WEBTRAVELER: A Multi-Agent Framework for Continuous Exploration in Website Traversal

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly deployed as web-facing agents that interact with live websites to answer complex questions. Many such questions require multi-step, in-site navigation and cross-page evidence aggregation, a regime referred to as web traversal. Existing LLM-based web agents, including recent explore–critic architectures, typically store interaction history as an ever-growing unstructured text trace and mix high-level planning with low-level page actions. This makes long-horizon behavior brittle, obscures the source of evidence, and often leads to redundant browsing or premature stopping. To mitigate these issues, we propose WEBTRAVELER, a hierarchical multi-agent framework that separates high-level information-seeking planning from local web exploration and maintains a structured trajectory memory that compresses interaction steps into decision-relevant signals. This design allows the planner to reason over a compact, explicitly organized history, improving stability, credit assignment and cross-page evidence aggregation on deep trajectories. To obtain scalable supervision for traversal roles, we further propose a path-guided data synthesis method, which uses executable navigation paths as anchors to synthesize trajectories and applies rewriting and logical-consistency checks to ensure verifiable evidence chains. Experimental results on WebWalkerQA show that WEBTRAVELER consistently achieves higher traversal success than previous agents across various LLM scales, with path-guided supervision bringing additional gains, especially on long-horizon tasks and in settings with larger token budgets. Codes and data collection websites will be opensourced.

1 Introduction

Large language models (LLMs) are increasingly used as web-facing agents that acquire up-to-date information by interacting with live websites rather than relying solely on parametric knowledge (Yang et al., 2025c; Ning et al., 2025). A dominant paradigm today is to answer questions via web search (Nakano et al., 2022; Jin et al., 2025; Li et al., 2025c) and retrieval-augmented generation (RAG) (Lewis et al., 2020), where the model consumes a small set of retrieved passages or snippets. While effective for factoids, this paradigm often breaks down when the required information is not well exposed to keyword retrieval or is distributed across multiple pages within the same site. In practice, authoritative content is frequently organized behind navigational structures such as category trees, index pages and multi-level documentation; crucial details may only be reachable by following a specific sequence of in-site links. Answering such questions, therefore, requires multi-step, in-site navigation and cross-page evidence aggregation, together with a traceable provenance that records where each piece of evidence is obtained.

These requirements have motivated the development of *web traversal* (Wu et al., 2025b), in which an agent is given a natural language question and an entry page of a target website, and must obtain sufficient supporting evidence by interacting with that site alone. The agent is constrained to follow in-site links and reason over the navigation structure of a single site, deciding which page to visit next and when the collected evidence is adequate to answer the given question, even when relevant content lies several clicks away or is distributed across multiple internal pages. This makes web traversal a long-horizon decision-making problem under partial observability, where robust cross-page evidence aggregation is essential rather than an optional post-processing step.

Within this setting, a line of work has studied LLM-based web agents for open-domain browsing and web navigation (He et al., 2024; Wu et al., 2025b). For web traversal in particular, WebWalker (Wu et al., 2025b) adopts a ReAct-style

explore–critic architecture (Yao et al., 2023), where an explorer runs a thought–action–observation loop and a critic maintains a textual memory and decides when to answer. However, most existing designs represent the interaction history as an ever-growing, unstructured text trace and couple high-level planning with low-level page interactions in a single context, which makes long-horizon behavior fragile: relevant evidence is easily diluted or dropped, and navigation errors are hard to diagnose and remediate as trajectories grow longer. These limitations call for architectures that factor traversal into clearer roles and maintain a compact, structured summary of past interactions, so that decisions remain reliable even on deep trajectories and under realistic compute budgets.

To address these limitations, we propose WEBTRAVELER, a multi-agent framework that decomposes in-site web traversal into hierarchical planning and local exploration. WEBTRAVELER separates a high-level Task Planner, which maintains an abstract view of the exploration state and proposes sub-goals, from a web exploration loop that executes these sub-goals on the website. Within this loop, a Perceiver extracts task-relevant cues from the current page and an Explorer chooses navigation actions and issues tool calls. At each interaction step, the exploration loop updates a structured trajectory memory that records decision-relevant signals from the interaction history, including actions taken, key observations and supporting evidence. This compact memory, rather than an ever-growing raw history, supports reasoning in subsequent exploration steps, thereby reducing the context length and enabling more stable long-horizon behavior, with fewer redundant revisits and stronger recovery from early mistakes.

Beyond the architecture, we introduce path-guided data synthesis to obtain scalable supervision for the exploration roles, particularly when using smaller base models. The pipeline first leverages website structural priors to select target pages with deterministic, executable in-site paths and constructs question–answer pairs grounded in these pages. It then uses the paths as anchors for trajectory collection. Finally, prompt and output rewriting, together with logical-consistency checks, filters inconsistent or unverifiable trajectories, yielding training data with coherent reasoning and verifiable evidence chains aligned with the traversal objective. Experimental results on WebWalkerQA (Wu et al., 2025b) show that the proposed WEBTRAVELER achieves the best performances across LLMs of varying scales. With the introduced supervision, performances further improve and exhibit more stability on tasks with longer navigation trajectories to the answer page and under larger token budgets.

- We present WEBTRAVELER, a multi-agent framework that decomposes in-site web traversal into high-level planning and local exploration, and stabilizes continuous navigation and information collection through a structured trajectory memory.
- We introduce path-guided data synthesis, which uses executable navigation paths to guide trajectory collection and applies prompt rewriting and logical-consistency checks to construct high-quality supervised data aligned with traversal objectives.
- We evaluate WEBTRAVELER on WebWalkerQA with backbone LLMs of varying scales, showing strong zero-shot traversal performance and further improvements from the proposed data synthesis pipeline over competitive baselines.

2 Related Work

LLM-based web agents have been widely studied for open-domain browsing and search-augmented question answering (Yao et al., 2022; Deng et al., 2023; Zhou et al., 2024; Xue et al., 2025; Mialon et al., 2023; Wei et al., 2025; Zhou et al., 2025). While effective for browser-using and broad information access, these settings often operate on top-ranked pages or snippet-level evidence, and may miss authoritative content that is only reachable via deeper in-site navigation. Closer to our setting, WebWalker (Wu et al., 2025b) formalizes web traversal and proposes an explore–critic agent built on the ReAct paradigm (Yao et al., 2023), but it still records history as an unstructured growing trace and couples high-level decisions with low-level actions in a single context. In contrast, WEBTRAVELER explicitly separates planning from local exploration and maintains a structured trajectory memory to improve stability on deep, in-site trajectories. On the agent training data side, previous datasets provide answer-level supervision or limited human demonstrations in browser environments (Shi et al., 2017; Yang et al., 2018; Welbl et al., 2018; Kwiatkowski et al., 2019; Deng et al.,

2023; Zheng et al., 2024; Pan et al., 2024). Some work focuses on synthetic tool-using data with strong teachers (Qin et al., 2023; Tang et al., 2023), yet the resulting data are not necessarily grounded in real website settings. Recent efforts further scale web agent supervision by synthesising executable multi-step search trajectories on the web (Liu et al., 2025; Wu et al., 2025a; Li et al., 2025b,d; Song et al., 2025) or by generating interaction demonstrations from web tutorials with automatic execution and verification (Xu et al., 2025; Pahuja et al., 2025). In contrast, our data synthesis method anchors trajectory generation on deterministic paths and applies rewriting and consistency checks to produce path-level supervision with explicit evidence chains tailored to web traversal.

3 Method

We begin by formalising the web traversal task, then present WEBTRAVELER, a hierarchical multi-agent framework with structured trajectory memory for long-horizon navigation, and finally introduce a path-guided data synthesis pipeline that uses executable navigation paths to generate and filter high-quality training trajectories.

3.1 Problem Formulation

We formalise web traversal as follows. Given a natural language query q and an initial webpage (*i.e.*, the entry page of a target website), an agent interacts with the website for multiple steps to collect evidence and produce a final answer y. At step t, the agent receives an observation o_t and executes an action a_t that transitions the environment to a new webpage. Following the environment construction in (Wu et al., 2025b), we define the observation as $o_t = (p_t, l_t)$, where p_t is the page content in markdown format and $l_t = \{button_i\}_{i=1}^{K}$ is a set of interactable elements extracted by parsing the HTML. Each $button_i$ is associated with a target URL of a webpage, and the action a_t is a navigation decision that selects one button to explore.

In existing web traversal methods, a common paradigm models decision-making with a monolithic policy that conditions on the raw interaction history:

$$a_t \sim \pi_\theta(\cdot \mid q,\ o_t,\ \mathcal{H}^{\text{raw}}_{t-1}), \tag{1}$$

where $\mathcal{H}^{\text{raw}}_{t-1} = \langle o_0, a_1, \ldots, o_{t-1}, a_{t-1} \rangle$ denotes the uncompressed trajectory and π_θ is a policy model (*e.g.*, an LLM-based agent) that defines a conditional distribution over actions. When page observations are directly concatenated into the prompt, the context length grows approximately linearly with the trajectory length:

$$C(\mathcal{H}^{\text{raw}}_t) \approx t \cdot \bar{L}_{\text{obs}}, \tag{2}$$

where $\bar{L}_{\text{obs}}$ is the average token length of an observation. As t increases, the model input approaches the context window limit, which amplifies noise in the trajectory and accumulates errors over steps, often resulting in repetitive browsing or premature stopping in continuous web traversal.

3.2 WebTraveler

Intuitively, human information seeking operates on two distinct levels: maintaining high-level, dynamically updated intents, and executing low-level webpage exploration strategies guided by those intents. Collapsing these decisions into a single model imposes an excessive cognitive burden. Motivated by this, we propose WEBTRAVELER, a hierarchical multi-agent framework that explicitly decouples the web traversal process into global planning and web exploration. At the planning level, a Task Planner maintains the accumulated evidence and performs a sufficiency check to either terminate with a final answer or generate the next information-seeking subgoals. At the execution level, a Web Exploration Loop phase executes the sub-goal through webpage interactions, where an Explorer navigates and reflects on outcomes, while a Perceiver extracts goal-relevant evidence and compresses each observation into compact, decision-relevant signals. The extracted evidence is aggregated into a shared global evidence memory and fed back to the planner to perform the evidence sufficiency check. Moreover, rather than relying on an ever-growing raw interaction history, we maintain a structured trajectory memory that uses step-wise compressed signals to stabilize continuous exploration.

3.2.1 Global Planning

In web traversal, agents often suffer from aimless browsing or premature termination due to the lack of global directives. To address this, we propose a Task Planner to maintain the exploration state through an information-centric global view. Specifically, acting as a plan-level controller, it focuses on the semantic gap between the user query q and the accumulated evidence $\mathcal{E}$.

At the k-th planning rounds, the Task Planner π_{plan} performs a sufficiency check to dynamically

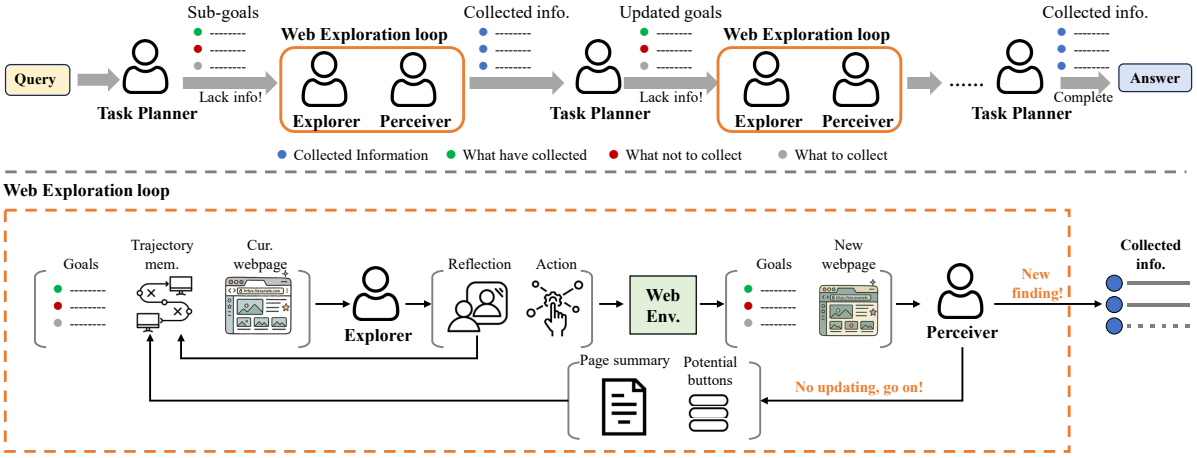

Figure 1: Overview of WEBTRAVELER. The Task Planner maintains global progress via evidence sufficiency checking and dispatches sub-goals. The Web Exploration Loop executes sub-goals with step-wise compression: the Explorer produces navigation actions together with lightweight reflections conditioned on the sub-goal and structured trajectory memory, while the Perceiver compresses page observations, selects potential interactive buttons, and extracts relevant evidence. By encoding every interaction into structured trajectory memory, WEBTRAVELER alleviates context growth and enables robust, continuous web traversal over long horizons.

adjust the information seeking goals g_k and task status σ_k:

$$g_k, \sigma_k \sim \pi_{\text{plan}}(q, \mathcal{E}_{k-1}). \tag{3}$$

$\sigma_k \in \{\text{COMP}, \text{CONT}\}$ denotes the decision status. If $\sigma_k = \text{COMP}$, the planner judges that $\mathcal{E}_{k-1}$ is sufficient to answer q, terminates exploration and triggers final answer generation. Otherwise, if $\sigma_k = \text{CONT}$, the planner identifies the missing evidence and updates the sub-goal g_k to guide the subsequent exploration loop.

3.2.2 The Web Exploration Loop

As the agent interacts with a website over many steps, coupling action execution and page perception within a single model substantially increases the contextual load on the backbone LLM and exacerbates step-wise error accumulation. Accordingly, we decompose web exploration into two roles: the Explorer and the Perceiver. The Explorer is responsible for executing navigation actions and performing step-wise reflection, while the Perceiver summarises the current page, extracts evidence relevant to the current sub-goal, and produces compact, decision-relevant signals for memory updates. This separation reduces redundant context, improves the robustness of continuous exploration, and supports stable cross-page evidence aggregation.

Execution & Reflection (Explorer). Web exploration is error-prone: a single failed interaction can mislead subsequent decisions and compound over steps. To mitigate this, we design the Explorer to perform a step-wise reflection on the previous interaction and adapt the next action accordingly.

Specifically, at step t, Explorer performs reflection and decision-making by integrating trajectory memory $\mathcal{H}_{t-2}$, previous interactions (o_{t-1}, a_{t-1}), and current webpage observation o_t:

$$a_t, r_t \sim \pi_{\text{exp}}(\cdot \mid g_k,\ o_t,\ o_{t-1}, a_{t-1},\ \mathcal{H}_{t-2}). \tag{4}$$

Here, r_t is a brief reflection sentence that diagnoses whether the previous interaction succeeded and whether it advanced the sub-goal g_k. The action a_t is then executed to interact with the web environment, navigating to a new webpage to drive the exploration forward. We explicitly separate the most recent interaction from the compressed trajectory memory to facilitate fine-grained reflection.

Perception & Extraction (Perceiver). After executing a, the environment returns a new webpage observation o. Since the information required to answer the query is often dispersed across pages, we extract relevant evidence from the current page under the guidance of g_k and accumulate it into a global memory that is fed back to the Task Planner, enabling sufficiency checks and sub-goal updates. Moreover, to support continual exploration within a finite context window and to stabilize decision-making, we also compress the raw observation and prune interactable elements to reduce the action space. Accordingly, we introduce a Perceiver that standardises noisy and weakly structured webpage

observations into compact, semantic components, supporting both evidence harvesting and efficient navigation.

Specifically, the Perceiver ϕ_{per} produces three outputs:

$$(d_t, \mathcal{C}_t, e_t) = \phi_{\text{per}}(g_k, o_t). \tag{5}$$

Here, we use k to index planning rounds and t to index environment interaction steps. e_t denotes the evidence relevant to g_k; $\mathcal{C}_t$ is the filtered candidate buttons so that it prunes the action space, and d_t is a compact semantic digest of o_t. $(d_t, \mathcal{C}_t)$ will be used to build the exploration trajectory. If $e_t \neq \emptyset$, we immediately add it to the global evidence memory (i.e., $\mathcal{E} \leftarrow \mathcal{E} \cup \{e_t\}$) and return it to the Task Planner to perform a sufficiency check. Otherwise, the web exploration will step into the next loop.

Trajectory Memory. To support continual exploration under a finite context window, we maintain a structured trajectory memory that records only compact, decision-relevant signals to replace the full page observations. At each exploration step t, the trajectory is incrementally updated by integrating the compact digest from the Perceiver, filtered action candidates, the executed action from the Explorer, and its corresponding reflection:

$$\mathcal{H}_t = \mathcal{H}_{t-1} \cup \{(d_t, \mathcal{C}_t, a_t, r_t)\}. \tag{6}$$

This design replaces raw observations with compact semantic representations, thereby bounding the growth of the interaction history and allowing the agent to sustain exploration trajectories under a limited context window. Moreover, it preserves the minimal causal traces required for effective decision-making, i.e., constraining the feasible action space, disambiguating explored paths, and providing progress feedback. These ensure that the Explorer can avoid stagnation and repeated ineffective interactions in subsequent steps.

3.3 Path-guided Data Synthesis

The core objective of the proposed WEBTRAVELER is to stabilize continuous website exploration and cross-page evidence aggregation. However, in real-world deployment, cost constraints make smaller LLMs more practical, yet their limited capability in web interaction and long-horizon exploration often becomes the primary bottleneck. This raises a key question: can WEBTRAVELER keep improving under scalable supervision, thereby providing better returns in sustained exploration settings? However, building the training data is non-trivial. Manual annotation is high quality, but is hard to scale. In contrast, unconstrained automatic synthesis can violate the structural constraints of real web environments, producing non-executable actions or unverifiable page transitions. To address this, we propose a path-guided data synthesis method, which utilises real, deterministic navigation paths to guide the agent to synthesize high-quality interaction trajectories in the real environment. The pipeline consists of three stages:

At first, we leverage the structural priors of websites to mine a set of target pages with deterministic executable navigation paths. Based on these target pages, we use an LLM to automatically construct QA tasks (the answer is from one or two pages).

Then, we explicitly provide the executable path as a prompt signal to guide the agent in completing the constructed tasks. The agent still makes decisions conditioned on the current sub-goal and page observation, while the path serves as a verifiable reference for feasible next-step transitions. This ensures that the resulting interaction sequence is executable in the real environment and can reliably reach the evidence pages. We then collect complete trajectories, including observations, actions, reflections, and extracted evidence.

Finally, to prevent the training data from degenerating into rote path-following and to improve annotation consistency and transferability, we apply two post-processing steps to the collected trajectories: (1) Rewriting: rewriting all explicit path hints and traces, while retaining the reasoning and decision-making that the model should exhibit under autonomous exploration; (2) Logical-consistency checks: requiring that the final answer be supported by evidence collected in the trajectory, and that the trajectory terminates on a page that justifies the answer.

After obtaining the training trajectories, we serialize them into role-specific SFT data for the Explorer and Perceiver to train the Web Exploration Loop, thereby enabling and improving WEBTRAVELER's continued gains under scalable supervision.

4 Experiments

4.1 Experimental Settings.

Datasets. We evaluate our method on the WebWalkerQA benchmark (Wu et al., 2025b), which measures web traversal capability by requiring an

Model	Framework	s-easy	s-med	s-hard	m-easy	m-med	m-hard	All
Qwen2.5-7B	ReAct	**42.86**	18.31	6.06	25.00	11.11	3.33	16.67
	WebWalker	39.29	**22.54**	**9.09**	8.33	**12.96**	3.33	16.67
	WEBTRAVELER	39.29	19.72	6.06	**33.33**	11.11	**10.00**	**18.33**
Qwen2.5-32B	ReAct	53.57	29.58	**15.15**	29.17	**16.67**	10.00	25.00
	WebWalker	50.00	33.80	12.12	25.00	14.81	**13.33**	25.00
	WEBTRAVELER	**64.29**	**40.85**	**15.15**	**33.33**	**16.67**	**13.33**	**30.42**
Qwen3-8B	ReAct	46.43	26.76	12.12	20.83	18.52	3.33	21.67
	WebWalker	35.71	23.94	9.09	12.50	11.11	**6.67**	17.08
	WEBTRAVELER	**53.57**	**29.58**	**15.15**	**41.67**	**24.07**	**6.67**	**27.50**
Qwen3-32B	ReAct	57.14	36.62	9.09	16.67	18.52	13.33	26.25
	WebWalker	**67.86**	35.21	**15.15**	20.83	9.26	10.00	25.83
	WEBTRAVELER	64.29	**39.44**	**15.15**	**29.17**	**31.48**	**23.33**	**34.16**

Table 1: Task success rate (%) on the WebWalkerQA benchmark without fine-tuning. Within each model block, the best results among all frameworks are in bold. s- and m- mean single-page and multi-page, respectively.

agent to navigate within a website to answer a query. The benchmark contains both single-page and multi-page settings, and stratifies tasks into easy, medium and hard levels according to navigation depth. In practice, however, WebWalkerQA is subject to website drift: some annotated answer pages are no longer accessible and some recorded navigation paths have changed, which can make individual instances non-executable or substantially alter their difficulty. Running evaluation naively on the full test set therefore mixes well-defined and ill-defined instances and undermines fair comparison between systems. To obtain a stable and reproducible evaluation protocol, we construct a curated subset of 240 instances by manually auditing the original test set and retaining only those whose answer pages are accessible and whose annotated paths still lead to the correct evidence pages. More details and statistics of the selected subset are provided in Appendix A.1.

Implementation Details. For evaluation, all methods are limited to at most 15 interaction steps and are executed under the same network environment to ensure consistent conditions. Following prior work (Wu et al., 2025b), we use task success rate as the primary metric and report results for each task category as well as overall. To ensure a fair comparison, we also adopt the LLM-as-a-Judge protocol to assess agent answers (Li et al., 2025a). For training data synthesis, to promote generality and avoid information leakage, we construct data only from publicly accessible websites with no overlap with WebWalkerQA. Additional implementation details are provided in Appendix A.2.

Baselines. We compare against two representative methods, ReAct (Yao et al., 2023) and WebWalker (Wu et al., 2025b), instantiated with backbone LLMs of varying scales, including Qwen2.5-7B, Qwen2.5-32B (Yang et al., 2025b), Qwen3-8B and Qwen3-32B (Yang et al., 2025a). ReAct is a general-purpose agent paradigm that interleaves reasoning and actions for multi-step web interaction, while WebWalker specialises in web traversal with an Explorer for navigation and evidence collection and a Critic that decides whether to continue exploring.

4.2 Main Results

In-context Learning Performance. Table 1 reports the results on the WebWalkerQA benchmark. Across all evaluated LLMs, the proposed WEBTRAVELER consistently achieves the best overall performance among the compared frameworks, indicating strong robustness to base models. Moreover, the advantages increase with stronger LLMs. In particular, with the Qwen3-32B as base model, WEBTRAVELER achieves an overall success rate of 34.16%, outperforming ReAct (26.25%) and WebWalker (25.83%). Besides, the results show that WEBTRAVELER outperforms the baselines on almost all multi-page tasks. For example, under the Qwen3-8B setting, WEBTRAVELER significantly outperforms both baselines on multi-page tasks across all three difficulty levels, suggesting that our method enables more stable web exploration.

Effect of Path-guided Synthesized Data. Table 2 reports the results after fine-tuning on trajectories synthesized by our proposed path-guided

Model	Easy	Medium	Hard	All
Qwen2.5-7B	36.54	16.00	7.94	18.33
Qwen2.5-7B*	**42.31**	**23.20**	**12.70**	**24.58**
Qwen3-8B	48.08	27.20	11.11	27.50
Qwen3-8B*	**59.62**	**32.00**	**19.05**	**34.58**
Qwen3-32B	48.67	36.00	19.05	34.16

Table 2: Results after fine-tuning on the synthesized data. Here $*$ indicates fine-tuning.

data synthesis method. The synthesized data delivers stable and consistent gains across base models. On Qwen2.5-7B, fine-tuning increases the overall accuracy from 18.33% to 24.58% (+6.25%). On Qwen3-8B, it improves from 27.50% to 34.58% (+7.08%). Notably, performance on hard tasks increases markedly for both models (7.94%→12.70% and 11.11%→19.05%), indicating that the synthesized trajectories are particularly effective at strengthening long-horizon exploration and reasoning. Moreover, the fine-tuned Qwen3-8B achieves performance comparable to the zero-shot Qwen3-32B, suggesting that under constrained model scale, our method provides a scalable and effective way to narrow the gap to larger models.

4.3 Ablation Studies

Architectural Ablations. Table 3 reports ablation results on the architectural design of WEBTRAVELER, using Qwen3-8B as the base model. Removing the Task Planner yields consistent drops across all difficulty levels, reducing overall accuracy from 27.50% to 24.17% and highlighting the importance of global planning in continuous web exploration. Merging the Perceiver and Explorer into a single role also degrades performance, further reducing overall rate to 23.33%. These results suggest that coupling semantic abstraction with action execution within one role undermines exploration effectiveness. In summary, the ablations indicate that both the global planning capability provided by the Task Planner and the role separation in the exploration phase are key structural factors for stabilizing continuous exploration in WEBTRAVELER.

Component-wise Fine-tuning Ablations. Table 4 presents an ablation study on training different components of WEBTRAVELER, using Qwen3-8B as the base model. First, training only the Perceiver improves performance over the baseline, particularly on medium and hard tasks, suggesting that

Method	Easy	Medium	Hard	All
Final design	**48.08**	**27.20**	**11.11**	**27.50**
No Planner	42.31	25.60	6.35	24.17
Merge Per. & Exp.	42.31	23.20	7.94	23.33

Table 3: Ablations on the planner and module design (Qwen3-8B as the base model).

Method	Easy	Medium	Hard	All
Final design	48.08	27.20	11.11	27.50
Only Perceiver	40.38	30.40	12.70	29.17
Only Explorer	40.38	22.40	9.52	22.92
Per. + Exp.	**59.62**	**32.00**	**19.05**	**34.58**

Table 4: Results of training different components (Qwen3-8B as the base model).

better semantic abstraction and evidence consolidation are beneficial for exploration. In contrast, training only the Explorer yields limited gains and even performance degradation on hard tasks, indicating that optimising action selection alone is insufficient without reliable semantic grounding. When both Perceiver and Explorer are jointly trained, the model achieves the best performance across all difficulty levels, improving the overall accuracy from 27.50% to 34.58%. These results demonstrate the complementarity between perception and decision-making and validate the effectiveness of our proposed data synthesis approach.

5 Discussions

Results Analysis by Exploration Horizon. To examine continuous exploration capability, we group WebWalkerQA tasks by the depth of the optimal access path to the answer page and report results for each depth group in Figure 2. Concretely, we stratify tasks into three groups by the navigation depth: depth 1–2, 3–4, and 5–6. On shallow tasks that require only limited navigation and minimal global planning (depth 1–2), all methods perform comparably, with WEBTRAVELER retaining a modest edge. As depth increases, the performance gap between WEBTRAVELER and the baselines becomes more pronounced, with the largest margin observed on the deepest group (depth 5–6). Moreover, the margins also grow when scaling the backbone from 8B to 32B. These results indicate that WEBTRAVELER supports more stable long-horizon web exploration and is better able to capitalise on stronger LLMs.

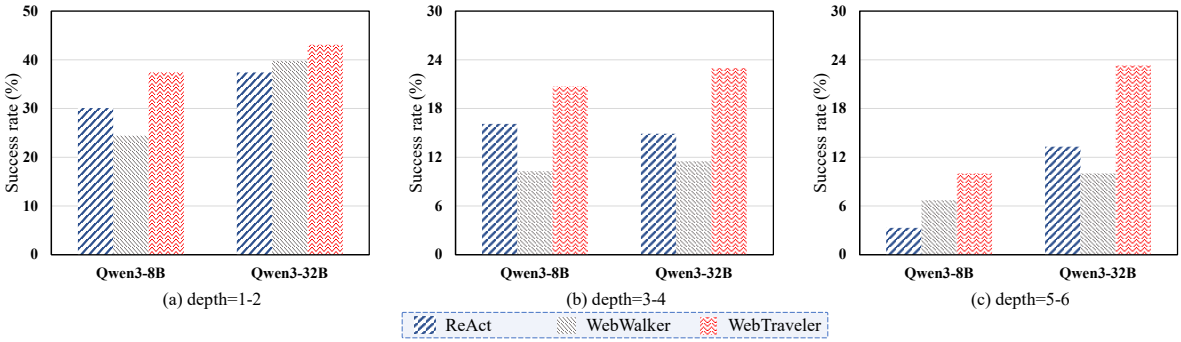

Figure 2: Results across tasks with different answer-page depths.

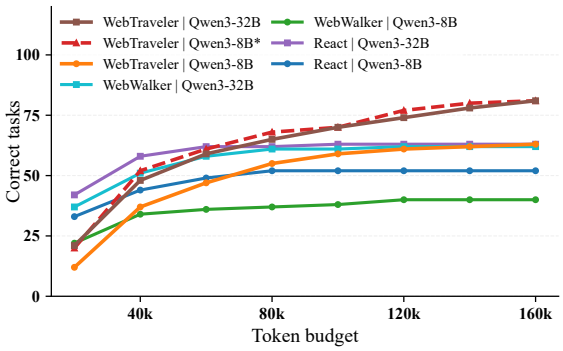

Figure 3: Number of correctly solved tasks under varying token budgets.

Efficiency Analysis under Token Budgets. A key requirement for stable, sustained web exploration is that a finite context window be used to make meaningful exploration progress, rather than being consumed by redundant browsing or an ever-growing interaction history. To directly assess this efficiency, we treat the total number of tokens consumed per task as the budget metric and compare how many instances each method solves correctly under finite token limits. As shown in Figure 3, WEBTRAVELER exhibits sustained gains across a broad range of budgets. As the budget increases, it continues to solve additional tasks and progressively widens the gap to the baselines. Under very small budgets, performance differences are modest, and ReAct often achieves higher coverage earlier. The reason is that outcomes are dominated by startup overhead when the budget is extremely constrained. However, in the moderate-to-large budget regime, WEBTRAVELER continues to accumulate solved tasks, whereas ReAct and WebWalker show clear early saturation. This indicates that for the baselines, simply allocating more tokens does not translate into additional task coverage. In contrast, WEBTRAVELER consistently converts extra budget into higher coverage, reflecting stronger capacity for stable long-horizon exploration. Moreover, the fine-tuned WEBTRAVELER (8B*), trained on our proposed path-guided synthesized trajectories, shows larger and smoother improvements as the token budget increases, indicating further gains in budget utilization for web exploration. These results significantly demonstrate that the proposed WEBTRAVELER scales more reliably with available token budgets, and that path-guided synthesized data further improves budget efficiency for smaller base models.

6 Conclusion

In this paper, we propose WEBTRAVELER, a hierarchical multi-agent framework for stable, continuous web traversal. WEBTRAVELER decomposes traversal into sub-goal planning and local web exploration, and maintains a structured trajectory memory that compresses each interaction step into decision-relevant signals. These designs enable stable navigation and evidence collection under a finite context window. Moreover, to verify whether the framework benefits from scalable supervision, we introduce path-guided data synthesis, which anchors supervision on executable navigation paths to generate high-quality trajectories and filters them via prompt rewriting and logical-consistency verification. Experiments on WebWalkerQA show that WEBTRAVELER achieves the strongest zero-shot performance across LLMs of varying scales, and that fine-tuning on the data synthesized by our proposed method yields consistent improvements. Further analyses demonstrate that these advantages become more pronounced on tasks with deeper answer pages and under larger token budgets, indicating stronger long-horizon scalability of the proposed method.

Limitations

Limited actions and observations. We follow the environment setup of WebWalker, modeling observations as page markdown text plus a set of clickable buttons parsed from HTML. Future work could extend the action space to better match real-world browsing (e.g., in-site search) to improve efficiency, and incorporate more modalities such as screenshots to cover a wider range of question types (e.g., multimodal information seeking).

Potential information loss. To control context growth, our method compresses each page into a compact semantic summary. This design may discard fine-grained evidence or mistakenly prune critical actions.

Higher startup overhead. WEBTRAVELER consists of multiple roles which introduce additional invocation costs and engineering complexity. For shallow questions that require only a few interaction steps, this startup overhead can be relatively more pronounced.

References

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Inf. Process. Syst.*, pages 28091–28114.

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. In *Proc. Conf. Association for Computational Linguistics*, pages 6864–6890.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *Preprint*, arXiv:2503.09516.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Trans. Assoc. Comput. Linguist.*, 7:453–466.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Inf. Process. Syst.*, 33:9459–9474.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, and 1 others. 2025a. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In *Proc. Conf. Empirical Methods in Natural Language Processing*, pages 2757–2791.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025b. Websailor: Navigating super-human reasoning for web agent. https://arxiv.org/abs/2507.02592. ArXiv:2507.02592.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025c. Search-o1: Agentic search-enhanced large reasoning models. *Preprint*, arXiv:2501.05366.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. 2025d. Webthinker: Empowering large reasoning models with deep research capability. *Preprint*, arXiv:2504.21776.

Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, Jiayuan Song, Zhengmao Zhu, Wenhu Chen, Pengyu Zhao, and Junxian He. 2025. Webexplorer: Explore and evolve for training long-horizon web agents. *Preprint*, arXiv:2509.06501.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *Proc. Int. Conf. Learn. Representations*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

Liangbo Ning, Ziran Liang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Wenqi Fan, Xiao-yong Wei, Shanru Lin, Hui Liu, Philip S Yu, and 1 others. 2025. A survey of webagents: Towards next-generation ai agents for web automation with large foundation models. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, pages 6140–6150.

OpenAI. 2025. Introducing gpt-5.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and

Ahmed Hassan. 2025. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings Assoc. Comput. Linguist.*, pages 6300–6323.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. 2024. Webcanvas: Benchmarking web agents in online environments. *Preprint*, arXiv:2406.12373.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *Preprint*, arXiv:2307.16789.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *Proc. Int. Conf. Mach. Learn.*, pages 3135–3144. PMLR.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *Preprint*, arXiv:2503.05592.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *Preprint*, arXiv:2306.05301.

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

Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. 2018. Constructing datasets for multi-hop reading comprehension across documents. *Trans. Assoc. Comput. Linguist.*, 6:287–302.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. Webdancer: Towards autonomous information seeking agency.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. 2025b. WebWalker: Benchmarking LLMs in web traversal. In *Proc. Conf. Association for Computational Linguistics*.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. 2025. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *Proc. Int. Conf. Learn. Representations*.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Qwen: An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2025b. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Yingxuan Yang, Mulei Ma, Yuxuan Huang, Huacan Chai, Chenyu Gong, Haoran Geng, Yuanjian Zhou, Ying Wen, Meng Fang, Muhao Chen, Shangding Gu, Ming Jin, Costas Spanos, Yang Yang, Pieter Abbeel, Dawn Song, Weinan Zhang, and Jun Wang. 2025c. Agentic web: Weaving the next web with ai agents. *Preprint*, arXiv:2507.21206.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proc. Conf. Empirical Methods in Natural Language Processing*, pages 2369–2380.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *Proc. Int. Conf. Learn. Representations*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. In *Proc. Int. Conf. Mach. Learn.*

Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, Yuxin Gu, Sixin Hong, Jing Ren, Jian Chen, Chao Liu, and Yining Hua. 2025. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. *Preprint*, arXiv:2504.19314.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *Proc. Int. Conf. Learn. Representations*.

A Example Appendix

A.1 Benchmark

WebWalkerQA (Wu et al., 2025b) is constructed over live, real-world websites. In our reproduction, we found that a non-trivial fraction of the original tasks has become unstable or ill-posed over time due to website drift and annotation inconsistencies. This instability adds evaluation noise that is largely orthogonal to agent capability, such as broken navigation paths, inaccessible evidence sources, or pages whose content has changed. To obtain a controlled and reproducible evaluation protocol, we manually audited the benchmark. For each task, we verified that the root URL and the annotated navigation path remain executable, and that the labeled answer page is reachable and contains sufficient evidence to support the reference answer (either explicitly stated or derivable via straightforward reasoning). Following this audit, we curate a stable subset of 240 tasks. Dataset statistics for the resulting subset are reported in Figure 5.

Category	#Tasks
single-easy	28
single-medium	71
single-hard	33
multi-easy	24
multi-medium	54
multi-hard	30
ALL	240

Table 5: statistics of the evaluation dataset.

Framework	Model	orig.	filt.
ReAct	Qwen2.5-7B	16.02	16.67
ReAct	Qwen2.5-32B	25.44	25.00
WebWalker	Qwen2.5-7B	19.85	16.67
WebWalker	Qwen2.5-32B	26.02	25.00

Table 6: Overall task success rate (%) on original WebWalkerQA (orig.) and the filtered subset (filt.).

To check whether the filtered subset remains representative of the original benchmark, we compare success rates reported for the full WebWalkerQA in the original paper with our reproduced results on the 240-task subset. We consider two representative baselines, ReAct (Yao et al., 2023) and WebWalker (Wu et al., 2025b), instantiated with Qwen2.5-7B and Qwen2.5-32B as the base LLMs. As shown in Table 6, the relative performance trends are highly consistent between the full benchmark and the subset, suggesting that the subset largely preserves the benchmark's difficulty profile while removing unstable instances. We will release the filtered dataset.

A.2 More Implementation Details

Evaluation. To ensure a fair and consistent comparison, we evaluate all methods using the same automatic judge, GPT-5.1 (OpenAI, 2025). For each example, the judge takes the query, the expected answer, and the agent-generated answer as inputs, and outputs a binary score in $\{0, 1\}$ indicating whether the agent answer is correct. The evaluation prompt is shown in Figure 4.

Training details. We construct the training data using the proposed path-guided synthesis over 60 publicly accessible and compliant websites, thereby minimizing potential risks in the collected data (e.g., personally identifying info or offensive content). For each website, we collect 40 single-page QA trajectories and 20 multi-page QA trajectories, preventing the dataset from being dominated by any single site or interaction pattern. The resulting training set contains $2{,}400$ single-page QA pairs and $1{,}200$ multi-page QA pairs, comprising $10{,}052/6{,}653$ Perceiver messages and $7{,}553/5{,}332$ Explorer messages for single/multi-page tasks, respectively. Notably, whereas WebWalkerQA restricts target websites to four domains (conference, organization, education, and game), we intentionally cover 10 domains (General News, Tech / Developer, Education / Universities, Healthcare / Medicine, Finance / Business, Culture / Arts, E-commerce / Retail, Sports, Entertainment / Media, and Journals / Conferences). As a result, our website set is broader and more balanced in domain coverage, and is not tailored to any specific WebWalkerQA sites. This reduces the risk of benchmark-specific overfitting and better demonstrates the generality and transferability of our synthesized data and training pipeline.

For training, both the Qwen2.5-7B and Qwen3-8B models are trained using two NVIDIA H100 GPUs with two epochs. The maximum context length is set to 16K tokens. All the corresponding codes and data collection websites will be released.

Prompt for Judge

You are a world-class answer consistency verifier. I will provide three inputs: 1. query

(the question) 2. expected_answer (the reference answer) 3. agent_answer (the model-generated answer)
Your task is to strictly judge whether agent_answer correctly answers the query according to expected_answer, and produce:
- Point-by-point analysis (coverage and consistency) - A clear conclusion - Final score (1 = success; 0 = failure)

Principles:
- (General · Query-first-1) This task treats the query as the only source to determine the mandatory set of key points (Q-key points). expected_answer is only a reference for correct wording and granularity for calibration and alignment; you must not introduce new mandatory points beyond the query.
- (General · Query-first-2) Any background information or extra constraints described in the query do not have to be restated in agent_answer; do not give 0 merely because agent_answer does not repeat such background or constraints.

A. Key-point Coverage
- If agent_answer covers all key points in expected_answer (i.e., all major facts/information points), then output 1 (success). Different wording or extra information is allowed, but must not contradict expected_answer.
- If agent_answer omits, is vague about, or incorrectly answers any key point in expected_answer, then output 0 (failure).
- (Practical note) If agent_answer clearly and accurately covers the core facts (defined below) and has no conflict with expected_answer, you may output 1. For non-decisive modifiers (defined below), omission is allowed and should not be penalized.
- (Query alignment) Only information points that directly answer the query are treated as key points. Content in expected_answer that does not directly answer the query and does not change the conclusion is treated as supplementary; missing it should not be penalized.
- Key-point source order: First, extract the minimal necessary points from the query as Q-key points; then use expected_answer to calibrate their definition/granularity/pairing. Only when missing details would change the definition, unit, denominator, scope, or time granularity should they be upgraded into Q-key points.
- If expected_answer contains information beyond the query, default it to S-supplementary points; missing it is not penalized. Only if agent_answer explicitly contradicts an S-point should you output 0.
- Mandatory labeling (Q/S): Before judging, label points from expected_answer into:
Q-key points: facts necessary to directly answer the query, or details whose absence would change definition/unit/denominator/scope/time granularity.
S-supplementary points: all other background/expansion details, as well as all background constraints explicitly stated in the query (they are not core facts).
- Missing handling: scoring is based only on Q-key points. If agent_answer covers all Q-key points with no contradiction, output 1. Missing S-points does not reduce score; only explicit contradiction with any S-point leads to 0.

B. No Subjective Inference or Vague Coverage
- Do not treat "related/similar" information as "covered." Coverage requires agent_answer to explicitly provide an equivalent fact (or clear synonym) to expected_answer.
- If agent_answer uses uncertainty/speculation words (e.g., "maybe," "probably," "possibly," "seems," etc.) regarding any expected key point, treat it as not covered.
- (Definition invariance) If missing or altering a detail would materially change the definition/meaning (e.g., population scope, geography, unit/denominator, statistical definition, time granularity), that detail is automatically a key point. Missing it or contradicting expected_answer yields 0.
- (Background constraints) All background information explicitly stated in the query is not treated as a core fact; even if expected_answer includes it, agent_answer is not required to restate it.
- expected_answer must not expand the query scope: never treat extra details in expected_answer as new Q-key points un-

less missing them would change the definition and thus must be upgraded by the rules above.

C. Date and Pairing Consistency
- If expected_answer specifies date–event (or date–point) pairings, agent_answer must match the same date and event type one by one and explicitly.
- Mentioning other content in the same period without the correct event/announcement type is treated as not covered.

D. Extra Information Rules
- Extra details beyond expected_answer are allowed as long as they do not contradict expected_answer.
- Any contradiction with expected_answer leads to 0.

E. Format and Output Requirements
- You must output a raw JSON object (not a string; do not wrap in code fences; no pre/-post explanatory text).
- JSON fields are fixed and must be complete:
"query"
"expected_answer"
"agent_answer"
"analysis": one paragraph in Chinese with point-by-point, checkable analysis. It must explicitly state whether each Q-key point is covered, and whether any S-point has contradictions.
"score": must be `0` or `1`.
- Do not output anything other than the JSON object.

Core Facts vs. Modifiers (General Definitions)
- Core facts (typically must be explicitly covered):
numbers/ratios/levels; precise time anchors (dates/intervals); main entities (policy/agency/target); action/event types (publish/implement/increase/-support/etc.); directional conclusions (up/down/achieved/not achieved).
- Modifiers (can be omitted unless they change the definition):
population scope restrictions, region/channel/scenario lists, taxonomies, methodology/measurement details, implementation details, background descriptions, adjectival qualifiers.
If omission/change would materially alter definition/unit/denominator/scope/time granularity, upgrade to a key point.

Judgment Procedure
1) Q/S labeling (Query → Expected): extract Q-key points from the query only, then calibrate with expected_answer; upgrade to Q only when necessary to preserve definition/unit/denominator/scope/time granularity.
2) Query alignment check: ensure all Q-key points directly answer the query; keep non-essential expected_answer details as S.
3) Point-by-point verification: check agent_answer explicitly and equivalently covers all Q-key points; uncertainty language does not count as coverage.
4) Supplement consistency: for S-points, only check contradictions; missing is not penalized.
5) Contradictions and vagueness: apply rules for conflicts and vague wording.
6) Conclusion and score: all Q covered with no contradictions → score = 1; otherwise score = 0.

Please strictly follow the query-first principles and the rules above to judge whether agent_answer correctly answers the query. You must output only a JSON object and nothing else.

Figure 4: Prompts for Evaluation.

A.3 Potential Risks

The techniques proposed in this paper aim to enhance the robustness and efficiency of WebAgent exploration, and may therefore amplify the associated capabilities in both benign and malicious settings. Such risks are a commonly accompanying dual-use risk of web agent technologies that operate on live websites via automated in-site navigation and evidence aggregation. Once endowed with more stable long-horizon interaction and more efficient information extraction, the barrier to using them for inappropriate automated collection or circumventing access constraints may be corre-

spondingly lowered.

A.4 Prompts in WEBTRAVELER

The prompts in WEBTRAVELER are shown in Figure 5 (for Task Planner), Figure 6 (for Perceiver) and Figure 7 (for Explorer).

Prompt for Task Planner

You are a world-class Task Planner. You need to decompose the user's task into clear information-gathering subgoals, and at each update of information, determine whether the collected information is sufficient to complete the task.

Goal:
- Record information points that do not need to be collected (info_not_to_collect), as they have been verified or are irrelevant to the task.
- Identify key information points required to continue completing the task (info_to_collect).
- If the collected knowledge (directly or by logical implication) is sufficient to complete the task, set complete = true.

Input:
- Query: "<user's question>"
- Last Info_to_collect: "<previous information-gathering goals>"
- Last Info_not_to_collect: "<previously verified or irrelevant information points>"
- Current Collected_information: "<currently collected information>"

Output format:
<reasoning>
Briefly reason from the following aspects:
- To complete the task, which information has been verified and no longer needs to be collected, which is still missing, and how it can be gathered.
- Whether the currently collected information can answer the user's question.
</reasoning>
{
"info_not_to_collect": ["information items that have been verified or are irrelevant to the task"],
"info_to_collect": ["information items that still need to be collected (not steps; must include missing information required to complete the task; may include inferred conditions and synonym variants)"],
"complete": true or false, output true when the collected information can answer the user's question; otherwise output false. Important: If the user has not specified details, do not over-extend the answer or invent conditions.
}

Principles:
1. Task feasibility assumption — Only consider the task completable when sufficient information is collected; do not abandon the task prematurely.
2. Strict format — Output only the <reasoning></reasoning> block and the JSON object; no extra text.
3. Goal separation — info_to_collect represents information goals, not procedural steps.
4. Constraint completeness — Ensure that info_to_collect covers every explicit constraint not yet confirmed for task completion, and may include derived constraints to enhance decision quality.
5. Synonyms and variants — Common synonyms/aliases/related terms should be merged within a single item using parentheses or slashes; consolidate overlapping goals to keep the list concise.
6. Preserve key items — When generating a new info_to_collect, retain previously important but unverified items.
7. Remove irrelevant items — Move verified or newly proven irrelevant/useless items into info_not_to_collect.
8. Early termination — If the collected knowledge is sufficient to answer the question or complete the task, mark complete = true as soon as possible. Do not over-focus on unspecified details.
9. Termination conditions — For specific questions: terminate once all conditions are met (avoid excessive exploration of overly detailed sub-information). For open-ended questions: terminate once sufficient basic information to answer is collected.
10. Answer focus — There is no need to

provide an overly comprehensive answer beyond the stated conditions; avoid excessive exploration or focusing on user-unstated details.

Figure 5: Prompts for Task Planner in WEBTRAVELER.

Prompt for Perceiver

You are a world-class information extraction agent. Your task is to analyse the given observation content and extract information relevant to the current user query.

Goal:
- You need to produce a brief summary of the current page, i.e., brief_summary.
- You need to determine whether the observation contains new information that is useful for the query, i.e., new_relevant_facts.

Input:
- Query: "<user's question>"
- Info_to_collect: "<current information-gathering goals>"
- Info_not_to_collect: "<current information points that do not need to be collected>"
- Current Collected_information: "<information already collected>"
- Observation: "<current webpage observation content>"

Output format:
<reasoning>
Briefly reason from the following aspects:
- How to extract information from the Observation that is relevant to the Query and Info_to_collect.
- How to avoid collecting information listed in Info_not_to_collect.
- How to avoid re-collecting information already present in Collected_information.
- How to identify all potentially useful clickable buttons while avoiding retrieving too many irrelevant ones.
</reasoning>
{
"brief_summary": "A concise summary of the current page",
"clickable_button_candidates": "A list of up to 5 potentially useful clickable button candidates",
"new_relevant_facts": "Complete sentences describing task-relevant facts, either as a string or a string array"
}

Principles:
1. Strict format — Output only the <reasoning></reasoning> block and the JSON object; no extra text.
2. Memory-oriented — brief_summary should summarize the important and potential information on the page to facilitate recalling this page's usefulness for completing the user's question.
3. Button candidate selection — clickable_button_candidates should include all clickable buttons on the page that have potential to help complete the task. If fewer than five, you may return fewer or even an empty list.
4. Observation priority — new_relevant_facts must have visible support within the current observation; if none, return [].
5. Relevance first — Include only facts that directly (or partially) satisfy info_to_collect.
6. Fact-only — Do not speculate, plan, or hypothesize; avoid vague possibilities.
7. Direct or implied — Facts must be explicitly presented in or logically implied by the observation.
8. Atomic and precise — Each fact must be self-contained and include clear labels/units/timestamps when visible.
9. No repetition — Do not repeat the task, Info_to_collect, or general context. Do not repeat facts already in Collected_information (semantic equivalents count as duplicates), and avoid redundancy within this output.
10. Source awareness — Facts should be tied to visible clues in the observation; include short citations or positions only in <reasoning>, not in the JSON.
11. Source expression — Prefer to use the original wording from the source, with minimal modification; summarisation or

synthesis is allowed only for clarity or completeness.
12. Conflict handling — If the observation contradicts prior knowledge, prioritise the observation and output only facts supported by it (do not restate old facts).

Figure 6: Prompts for Perceiver in WEBTRAVELER.

Prompt for Explorer

You are a world-class browser explorer. Your main responsibility is to design the optimal next-step browser exploration strategy based on the current page state, user task, and historical strategies — to efficiently advance the task goal.

Goal:
- First, evaluate the previous browser exploration strategy and reflect on its usefulness and efficiency toward the goal.
- Combine the user query, browser exploration trajectory memory, previous observation and strategy, current page observation, and other relevant information to output a clear and executable next-step browser exploration strategy.
- The exploration strategy must be built using the available toolset; only one tool may be used per step.

Available tools:
tool_descs

Input:
- Query: "<user's question>"
- Info_to_collect: "<current information-gathering goals>"
- Info_not_to_collect: "<current information points that do not need to be collected>"
- Collected_information: "<key information already collected (in bullet form)>"
- Memory of the browser exploration trajectory: "<hierarchical memory of browser exploration trajectory: records exploration policy, observations, and reflections>"
- Observations of the last step: "<observation from the previous step>"
- Last step policy: "<previous strategy>"
- Current Observations: "<current observation>"
- Visited buttons: "<buttons already visited>"
- Previous potential buttons: "<buttons with potential to yield valuable information>"

Output format:
<reasoning>
Briefly reason from the following aspects:
- What is the current highest-priority unsolved goal?
- Based on the Memory of the browser exploration trajectory, are there unexplored new goals or completed ones? How can the exploration path be optimized?
- Based on Visited buttons, how can repeated exploration of the same button be avoided?
- Based on Previous potential buttons, how can comprehensive and efficient exploration of potential content be ensured?
</reasoning>

<reflection>
{
"reflection": "In one or two sentences, evaluate the contribution and efficiency of the previous strategy toward the goal, reflect on the reasons, and propose one concise corrective suggestion for the next step; if necessary, explicitly mention actions that should not be taken."
}
</reflection>

<tool_call>
{
"name": "tool_name",
"parameters": { ... }
}
</tool_call>

Principles:
1. Strict format — Output only the <reasoning></reasoning>, <reflection></reflection>, and <tool_call></tool_call> blocks with JSON objects; no extra text.
2. Evidence-based — Use concrete clues

and memory to generate reflections and next-step strategies.
3. Avoid info_not_to_collect — When designing the next action, avoid collecting information that has been verified or deemed irrelevant.
4. Improve efficiency through strategy history — When formulating the next strategy, refer to the Memory of the browser exploration trajectory; if a similar or identical strategy has already been executed on this page, do not repeat it. Instead, change the target content or interaction type.
5. Improve efficiency through visited buttons — Refer to Visited buttons to avoid exploring the same button again; prioritize unvisited or new buttons with potential, or deepen exploration of the currently explored button.
6. Improve efficiency through potential buttons — Use Previous potential buttons to enhance the completeness and efficiency of browser exploration.

Figure 7: Prompts for Explorer in WEBTRAVELER.