WebShortcutter: Enhancing Multimodal Web Agents through Failure Memory and Reusable Shortcut Synthesis

Anonymous ACL submission

Abstract

Multimodal web agents repeatedly make the same website-specific mistakes and redundantly execute identical multi-step procedures, yet current approaches offer no mechanism for retaining and reusing such environment-level experience. We propose WebShortcutter, which enhances web agents through two complementary mechanisms: (1) a **failure memory** that distills recurring error patterns from failed trajectories into an actionable mistake notebook, and (2) **reusable shortcut synthesis** that abstracts common multi-step action sequences from successful trajectories into executable, high-level skill functions directly invocable by the agent. The two components compose in a simple cascaded pipeline: the failure memory first improves trajectory quality, and shortcuts are then synthesized from the improved trajectories. On four WebVoyager websites, WebShortcutter improves task success rate from 52.6% to 70.5% while reducing average trajectory length by 28.8%, demonstrating that structured accumulation and reuse of website-level experience substantially benefits open-world web navigation.

1 Introduction

Building autonomous agents for real-world web navigation remains challenging despite the strong reasoning capabilities of large multimodal models (LMMs) (Zhou et al., 2023; Gur et al., 2023; Deng et al., 2024; Koh et al., 2024). Recent multimodal web agents such as WebVoyager (He et al., 2024) and SeeAct (Zheng et al., 2024) can interpret webpage screenshots and perform actions through paradigms like ReAct (Yao et al., 2022). However, despite their impressive general reasoning, these agents still struggle with two fundamental issues on real-world websites.

First, agents *repeat the same website-specific mistakes* across tasks. Agents may consistently fail to dismiss pop-up dialogs or mis-select calendar dates, etc. These errors are persistent: without a mechanism to record and recall past failures, the agent has no way to avoid them in future episodes. Second, agents *redundantly execute identical multi-step procedures*. Searching on Google Flights always requires the same sequence of entering origin, destination, dates, and clicking search. Yet the agent re-derives each step from scratch every time, incurring unnecessary reasoning overhead and accumulating error risk with each additional step.

Both problems share a common root: current web agents lack a mechanism for *persistent, environment-level experience reuse*. All reasoning is performed within each episode's context window, with no structured knowledge that accumulates across tasks on the same website. This stands in contrast to how humans navigate websites—we develop site-specific mental models and routinized procedures that transfer across tasks.

We propose WebShortcutter, a framework that equips multimodal web agents with two complementary mechanisms for website-level experience reuse:

- **Failure memory**: a concise *mistake notebook* distilled from failed trajectories, capturing website-specific pitfalls as actionable natural-language lessons.
- **Reusable shortcut tools**: executable functions synthesized from successful trajectories, each encapsulating a common multi-step sub-task as a single invocable skill.

These two mechanisms compose naturally in a cascaded pipeline: the failure memory produces cleaner trajectories, from which more reliable shortcuts are then synthesized. At inference, the agent is augmented with shortcut tools, which compile both the corrected behaviors and the procedural abstractions into deterministic, executable code.

We evaluate WebShortcutter on four WebVoyager benchmark webs (He et al., 2024). Results show that shortcut tools improve overall task suc-

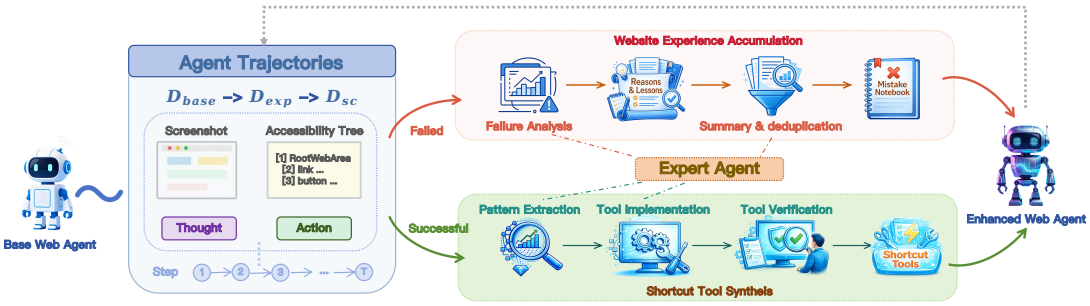


Figure 1: Overview of WebShortcutter. **Stage 1**: failed trajectories from $\mathcal{D}_{\text{base}}$ are analyzed to produce a mistake notebook $\mathcal{E}_w$. **Stage 2**: the notebook-augmented agent re-runs tasks to obtain improved trajectories $\mathcal{D}_{\text{exp}}$; successful ones are distilled into shortcut tools. **Inference**: the agent uses shortcut tools only.

cess rate from 52.6% to 70.5%, with gains up to +21.5% on Google Flights, while reducing average trajectory length by 28.8%. Our work demonstrates that structured accumulation and reuse of exploration experience is a promising direction for open-world web agents.

2 Related Work

Multimodal web agents. Early LLM-based agents relied on HTML or DOM representations (Zhou et al., 2023; Gur et al., 2023; Nakano et al., 2021). Recent multimodal agents such as WebVoyager (He et al., 2024), SeeAct (Zheng et al., 2024), and VisualWebArena (Koh et al., 2024) incorporate screenshots as visual inputs. However, most do not improve with accumulated experience.

Self-improving agents. AgentQ (Putta et al., 2024) combines MCTS with preference optimization, and OpenWebVoyager (He et al., 2025) proposes an exploration-feedback-optimization cycle. These methods improve at the general level but do not exploit website-specific patterns. Reflexion (Shinn et al., 2023) maintains verbal failure memory but does not synthesize reusable procedural tools from experience.

Tool synthesis. Automatic tool creation has been explored by Cai et al. (2024) and Yuan et al. (2024). Our work specializes tool synthesis to web navigation, where shortcuts must handle dynamic page elements and compose with one another to manage complex interaction flows.

3 Method

3.1 Overview

Following OpenWebVoyager (He et al., 2025), we model web navigation as a POMDP where the agent receives multimodal observations $o_t = (o_t^a, o_t^s)$ (accessibility tree and screenshot) and generates thought-action pairs (h_t, a_t). A trajectory τ is the complete sequence for a task, labeled as successful or failed by an automatic evaluator. We assume a set of base trajectories $\mathcal{D}_{\text{base}}$ collected by running the base agent on a task set.

WebShortcutter constructs two types of experience for each website w, as illustrated in Figure 1.

3.2 Failure Memory

For each website w, we collect failed trajectories $\mathcal{D}_w^- \subset \mathcal{D}_{\text{base}}$ and prompt a stronger Expert Agent to analyze each failure. The Expert Agent examines the full trajectory including screenshots and accessibility trees, identifies the root cause in concrete terms, and formulates a generalizable lesson that avoids referencing specific element IDs. The output is a structured (Reason, Lesson) pair per trajectory. Individual lessons are then consolidated via a summarization step that removes redundancy:

$$\mathcal{E}_w = \textsc{Summarize}\left(\{(\text{Reason}_i, \text{Lesson}_i)\}_{i=1}^{|\mathcal{D}_w^-|}\right)$$

The resulting mistake notebook $\mathcal{E}_w$ is prepended to the agent's system prompt during a second round of trajectory collection, providing website-specific guidance. Table 1 shows representative entries from the Booking notebook, which capture website-specific sequential dependencies (e.g., filters only work after submitting a search) and hidden UI patterns (e.g., collapsible filter sections) that the base agent repeatedly fails to handle.

3.3 Reusable Shortcut Synthesis

From improved successful trajectories $\mathcal{D}_w^+ \subset \mathcal{D}_{\text{exp}}$, the Expert Agent identifies recurring multi-step action patterns and synthesizes each into an executable Python function. Each shortcut adopts a

Example Lessons from Booking Mistake Notebook
1. After entering destination and dates, always click the **Search button** first before applying any filters. 2. Scroll the filter panel thoroughly; look for "Show all" buttons to expand collapsed sections. 3. When applying filters, verify each checkbox and do not remove previously applied filters.

Table 1: Representative entries from the Booking failure memory, capturing website-specific pitfalls.

stateless entry design: it re-reads the accessibility tree to determine the current page state, locates target elements via pattern matching, and executes actions with explicit retry and fallback logic. This defensive design ensures robustness when the shortcut is invoked at different points in a trajectory.

Shortcuts on the same website may also *cooperate*: on Allrecipes, the scroll-to-section shortcut internally delegates to the ad-closing shortcut whenever a modal overlay is detected mid-scroll, enabling complex interaction flows involving interleaved sub-tasks.

For each website, natural-language descriptions of shortcuts are appended to the system prompt. The agent invokes a shortcut via a special `Exec` action that calls the corresponding function with appropriate arguments, compressing multiple navigation steps into a single function call. To prevent cascading failures, each shortcut may be invoked at most once per trajectory; if the desired effect is not achieved, the agent falls back to standard actions.

While the Expert Agent drafts shortcuts automatically, a human developer verifies and tests each function in the target environment and provides feedback for revision—a semi-automated process essential for handling the fragility of dynamic web content (see Appendix B for examples).

3.4 Cascaded Pipeline

Rather than applying both mechanisms in parallel—which would cause shortcuts to inherit the base agent's systematic mistakes—we adopt a **cascaded** design: (1) collect $\mathcal{D}_{\text{base}}$; (2) extract $\mathcal{E}_w$ from failures; (3) collect improved trajectories $\mathcal{D}_{\text{exp}}$ with $\mathcal{E}_w$; (4) synthesize shortcuts from successes of $\mathcal{D}_{\text{exp}}$; (5) deploy the agent with shortcuts. The notebook's lessons are effectively "compiled" into the shortcut implementations—encoded as robust control flow rather than natural-language reminders—and need not be re-injected at inference.

4 Experiments

4.1 Setup

We build upon OpenWebVoyager (He et al., 2025) with **GPT-5.1** (OpenAI, 2025) as the web agent backbone and evaluator, and **Claude Opus 4.6** (Anthropic, 2026) as the Expert Agent for failure analysis and shortcut synthesis. We evaluate on four websites from the WebVoyager test set: **Booking** (44 queries), **Coursera** (42), **Google Flights** (42), and **Allrecipes** (45). We report **Task Success Rate** (auto-evaluated by GPT-5.1) and **Average Trajectory Length** (number of steps).

We compare three configurations: **Base** (standard prompting), **+Exp** (augmented with mistake notebook $\mathcal{E}_w$), and **+SC** (WebShortcutter, augmented with shortcut tools synthesized from $\mathcal{D}_{\text{exp}}$).

4.2 Main Results

Table 2 presents per-website and overall results for all three configurations.

Shortcuts yield the largest gains. +SC improves overall success from 52.6% to 70.5%. The largest per-site gains appear on Google Flights (+21.5%) and Coursera (+21.4%). Google Flights' rigid form structure is particularly amenable to procedural abstraction: a single shortcut replaces the entire origin/destination/date-entry workflow. Coursera benefits from shortcuts that reliably handle landing-page variations and dismiss cookie banners and iframe-based pop-ups. Allrecipes gains (+17.7%) stem from shortcuts that scroll to specific content sections while automatically dismissing advertisement overlays. Booking gains are smaller (+11.3%) due to more complex result-page scenarios and occasional network fluctuations.

Failure memory provides smaller, instruction-dependent gains. +Exp improves overall success to 59.0%, with stronger effects on Coursera (+9.5%) and Allrecipes (+8.8%) where failure patterns are concrete and unambiguous. Gains are more modest on Booking (+2.3%) and Google Flights (+4.8%): the notebook only biases reasoning at the thought level without guaranteeing correct actions, and over-generalized lessons can actively degrade performance.

Notebook value is realized through shortcuts. Although +SC does not include $\mathcal{E}_w$ at inference, its shortcuts were synthesized from notebook-improved trajectories. The lessons are compiled

Agent	Booking	Coursera	Google Flights	Allrecipes	Overall
Task Success Rate (%)					
Base	43.2	54.8	57.1	55.6	52.6
+Exp	45.5	64.3	61.9	64.4	59.0
+SC (WebShortcutter)	54.5	76.2	78.6	73.3	70.5
Avg. Steps (Finished Trajectories)					
Base	10.48	7.78	10.77	7.21	9.09
+Exp	11.14	8.11	10.60	8.25	9.38
+SC (WebShortcutter)	7.39	6.88	4.12	7.89	6.47
Avg. Steps (Successful Trajectories)					
Base	10.84	7.78	10.67	6.96	8.96
+Exp	11.20	8.11	10.35	7.86	9.22
+SC (WebShortcutter)	7.17	6.88	3.91	7.88	6.40

Table 2: Per-website and overall results. +Exp: Base Agent augmented with mistake notebook $\mathcal{E}_w$. +SC (WebShortcutter): Base Agent augmented with shortcut tools synthesized from notebook-improved trajectories $\mathcal{D}_{\text{exp}}$. A single shortcut invocation counts as one step despite compressing multiple low-level actions.

into executable control flow (e.g., the calendar-paging loop in `TOOL_booking_search_setup`) rather than retained as natural-language reminders.

Trajectory compression. +SC reduces overall average trajectory length by 28.8% for finished trajectories (9.09→6.47) and 28.6% for successful ones (8.96→6.40). The compression is most dramatic on Google Flights (10.77→4.12 finished steps), where the search shortcut replaces the entire form-filling sequence. Shorter trajectories reduce the cumulative risk of hallucination and action mis-selection that grows with extended reasoning chains. Meanwhile, +Exp slightly *increases* trajectory length (9.38 steps), as the notebook encourages more careful exploration rather than premature termination. On Allrecipes, +SC steps (7.89) are slightly higher than Base (7.21) because shortcuts enable more correct completions on complex tasks requiring additional post-search steps, which is offset by the substantially higher success rate.

4.3 Analysis

Shortcut invocation. Shortcuts are invoked in 91.9% of trajectories (100% on Google Flights), confirming reliable recognition of invocation opportunities (Appendix A).

Individual shortcut contribution. A leave-one-out ablation on Booking (Table 3) shows that `TOOL_booking_search_setup` contributes the most (−9.0% when removed, nearly reverting to Base), as it handles the most error-prone phase: pop-up dismissal, destination entry with retry, and calendar date selection. The filter shortcut contributes a smaller but positive −2.2%.

Configuration	Task Success Rate
Base (no shortcuts)	43.2
+SC (both shortcuts)	54.5
w/o search setup	45.5
w/o apply filters	52.3

Table 3: Leave-one-out ablation on Booking.

Practical designs. Three common small designs make shortcuts more reliable. First, stateless entry: before each small step, the shortcut re-fetches the AX Tree instead of assuming the page hasn't changed. Second, bounded retry: it retries a limited number of times on failure to avoid infinite loops. Third, reactive delegation: when faced with obstacles like pop-ups, the shortcut may call other shortcuts instead of trying to solve everything itself.

5 Conclusion

We presented WebShortcutter, a framework that enhances multimodal web agents through failure memory and reusable shortcut synthesis. By distilling website-specific failure patterns into mistake notebooks and abstracting recurring action sequences into reusable shortcut tools, WebShortcutter enables structured experience accumulation and reuse across tasks. Our cascaded pipeline—where failure memory first improves trajectories, and shortcuts are then synthesized from the corrected behaviors—yields substantial gains in both success rate and trajectory efficiency. These results suggest that systematically capturing and reusing environment-level experience is a promising paradigm for building more capable open-world web agents.

Limitations

Our framework has several limitations. First, shortcuts are website-specific and do not transfer across websites; developing cross-website shortcut generalization is an important direction. Second, while shortcut drafting is automated by the Expert Agent, verification and debugging still require human effort (testing on real tasks and providing feedback), which limits scalability to many websites. Third, our evaluation covers four websites; expanding to more diverse categories (e.g., e-commerce, government portals) would strengthen generalizability. Fourth, shortcuts rely on accessibility-tree-based element identification, which may break under major UI redesigns, requiring periodic re-synthesis. Fifth, the mistake notebook's effectiveness depends on lesson quality—incorrect or over-generalized entries can degrade performance, making curation non-trivial. Finally, a promising direction is to more tightly integrate the two mechanisms, for instance by leveraging shortcut-executed trajectories to iteratively refine both lessons and skills.

Ethics Statement

Our experiments use publicly accessible websites via standard browsers without collecting personal data. All interactions are automated sessions without real user accounts. Shortcut tools interact with websites as a normal user would, without exploiting vulnerabilities. We acknowledge that large-scale deployment of web agents could burden web services; our experiments involve limited queries per website. The use of proprietary LLMs (GPT-5.1, Claude Opus 4.6) follows their respective providers' terms of service.

References

Anthropic. 2026. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

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

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*.

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

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. 2025. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.

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

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

OpenAI. 2025. Gpt-5.1: A smarter, more conversational chatgpt. https://openai.com/index/gpt-5-1/.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R Fung, Hao Peng, and Heng Ji. 2024. Craft: Customizing llms by creating and retrieving from specialized toolsets. *arXiv preprint arXiv:2309.17428*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Website	# Shortcuts	Invoke Rate
Booking	2	93.18%
Coursera	2	95.24%
Google Flights	1	100.00%
Allrecipes	2	80.00%
Total	7	91.91%

Table 4: Shortcut invocation statistics. Invoke Rate is the fraction of trajectories where at least one shortcut is invoked.

A Shortcut Invocation Statistics

Google Flights achieves 100% invocation because every task requires the structured search form. Allrecipes has a lower rate (80%) because some queries only require reading top-level page information without scrolling to specific sections.

B Shortcut Tool Examples

Table 5 summarizes representative shortcut tools. Figure 2 shows a full implementation, demonstrating the stateless entry, bounded retry, and inter-shortcut composability patterns described in the main text.

Booking Shortcuts
`TOOL_booking_search_setup`: Dismisses sign-in pop-ups, enters the destination with retry logic, pages the calendar to locate check-in/check-out dates, and submits the search.
`TOOL_booking_apply_filters`: Iterates over filter names, expands collapsed sections via "Show all," and applies each filter with bounded scrolling.
Google Flights Shortcut
`TOOL_search_google_flights`: Sets trip type and seating class, fills origin/destination/dates, and submits the search form.

Table 5: Representative shortcut tools with their functionality.

C Trajectory Comparison

Figure 3 illustrates a representative example.

D Prompts

D.1 Failure Analysis Prompt

The following prompt is provided to the Expert Agent (Claude Opus 4.6) along with the full trajectory (including screenshots) of a failed task. The model is asked to produce a structured JSON object containing the failure reason and a generalizable lesson.

> I have provided you with the interaction records of the webagent and the {website} website. The trajectory contains the TASK and for each step, the "Thought, Action, and the corresponding Web Element (if any)." The specific content in Web Element is derived from the accessibility tree.
> Please note that this was a failed interaction because the agent did not complete the task assigned by the user. Based on the trajectory and all the screenshots, summarize the main issue in this interaction in one sentence. Please use specific language, but avoid referencing specific webpage element IDs.
> Some example outputs: {Example outputs}
> Based on the above web interaction records, please briefly summarize the main reasons for failure, and then provide a general lesson. Notice: Maintain generality in language so that the lesson is not only applicable to the current TASK.
> Output format: {"Reason": "xxx", "Lesson": "xxx"}

D.2 Experience Summarization Prompt

After individual failure analyses are collected for all failed trajectories on a given website, the following prompt is used to consolidate them into a concise, numbered mistake notebook.

> The following is a summary of experiences gained by {website} during Agent navigation. Please summarize this summary, keeping the language as concise as possible, removing redundancy, and using a numbered list format (1, 2, 3...). This will also make it easier to add unique information later.

D.3 Shortcut Synthesis Prompt

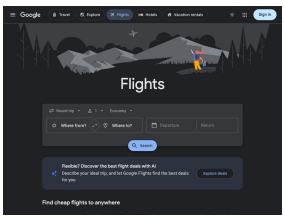
The following prompt (shown in abbreviated form) is provided to the Expert Agent along with several successful trajectories from the same website. The model identifies recurring multi-step patterns and drafts a Selenium Python function for each.

> **Shortcut Synthesis Prompt (Abbreviated)**
>
> You are an assistant skilled at summarizing web interaction experiences. You will be provided with 2 web interaction trajectories from a {website} website.
> Your task is to extract a Shortcut that consolidates multiple steps so that when encountering a similar task in the future, the Shortcut can be used to operate more efficiently.
> Requirements: A Shortcut should consist of multiple steps that together accomplish a small sub-task. It should not be overly complex and generally contain no more than 10 steps. The Shortcut should be encapsulated as a Selenium Python function named using the format: TOOL_{shortcut_name}.
> [Full implementation details including API functions for accessibility tree retrieval, click, type, and scroll actions.]

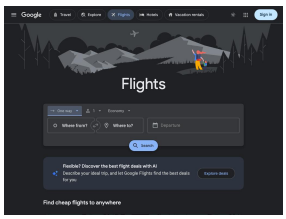
```python
def TOOL_allrecipes_scroll_to_section(driver_task, args, section_name):
    max_scrolls = 15

    # First, Scroll to Top
    driver_task.execute_script("window.scrollTo(0, 0);")
    time.sleep(1)

    for _ in range(max_scrolls):
        ac_tree, obs_info = get_webarena_accessibility_tree(driver_task)

        # Check pop_up
        if _check_and_close_ad(ac_tree, driver_task):
            # close accessibility tree
            ac_tree, obs_info = get_webarena_accessibility_tree(driver_task)

        # Find ele from AX tree
        found = False

        if section_name.lower() == "prep time":
            # "Prep Time:" StaticText, not heading
            pattern = r"\[(\d+)\]\s+StaticText\s+'Prep Time:"
            match = re.search(pattern, ac_tree)
            if match:
                found = True
                target_id = int(match.group(1))
        else:
            # "Ingredients", "Directions", "Nutrition Facts", "Reviews"
            pattern = r"\[(\d+)\]\s+heading\s+'" + re.escape(section_name)
            match = re.search(pattern, ac_tree)
            if match:
                found = True
                target_id = int(match.group(1))

        if found:
            # Use JavaScript scroll to ele
            web_ele = get_web_ele_from_ac_tree_id(
                obs_info, target_id, driver_task, args
            )
            driver_task.execute_script(
                "var rect = arguments[0].getBoundingClientRect();"
                "var absTop = rect.top + window.pageYOffset;"
                "window.scrollTo({top: Math.max(absTop - 200, 0), behavior: 'smooth'});",
                web_ele,
            )
            time.sleep(1)
            return True

        # No target found. Scroll down.
        exec_action_scroll_shortcut("down", driver_task, args)
        time.sleep(0.3)

    # If not found, scroll back to the top of the page.
    driver_task.execute_script("window.scrollTo(0, 0);")
    time.sleep(1)
    return False
```

Figure 2: Implementation of `TOOL_allrecipes_scroll_to_section`. The function scrolls to a target section while dismissing advertisement overlays via inter-shortcut delegation.

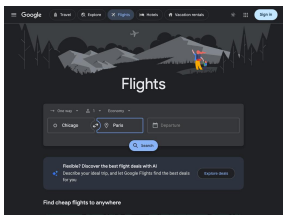
Base Agent Trajectory:



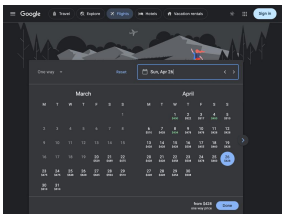
Init Page



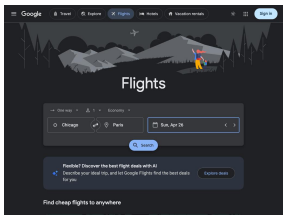
Step 1 - 2, Select Trip Type: One Way



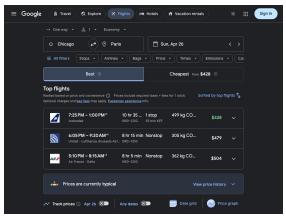
Step 3 - 4, Select City: Chicago and Paris



Step 5 - 6, Select Date: Apr 26, 2026

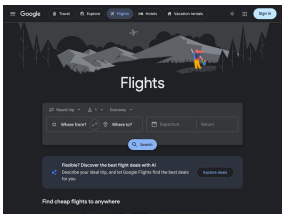


Step 7, Confirm Date

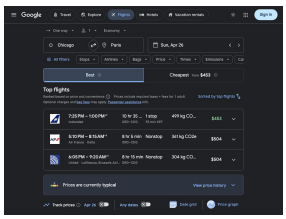


Step 8, Click Search Button and Find ANSWER

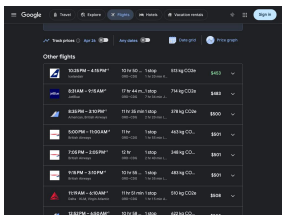
Agent Trajectory, +SC (WebShortcutter) :



Init Page



Step 1, Exec TOOL_search_google_flights



Step 2 - 3, Find ANSWER from Result Page

Figure 3: Trajectory comparison for a Google Flights task: *Show me the list of one-way flights today from Chicago to Paris.* The Base agent takes 8 steps but fails to display complete results. WebShortcutter completes the task in 3 steps via the search shortcut.