2024; Wong et al., 2024), and physical (Yu et al., 2023) environments has shown to success in agent performance. Particularly for digital agents built for web navigation tasks, most works focus on exploring skills offline with RL roll-outs (Gur et al., 2018; Liu et al., 2018; Putta et al., 2024; Qi et al., 2024) or LM-based prompting (Zhou et al., 2024b; Murty et al., 2024a; Patel et al., 2024). While this exploration stage could offer some supervised data to update the agent policy either parametric (Murty et al., 2024a; Patel et al., 2024) or non-parametrically (Zheng et al., 2023; Murty et al., 2024b), it often costs enormous extra computation and may suffer from the lack or mismatch in distribution with the downstream tasks at hand (Wang et al., 2024b). In contrast, our ASI does not rely on supervised data and can directly learn skills online without prior exploration.

Web Navigation Benchmarks Digital agents have been explored across a wide range of tasks (Yao et al., 2024; Kapoor et al., 2025; Xie et al., 2024b), among which one of the most popular application being browsing and navigating through versatile websites such as shopping (Yao et al., 2022), social media communication (Zhou et al., 2024a; Koh et al., 2024), knowledge work tasks (Drouin et al., 2024), and more (Deng et al., 2024). Our work focuses on general web navigation tasks using the WebArena (Zhou et al., 2024a) benchmark, meanwhile exploring other challenging scenarios such as scaled-up activities (Yoran et al., 2024) and cross-domain generalization (Deng et al., 2024).

7 Conclusion and Future Discussions

In this work, we present ASI to support web navigation agents to autonomously induce, verify, learn, and apply programmatic skills during online inference. Beyond achieving 23.5% success rate and 15.3% efficiency increases in general web tasks, we also showcase ASI's strengths for scaled-up web activities, thanks to the high-level action interface offered by the programmatic abstraction. Moreover, we examine skill generalizability to new, real-world websites, and find ASI still offers great efficiency while flexibly updating skills to new environments. While our work aims to offer insights on the optimal representation in agent skill acquisition, we still find multiple pieces in ASI worthy of further investigation, such as the conceptually or empirically suitable granularity of skills, the stability of the online evolving process, and the skill quality in comparison to human expert desiderata.

Acknowledgments

We would like to thank Jiayuan Mao, Yueqi Song, Boyuan Zheng, and Yu Su for the insightful discussions. We thank Yiqing Xie, Xinran Zhao, and Mingqian Zheng for their helpful comments on the paper draft. Zora is supported by the CMU Presidential Fellowship and Fujitsu Research. Apurva is supported by Amazon.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=qV83K9d5WB.

De Chezelles, Thibault Le Sellier, Maxime Gasse, Alexandre Lacoste, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, et al. The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024.

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? *arXiv preprint arXiv:2403.07718*, 2024.

Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *Philosophical Transactions of the Royal Society A*, 381(2251):20220050, 2023.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents. *CoRR*, abs/2403.08978, 2024. URL https://doi.org/10.48550/arXiv.2403.08978.

Gabriel Grand, Lionel Wong, Matthew Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. LILO: Learning interpretable libraries by compressing and documenting code. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=TqYbAWKMIe.

Izzeddin Gur, Ulrich Rückert, Aleksandra Faust, and Dilek Hakkani-Tür. Learning to navigate the web. *CoRR*, abs/1812.09195, 2018. URL http://arxiv.org/abs/1812.09195.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.371/.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *European Conference on Computer Vision*, pp. 161–178. Springer, 2025.

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

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ryTp3f-0-.

Bodhisattwa Prasad Majumder, Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A continually learning language agent for rapid task adaptation and generalization, 2024. URL https://openreview.net/forum?id=d5DGVHMdsC.

Shikhar Murty, Dzmitry Bahdanau, and Christopher D. Manning. Nnetscape navigator: Complex demonstrations for web agents without a demonstrator, 2024a. URL https://arxiv.org/abs/2410.02907.

Shikhar Murty, Christopher Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. Bagel: Bootstrapping agents by guiding exploration with language, 2024b. URL https://arxiv.org/abs/2403.08140.

Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=KjNEzWRIqn.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. Webcanvas: Benchmarking web agents in online environments. In *Agentic Markets Workshop at ICML 2024*, 2024. URL https://openreview.net/forum?id=O1FaGasJob.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. Large language models can self-improve at web agent tasks. *arXiv preprint arXiv:2405.20309*, 2024.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve, 2024. URL https://arxiv.org/abs/2407.18219.

Gabriel Sarch, Lawrence Jang, Michael Tarr, William W Cohen, Kenneth Marino, and Katerina Fragkiadaki. Vlm agents generate their own memories: Distilling experience into embodied programs of thought. *Advances in Neural Information Processing Systems*, 37: 75942–75985, 2024.

Amrith Setlur, Nived Rajaraman, Sergey Levine, and Aviral Kumar. Scaling test-time compute without verification or rl is suboptimal. *arXiv preprint arXiv:2502.12118*, 2025.

Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, May 2022. URL https://aclanthology.org/2022.acl-long.120/.

Eui Chul Shin, Miltiadis Allamanis, Marc Brockschmidt, and Alex Polozov. Program synthesis and semantic parsing with learned code idioms. *Advances in Neural Information Processing Systems*, 32, 2019.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents, 2024. URL https://arxiv.org/abs/2403.02502.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Zhiruo Wang, Graham Neubig, and Daniel Fried. TroVE: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *Forty-first International Conference on Machine Learning*, 2024a. URL https://openreview.net/forum?id=DCNCwaMJjI.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S Siegel, Jiahai Feng, Noa Korneev, Joshua B. Tenenbaum, and Jacob Andreas. Learning grounded action abstractions from language. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=qJ0Cfj4Ex9.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. In *Forty-first International Conference on Machine Learning*, 2024a. URL https://openreview.net/forum?id=l5XQzNkAOe.

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

Frank F Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on consequential real world tasks. *arXiv preprint arXiv:2412.14161*, 2024.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EEgYUccwsV.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

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

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, brian ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=SgTPdyehXMA.

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, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=oKn9c6ytLx.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. Proposer-agent-evaluator (pae): Autonomous skill discovery for foundation model internet agents. *arXiv preprint arXiv:2412.13194*, 2024b.

A Experiment Details

A.1 Agent Action Space

Table 7 shows the default action space the web navigation agents we employed in all the experiments. This action space remains the same for both (i) static, vanilla agent, as well as the (ii) adaptive agent that learn textual skills in memory, i.e., AWM.

Action Type	Description
`noop(wait_ms)`	Do nothing for specified time.
`click(elem)`	Click at an element.
`hover(elem)`	Hover on an element.
`fill(elem, value)`	Type into an element.
`keyboard_press(key_comb)`	Press a key combination.
`scroll(x, y)`	Scroll horizontally or vertically.
`select_option(elem, options)`	Select one or multiple options.
`goto(url)`	Navigate to a url.
`go_back()`	Navigate to the previous page.
`go_forward()`	Navigate to the next page.
`new_tab()`	Open a new tab.
`tab_close()`	Close the current tab.
`tab_focus(index)`	Bring tab to front.
`send_msg_to_user(text)`	Send a message to the user.
`report_infeasible(reason)`	Notify user that instructions are infeasible.

Table 7: Initial action space for agents throughout our experiments.

A.2 Prompts for LLM-Based Components

Prompt for Episode Evaluation

You are an expert in evaluating the performance of a web navigation agent. The agent is designed to help a human user navigate a website to complete a task. Given the user's intent, the agent's action history, the final state of the webpage, and the agent's response to the user, your goal is to decide whether the agent's execution is successful or not. Please be careful of each detail and strict about the evaluation process.
There are three types of tasks: 1. Information seeking: The user wants to obtain certain information from the webpage, such as the information of a product, reviews, map info, comparison of map routes, etc. The bot's response must contain the information the user wants, or explicitly state that the information is not available. Otherwise, e.g. the bot encounters an exception and respond with the error content, the task is considered a failure. Besides, be careful about the sufficiency of the agent's actions. For example, when asked to list the top-searched items in a shop, the agent should order the items by the number of searches, and then return the top items. If the ordering action is missing, the task is likely to fail. 2. Site navigation: The user wants to navigate to a specific page. Carefully examine the bot's action history and the final state of the webpage to determine whether the bot successfully completes the task. No need to consider the bot's response. 3. Content modification: The user wants to modify the content of a webpage or configuration. Carefully examine the bot's action history and the final state of the webpage to determine whether the bot successfully completes the task. No need to consider the bot's response.
IMPORTANT Please be strict about the evaluation process. Format your response into two lines as shown below:
Thoughts: <your thoughts and reasoning process>
Status: "success" or "failure"

Prompt for Episode Cleaning

You are a helpful assistant in summarizing web browsing actions. Your task is to summarize the main action taken and the corresponding state change, from the input thought process and environmental description.
Example input: "'Let me think about this step by step: 1) I've clicked the Columns button and based on the accessibility tree, I can now see a list of available columns that can be shown/hidden. 2) Looking through the available columns in the accessibility tree, I don't see any direct column for "Number of Orders" or similar metric. 3) Since I don't see the information I need in the available columns, this suggests that: - Either this information is available through a different path - Or we need to access the Sales/Orders section to correlate this information 4) The most logical path would be to: - Go to the Sales section where we can see all orders - And then analyze which customers have the most orders 5) From the accessibility tree, I can see the "SALES" link with bid "183" in the main navigation.
Let me click on the SALES link: This should open the Sales menu where we can access the Orders section to analyze customer order histories."'
Example output: "'Clicked the SALES link to access the Orders section for customer order histories."'

Prompt for Skill Induction

You are a proficient software engineer. Your task is to (1) summarize reusable functions as APIs from the provided action trajectories, and (2) rewrite the trajecoties using the reusable functions you generated in (1).
For (1), from the provided examples about the same task, you job is to generate Python functions that can be reused to solve (part of) these tasks. The functions should have mediocre complexity: (i) containing at least three actions and not too simple (e.g., a single line of code), (ii) not too complex (e.g., more than 10 lines of code), and should be general enough to be applied to other similar tasks. The arguments to these functions should be common variables (such as strings and lists), avoid using complex inputs such as another function.
The actions you can use are: 'click', 'fill', 'hover', 'keyboard_press', 'scroll'; 'tab_focus', 'new_tab', 'tab_close', 'go_back', 'go_forward'; 'send_msg_to_user', 'report_infeasible', 'select_option'. Do not use other undefined actions. Do not include any try-except blocks in the functions.
Please include 'Args', 'Returns', and 'Examples' in the function documentation.
For (2), write the instruction and rewritten code of each example. Do not include the answer response or example-specific information in the rewritten code. Pay attention to whether all link IDs are available before specifying them in the generated functions. If you use 'send_msg_to_user', make sure the message is decided within the function, instead of provided as an argument.
Make sure each function contains no less than 2 steps, and no more than 5 steps; to keep the functions simple and task-oriented. You can generate zero, one, or multiple functions depending on the provided examples.

B Skill Induction: Analysis

We provide more details about the skill induction process, in skill curation and reuse frequency (§B.1) and representative skill case studies (§B.2)

B.1 Skill Induction and Reusability

To provide more insights on how agents curate and reuse programmatic skills, for the main experiments on WebArena, we calculate the number of examples that (i) attempt to induce a new skill, (ii) successfully induce a new skill, and (iii) reuse a previously induced skill.

As shown in Table 8, the agent reuses at least 1 skill for 42.5% of the cases, which is pretty frequent across websites. Moreover, these skills are created using 7.6% of the tasks, demonstrating the high scalability of our skill learning approach.

Domain	Number of Examples			
	Attempted Induction	Successful Induction	Reuse Examples	Total
shopping	21	8	96	180
admin	38	15	108	156
reddit	24	11	14	105
map	13	10	80	109
gitlab	25	11	10	175

Table 8: Analyzing the number of examples that attempt, induce, and reuse skills.

B.2 Representative Skills

We demonstrate two representative types of skills, which (i) chain frequently-used-together actions, and (ii) involve advanced programming primitives.

Chaining Primitive Actions The most common type of skills chains multiple frequently-used-together primitive actions (e.g, `click`, `fill`) in a certain order, to reflect a particular common usage, such as the `search_product` tool illustrated below. This type of skill provides an easy portal for frequent workflows.

```
def search_product(name: str):
    click("Search textbox...")
    if has_popup_window():
        click("Close")
    fill("Search textbox", name)
    click("Search button")
```

Advanced Programming Primitives Besides a sequential chaining of actions, some skills contain more advanced logics, such as if-else or for/while loops, as the two examples below.

```
def navigate_and_sort_category(category_id: str, subcategory_id: str |
    None = None):
    """Navigate to a product category and sort items by price.

    Args:
        category_id: ID of the main category link
        subcategory_id: Optional ID of the subcategory link, if needed

    Returns:
        None

    Examples:
        navigate_and_sort_category('1241', '1873')  # PS4 under Video
            Games
        navigate_and_sort_category('1245')  # For main category only
    """
    click(category_id)  # Click main category
    if subcategory_id:
        click(subcategory_id)  # Click subcategory if provided
    select_option("1553", "Price")  # Sort by price ascending
```

```
def browse_category_by_navigation(menu_id_sequence: list):
    """Browse products by navigating through a sequence of menu IDs.
    This function allows navigation through a series of menu interactions.
    Args:
        menu_id_sequence (list): A list of menu IDs to interact
            sequentially, using hover actions followed by a click.
```

```
6
7
8     Example usage:
9         browse_category_by_navigation(['735', '786', '797']) # Navigates
              Home & Kitchen -> Storage & Organization -> Racks, Shelves &
              Drawers
10    """
11    for idx, menu_id in enumerate(menu_id_sequence[:-1]):
12        hover(menu_id)
13    click(menu_id_sequence[-1])  # Click the final id to land in the
          predefined category
```

C Scaled-Up and Cross-Website Tests

We provide the full list of tasks used in scaled-up (§4) and cross-website (§5) analyses in §C.1 and §C.2, respectively. In §C.3, we further perform significance testing to validate the findings from Table 4 and Table 6.

C.1 Scaled-Up Tasks

Table 9, Table 10, Table 11, Table 12 and Table 13 shows example scaled-up tasks studied on the shopping, admin, social forum, software development, and map websites.

Instruction	Checkpoints	Score
Add a wireless headphone, a water bottle, a notebook, a ground coffee, and a mug to my shopping cart.	Add a wireless headphone to cart; Add a water bottle to cart; Add a notebook to cart; Add a ground coffee to cart; Add a mug to cart.	5
Add the most expensive item from the video games category, the cheapest item from the Office Products category, and the most relevant coffee mug to my shopping cart.	Add the most expensive item from the video games category to cart; Add the cheapest item from the Office Products category to cart; the most relevant coffee mug to my shopping cart.	3
Add the cheapest wireless headphone, a water bottle, the most expensive notebook, a ground coffee, and a mug to my shopping cart.	Add the cheapest wireless headphone to cart; Add a water bottle to cart; Add the most expensive notebook to cart; Add a ground coffee to cart; Add a mug to cart.	5
Show me the ordered items for each cancelled order from Feb to May in 2023.	Show me the 5/17/23 order; Show me the 2/24/23 order; Show me the 2/11/23 order.	3
Iterative update my billing address to 231 Willow Way, Suite 100, Chicago, IL, 60601. Then, update my shipping address to 987 Sycamore Circle, Philadelphia, PA, 19102.	Successfully update my billing address; Successfully update my shipping address.	2

Table 9: Exemplar scaled-up browsing tasks on the shopping website.

C.2 Cross-Website Tasks

Table 14, Table 15, and Table 16 lists example tasks to test agent generalization abilities on shopping (OneStopMarket to Target), social forum (Postmill to Reddit), and software development (GitLab to GitHub) domains.

C.3 Significance Testing

Instruction	Checkpoints	Score
Tell me the the number of reviews that our store received by far that mention terms 'disappointed', 'satisfied', 'decent', 'not useful', and 'best'.	Return the correct number for terms 'disappointed', 'satisfied', 'decent', 'not useful', and 'best'.	5
I need to contact a list of customers. Find the customer name and email with phone number 2058812302, 2137418080, 2065555555, 8015551212, and 555-229-3326.	Return the correct name and email information for customers with each of the five phone numbers.	5
I will need to update our webpage to create a more energetic vibe. Change the page title of '404 Not Found' to 'Bruh bro you clicked the wrong page', the page title of 'Enable Cookies' to 'Cookie monster coming to your place', the page title of 'Home Page' page to 'This is the home page!!', the page with title 'Privacy Policy' to 'No privacy policy is needed is this dystopian world', and lastly, change the page 'About Us' to 'Secret'.	Change the page title correctly for each of the five pages.	5
I need to generate a bunch of report to show to the store manager in an hour. Could you help me generate a sales order report for the last month, over the last 45 days, and for Q1? I'll also need a refund report for last year, and a tax report for this year. Today is 3/15/2023.	Generate a sales report for 2/1/2023-2/29/2023; generate a sales report for 1/29/2023-3/15/2023; generate a sales report for 1/1/2023-3/15/2023; Generate a refund report for 1/1/2022-12/31/2022; Generate a tax report for 1/1/2023-3/15/2023.	5
Tell me the SKU of products that have 10 units, 3 units, and 0 units left. Also, give me the product names that have 2-3 units left.	Return the correct SKU for the first three questions; return the correct product names for the last question.	4

Table 10: Exemplar scaled-up browsing tasks on the shopping admin website.

Scaled-Up Tasks We conduct t-tests between (i) ASI and AWM, (ii) ASI and VANILLA agent. From the results in Table 17, we find the advantage of ASI in success rate and efficiency improvements, when comparing to both AWM and VANILLA agents, are statistically significant, as indicated by all t-statistics with absolute values over 2 and p-value below 0.05.

Method Pair	Success Rate		# Steps	
	t-stat	p-value	t-stat	p-value
ASI vs. AWM	-2.3601	0.0226	2.7664	0.0068
ASI vs. VANILLA	-4.0922	0.0002	2.1983	0.0296

Table 17: Results of significance testing on ASI's advantages for scaled-up web tasks.

Cross-Web Tasks We conduct similar significance testing on cross-web tasks and report the results in Table 18. While ASI does not significantly outperform AWM in success rate, given the presumably greater flexibility of textual workflows, ASI still exhibits significant advantages on the efficiency side. Furthermore, comparing ASI to static VANILLA agents, ASI achieves significant improvements in both success rates and efficiency (i.e., reduced number of steps), suggested by $|t| > 2$ and $p < 0.05$.

Method Pair	Success Rate		# Steps	
	t-stat	p-value	t-stat	p-value
ASI vs. AWM	-1.3980	0.1673	2.1238	0.0378
ASI vs. VANILLA	-3.5984	0.0007	2.5792	0.0125

Table 18: Results of significance testing on ASI's advantages for cross-web tasks.

Instruction	Checkpoints	Score
I'm planning to organize multiple meetups in the next few months. Help me post notices on virtual meetups for the little women on Apr 10th, for Harry Potter in May 15th, and for Jane Eyre in Jan 30th, in the most suitable forums in PostMill.	Post Apr 10th meetup; Post about May 15th meetup; Post Jan 30th meetup. All in book-related forums.	3
Could you tell me all forums with names related to computer science?	must include: deeplearning (1 pt), MachineLearning (1 pt); optionally (get 1 score if include any): science, askscience, technology.	3
Find the most relevant posts about jerseycity, newjersey, and nyc; and tell me how different they are.	Correctly find post about jerseycity; Correctly find post about newjersey; Correctly find post about nyc; Answer how different they are.	4
Thumbs down the top-2 posts in jerseycity, newjersey, and nyc forums, I don't like them.	Thumbs down the top-2 posts in the jerseycity forum; Thumbs down the top-2 posts in the newjersey forum; Thumbs down the top-2 posts in the nyc forum.	3
Reply "Thank you! This is super helpful!" to three posts about long-distance relationship advice.	Reply to three posts with the correct message. Need to be relevant to long-distance relationship advice.	3

Table 11: Exemplar scaled-up tasks on the Postmill website.

Instruction	Checkpoints	Score
Display the list of issues in the a11yproject/a11yproject.com repository that have labels related to 'help needed', and assign the most recent one to the top contributor of this repository.	Display the help-wanted issues; find the top contributor; assign him to the most recent help-needed issue.	3
Set up a new, empty repository with the name `agent_skill_induction`, and create a MIT license file. Then, invite Abishek and Vinta as collaborators.	Create a new repository with given name; Create a MIT license inside; Invite both collaborators.	3
Start a private project `web_agent_android_xl` with Android template and add primer, convexegg, abishek as members.	Create the repository private and with Android template; Invite all three people as members.	2
Add the following users to repo `a11y-webring.club` as developer: [abisubramanya27, lahwaacz], and [yjlou, a11yproject] as maintainer.	Add abisubramanya27 and lahwaacz as developers; Add yjlou and a11yproject as maintainers.	2
Add the following users [abisubramanya27, lahwaacz, yjlou, a11yproject] to repo `a11y-webring.club`, make sure to assign them different roles.	Add abisubramanya27 with role 1; Add lahwaacz with role 2; Add yjlou with role 3; Add a11yproject as role 4. Role 1–4 need to be all different.	4

Table 12: Exemplar scaled-up tasks on the GitLab website.

Instruction	Checkpoints	Score
Search for the closest restaurants, cafes, parking, and banks to Carnegie Mellon University on the map.	Return the closest restaurants; Return the closest cafes; Return the closest parking; Return the closest banks.	4
I will need to go to multiple places from Carnegie Mellon University today, including the Univ of Pittsburgh, UPMC shadyside, the Schenley park, and Squirrel Hill. Could you should me the driving route to all those places?	Show me driving route from CMU to UPitt; Show me driving route from CMU to UPMC; Show me driving route from CMU to Schenley Park; Show me driving route from CMU to Squirrel Hill.	4
Show me the route of driving from CMU to University of Pittsburgh, then walking to the Schenley Park; next, bike to UPMC shadyside, and walk to Squirrel Hill after that.	Show me CMU $\rightarrow$ Upitt route by car; Show me Upitt $\rightarrow$ Schenley Park route by foot; Show me Schenley Park $\rightarrow$ UPMC route by bike; Show me UPMC $\rightarrow$ Squirrel Hill route by foot.	4
Check if the Univ of Pittsburgh, UPMC shadyside, schenley park, and squirrel hill can be reached within one hour by walking, if departing from Carnegie Mellon University.	Return yes to route 1, route 2, route 3, and route 4.	4
Tell me the coordinates of Univ of Pittsburgh, UPMC shadyside, schenley park, squirrel hill, and CMU in DD format.	Return the coordiates of each of the four places.	4

Table 13: Exemplar scaled-up tasks on the Map website.

Instruction	Checkpoints	Score
Show me the options for Canon photo printer?	Return the correct search result.	1
I have a lot of Nintendo Switch game cards now, help me find the best storage option to fit all 11 cards.	Return one valid product.	1
What is the price range for beauty products?	Return the correct price range.	1
Show me products under $25 for woman shoes	Display correct products.	1
Show the least expensive shoe storage with a minimum storage capacity of 12 pairs.	Display correct products.	1

Table 14: Exemplar shopping tasks on the target website.

Instruction	Checkpoints	Score
Tell me the names of books recommended in the latest five posts in the books forum	Find the r/books forum; Find the most recent 5 posts; Give the correct answer.	3
Tell me the titles of the 5 most recent posts about little women in the books forum	Find the r/books forum; Find little women related posts; Sort the posts by newest.	3
What are the recommended products for noise-canceling headphones within a budget of $200 in r/headphones	Find the r/headphones forum; Correctly search with noise-canceling, under $200 requirements; Return a valid headphone recommendation.	3
Find 3 pieces of advices about deal with long-distance relationships in a subreddit for relations.	Navigate to a forum about relations; find at least 3 pieces of advice from relevant posts.	2
Find if there are any jeep wrangler meetups. If so, when and where?	Search in jeep wrangler related forums; Return a valid answer based on the search result.	2

Table 15: Exemplar social forum tasks on the reddit website.

Instruction	Checkpoints	Score
Tell me the full address of all international airports that are within a driving distance of 30 miles to Carnegie Mellon University	Return Pittsburgh International Airport.	1
I will arrive Pittsburgh Airport soon. Provide the name of a Hilton hotel in the vicinity, if available. Then, tell me the the walking distance to the nearest supermarket own by a local company from the hotel.	Show me the hotels; Find a nearby supermarket; Show me the walking route from the hotel to the supermarket.	3
Show me the walking route from nearby hotels to CMU, Pittsburgh that take at most 5 minutes?	Find a hotel that meets the walking time requirement; Show me the walking route.	2
I am at CMU Pittsburgh, how long it takes to the nearest USPS postal office with different transportation methods?	Return travel time by car, by foot, by bus, and by bike.	4
Tell me the coordinates of Carnegie Mellon Cafe in DD format.	Return the correct coordinates.	1

Table 16: Exemplar social forum tasks on the Google Maps website.