Web Sitemap Knowledge Can Enhance Autonomous Browsing

Yuyao Zhang[1], **Hongyu Lu**[1], **Jiajie Jin**[1], **Hongjin Qian**[2], **Shiyu Li**[1]
Zhao Yang[1], **Yutao Zhu**[1], **Ji-Rong Wen**[1], **Zhicheng Dou**[1]*
[1]Renmin University of China, [2]BAAI
{yuyaoz, douzc}@ruc.edu.cn

Abstract

Recent advances in large language models (LLMs) have enabled web agents to perform interactive tasks on real-world websites. However, existing agents still suffer from limited robustness, efficiency, and task success, largely due to their lack of structural understanding of websites and the absence of browsing priors in pre-trained models. To address these challenges, this paper proposes the **W**eb **A**gent **S**itemap **P**rotocol (WASP), an agent-oriented sitemap that integrate structured website knowledge into web agents. WASP adopts a dual-granularity design, providing global site-level structure and local page-level semantic and interaction guidance. We also introduce a framework LightASM for constructing such sitemaps by identifying core pages and generating concise semantic summaries and block-level descriptions. Experiments on real-world browsing benchmarks demonstrate that WASP substantially improves the robustness, efficiency, and effectiveness of LLM-based web agents **without extra training**.

1 Introduction

A significant part of our learning, work, daily life, and entertainment activities occur online. However, many routine web interactions, such as clicking, typing, scrolling and filling forms, still remain repetitive, time-consuming, and cognitively taxing. As these digital worlds' tasks continue to grow in complexity and scale, addressing these challenges through effective web automation has become increasingly crucial to enhancing productivity and improving user experience.

Previous automation methods relying on handcrafted scripts and heuristic rules, such as DOM parsing and selector-based interactions (Kushmerick, 2000; Arasu and Garcia-Molina, 2003), face significant limitations: they lack robustness against minor webpage changes, exhibit poor generalization to new tasks or sites, and lack the intelligence to handle complex, personalized user needs. Thus, developing more flexible and intelligent automation techniques is essential for achieving efficient and reliable web automation.

Recent advances in large language models (LLMs) have demonstrated remarkable potential to overcome these limitations. LLMs have achieved strong performance across a wide variety of domains and tasks (Zhao et al., 2025), fueling the vision of intelligent agents that can perceive environments, understand user intent, and carry out complex reasoning and planning. Despite these advances, however, recent studies show that **LLM-driven agent still face significant challenges in autonomous web browsing tasks**—even those requiring limited reasoning—highlighting a critical gap in current capabilities (Deng et al., 2023; Koh et al., 2024; Xue et al., 2025a).

Real-world websites pose significant challenges for web agents. First, **LLM-based web agents do not fully understand website structure**. These websites often span numerous pages with deep and complex navigation hierarchies, which are difficult to infer from isolated page-level observations. Second, **observation modalities fail to provide fine-grained semantic understanding of web interfaces**. HTML-based inputs are token-intensive and lack spatial layout information (**?**), while visual snapshots cannot expose dynamic or hidden interface components (*e.g.*, folded navigation bars) to interaction. As autonomous browsing tasks typically require multi-page traversal and intent-aware interface interpretation, these limitations severely constrain current web agents' ability to operate robustly on real-world websites.

Existing research focuses on designing effective workflows (Sodhi et al., 2023; Zheng et al., 2024; Wang et al., 2025), developing multi-agents framework, or enhancing the reasoning ability of

*Correpsonding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 29307–29321

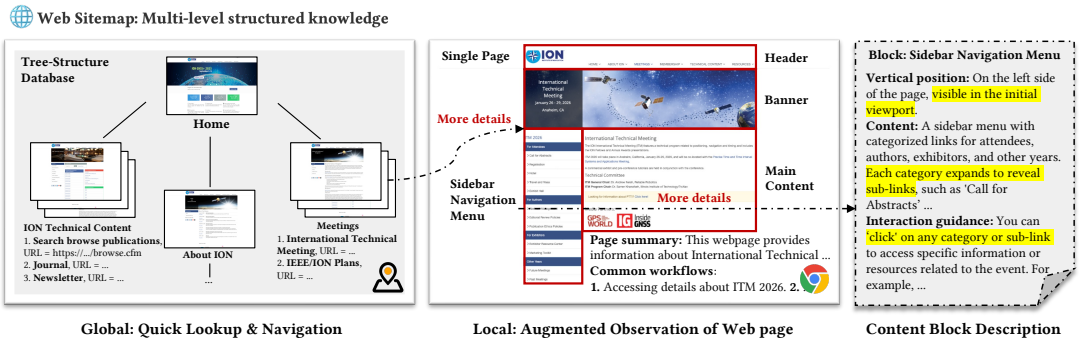

Figure 1: The overview of Web Agent Sitemap. For each website, it stores the navigation relationships in tree-structure and provides augmented observation of core pages for web agents.

backbone LLMs (Sodhi et al., 2023; Zheng et al., 2024; Wang et al., 2025; Lai et al., 2024; Qi et al., 2024; OpenAI, 2024b), emphasizing the internal ability of web agents while neglecting the external prior knowledge about the website environment.

Although some methods attempt to improve environmental perception through "world models" (Gu et al., 2025; Chae et al., 2024), they typically rely on modeling local environmental state transition rather than capturing the inherent structure and content of real websites, which also incurs inference overhead. As a result, **current web agents still lack a comprehensive understanding of the website they interact with**. This raises a natural yet important question: *How can we enable language-driven web agents to become familiar with real-world websites and effectively understand the elements and content of web pages?*

Many web service providers offer public XML sitemaps that present a navigable overview of website structure to human users . [1] These sitemaps enumerate a large set of URLs and serve as high-level site documentation. However, they remain limited for web agents. First, human-oriented sitemaps include many irrelevant or auxiliary pages, making them difficult for agents to utilize. Second, during page-level interactions, sitemaps provide no additional guidance, forcing agents to rely solely on HTML text or visual snapshots without actionable structural priors. An effective sitemap for web agents should instead offer actionable guidance throughout interactions, functioning as a true "map" that directs the agent toward its goals.

Based on these observations, we propose to extract structured knowledge from websites to improve autonomous browsing. Specifically, we introduce the Web Agent Sitemap Protocol (WASP), a sitemap service that defines a standardized interface between agents and websites. WASP represents a website as a tree-structured abstraction of multi-level knowledge, including page-level summaries and block-level descriptions for core pages, providing agents with explicit semantic and navigational guidance. Through this protocol, agents can query pre-defined sitemaps at each step to reason about global website structure, reducing trial-and-error exploration, while also accessing augmented page-level observations and actionable interaction guidance during browsing.

To efficiently construct such sitemaps, we further propose LightASM, a lightweight sitemap construction framework. LightASM first adopts a function-based crawling strategy to identify core pages, substantially reducing storage and construction overhead. It then leverages website structural priors and browsing expertise to generate multi-level abstractions with fine-grained interaction guidance (Figure 1). During inference, agents can look up the pre-built sitemap at each step to access website knowledge. Experiments on two available real-world autonomous browsing benchmarks, WebWalker (Wu et al., 2025) and WebVoyager (He et al., 2024), show that WASP significantly improves task success rates, generalization, and interaction efficiency.

Our contributions are as follows:

(1) We propose WASP, a protocol designed for web agents, which establishes a paradigm for efficient interaction between agents and websites.

(2) We propose a LightASM, a lightweight sitemap construction framework that extracts web knowledge across global and local levels.

(3) We design WASP as an extensible service that can be queried by LLM agent and can be deployed as web service to support robust web automation.

[1] https://en.wikipedia.org/wiki/Site_map

2 Related Work

2.1 Autonomous Browsing Benchmarks

Early autonomous browsing benchmarks such as MiniWoB++ (Shi et al., 2017) focus on basic interactions on synthetic websites. Mind2Web (Deng et al., 2023) provides an offline setting of cached real-world websites for evaluation, but this environment are not dynamic and can not provide feedback for agents. To address the limitations of static environments, WebLINX (**?**), WebShop (Yao et al., 2022) introduce a sand-boxed dynamic web simulation, enabling multi-turn interaction and exploration of web agents. WebArena (Zhou et al., 2024) extends this by providing fully functional containerized website replicas using Docker, enabling more realistic and reproducible evaluations. Recent work such as WebVoyager (He et al., 2024), Online-Mind2Web (Xue et al., 2025b), WebWalker (Wu et al., 2025) and BrowserComp (OpenAI, 2024a) requires agents to perform tasks on real-time websites, highlighting challenges in **dynamic content, element loading, and interaction robustness**.

2.2 Web Agents

Since web agents need to interact with websites to complete tasks, work such as SteP (Sodhi et al., 2023), Synapse (Zheng et al., 2024), and AWM (Wang et al., 2025) has focused on designing or extracting task-specific browsing workflows as in-context examples to improve agent performance. Other studies explore how to enhance browsing capabilities directly by fine-tuning agents on trajectory data or introducing reinforcement learning–based strategies (Lai et al., 2024; Qi et al., 2024; OpenAI, 2024b). These approaches are lack of incorporation of prior knowledge about the website environment.

To address this, WebDreamer (Gu et al., 2025), WMA (Chae et al., 2024), and WebEvolver (Fang et al., 2025) introduce world models to predict environmental transitions and guide action selection. More recent studies even explore leveraging website APIs as an alternative means of execution (Song et al., 2025; Zheng et al., 2025). However, these forms of prior knowledge often emphasize local state transitions, while largely overlooking the global structure of websites. As a result, **web agents tend to lack a holistic understanding of the environments they work in**—a key bottleneck in achieving robust performance in complex real-world web scenarios.

3 Preliminary

In this section, we give a definition of autonomous browsing tasks and some insights about how to build an intelligent web agent from this definition.

3.1 Problem Formulation

Web agents can interact with the web environment by formatted action commands in natural language. This process can be formulated as three parts:

Web Environment. The web environment $\mathcal{E}$ can be denoted as:

$$\mathcal{E} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T} \rangle, \tag{1}$$

with the state space of the website $\mathcal{S}$, the action space $\mathcal{A}$ (*e.g.*, `CLICK`, `TYPE`, `SCROLL`, etc.), the observation space $\mathcal{O}$ with observation function $f : \mathcal{S} \to 2^{\mathcal{O}}$ (*e.g.*, a webpage can be observed as HTML text, snapshot or accessible tree, etc.), and the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, which is implemented by the website environment.

Autonomous Browsing Task. Given a task instruction I, the current observation $o_t \in \mathcal{O}$ and the history $h_{1:t-1} = \{(o_k, a_k)\}_{k=1}^{t-1}$, agent needs to take an action $a_t \in \mathcal{A}$. The action results in a new state $s_{t+1} \in \mathcal{S}$ and its observation $o_{t+1} \in \mathcal{O}$ based on the transition function $\mathcal{T}$ and the observation function f. For each $a_t \in \mathcal{A}$,

$$a_t \sim \pi_\theta(a | h_{1:t-1}, o_t, \mathcal{I}), \tag{2}$$

where π_θ is the parameterized policy model by a designed web agent, and some common actions are listed in Table 5.

Task Evaluation. Given an observation sequence $\{o_i\}_{i=1}^{n+1}$ and the set of tasks-relevant observations $\mathcal{O}_I$, the task is considered successful if the final response is correct and there exists a subset of key observations $\{o_{k_1}, o_{k_2}, ..., o_{k_m}\}$, where $m \leq n+1$, such that:

$$\forall i \in \{1, ..., m\}, o_{k_i} \in \mathcal{O}_I. \tag{3}$$

3.2 Insights from Formulation

From the problem formulation, we identify three key challenges for autonomous web agents.

Strong policy models alone are insufficient. While stronger reasoning models, improved training algorithms, or more sophisticated agent workflows can enhance action selection, they primarily address the policy π_θ and often require substantial data or system complexity.

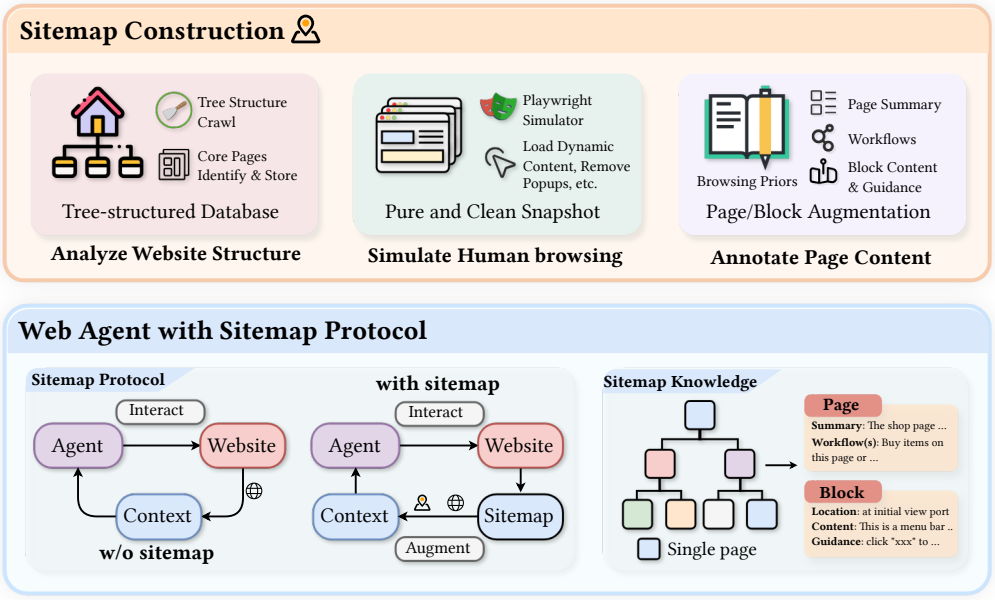

Figure 2: Overview of WASP that contains a sitemap construction framework and autonomous browsing with sitemap.

Web state transitions are inefficient. Although tasks typically involve only a small number of key pages, reaching these pages often requires many intermediate navigation steps due to website-specific transition dynamics $\mathcal{T}$, most of which are irrelevant to the task.

State observations are incomplete. Effective decision-making critically depends on high-quality state observations. However, existing observation functions $f(\cdot)$—such as HTML text, window-sized screenshots, or accessibility trees—provide only partial views and lack the structural and semantic priors necessary for efficient browsing.

4 Method: Web Agent Sitemap Protocol

In this section, we introduce the Web Agent Sitemap Protocol (WASP), an agent-oriented abstraction of websites, together with LightASM, a lightweight framework for constructing WASP-compliant sitemaps.

4.1 Web Agent Sitemap Protocol

Modern websites expose rich structural and interaction priors to human users, but such knowledge is largely inaccessible to LLM-based web agents. To bridge this gap, we design WASP as a standardized protocol that enables websites to expose structured, agent-consumable knowledge beyond raw HTML or visual observations.

Table 1: Two-level design of the Web Agent Sitemap Protocol.

Granularity	Description	Agent Use
Global Level	Site structure, core pages, link hierarchy	Navigation, planning
Local Level	Per-page summary, semantic element blocks, workflows	Grounding, interaction

Protocol Overview. WASP defines a JSON schema that represents websites as structured, hierarchical knowledge base for web agents. The schema formalizes the organization of **site-level structure** (e.g., page hierarchy and child links), **page-level semantics** (e.g., functional summaries and typical user workflows), and **interaction guidance** (e.g., semantic content blocks with actionable elements), enabling consistent and interoperable integration across different websites and agent architectures. The complete schema specification is provided in Appendix B.

Dual-Granularity Design Principle. At the core of WASP is a dual-granularity representation, which decomposes website knowledge into a global level and a local level (Table 1):

Global level captures the overall website structure, including core pages and their navigational relationships, supporting long-horizon planning and navigation.

Local level augments individual pages with se-

mantic summaries, content block descriptions, and common workflows, enabling fine-grained grounding and interaction.

This separation allows agents to reason jointly about global navigation and local interaction, while avoiding the noise and inefficiency of modeling all pages uniformly.

4.2 LightASM: Sitemap Construction

To efficiently construct web agent sitemaps at low cost, we introduce **LightASM**, a **Light**weight framework for **A**gent-oriented **S**ite**M**ap construction. LightASM decomposes sitemap construction into three stages: (1) identifying core pages, (2) capturing dynamically rendered content, and (3) generating structured semantic annotations.

4.2.1 Website Structure Analyzer.

The Website Structure Analyzer is designed to explore, select core pages and filter pages thar are noisy for web agents to execute tasks.

Core page selection. The analyzer traverses the website starting from the homepage and identifies a subset of *core pages* using a scoring-based filtering strategy. For each candidate page p, we define a page importance score:

$$s(p) = \mathrm{Score}(p), \tag{4}$$

where $\mathrm{Score}(\cdot)$ is a configurable function that may incorporate factors such as click depth, URL patterns, structural redundancy or other heuristic priors. Pages with high scores are retained in the sitemap, while low-scoring or redundant pages are filtered out. This abstraction allows flexible implementations on different websites while enforcing a compact and representative site structure.

Noise reduction. Domain-specific heuristics and URL-pattern-based filters are further applied to remove auxiliary pages such as archives or duplicated templates (Appendix F). This hierarchical exploration produces a compact yet representative sitemap that preserves essential navigational structure with minimal crawling overhead.

4.2.2 Simulated User Explorer.

Many websites rely on dynamic rendering and interaction-triggered content loading, making static crawling insufficient to recover complete page information. To address this, we design the Simulated User Explorer to actively interacts with webpages to expose hidden or delayed content.

Interaction-driven exploration. For each page p, the explorer simulates a sequence of lightweight user interactions, including clicking expandable elements, scrolling through the page, and dismissing obstructive pop-ups or dialogs. These actions are designed to simulate common user behaviors and to trigger content that is conditionally rendered or initially hidden from view.

State collection. As interactions are executed, the explorer records the changed states of the intial page. Formally, for each page p, we collect a set of observations:

$$\mathcal{S}(p) = \{(D_k, I_k)\}_{k=1}^{K(p)}, \tag{5}$$

where D_k and I_k denote the DOM tree and the corresponding screenshot at interaction step k. This representation captures both structural and visual changes induced by simulated user actions such as close the pop-ups, scroll down to get full page and other useful interactions.

By doing this, the explorer recovers a more complete view of page content and layout than static snapshots. This process improves robustness to dynamic loading mechanisms and interface variations, providing reliable inputs for downstream semantic annotation and sitemap construction.

4.2.3 Page Content Annotator.

After identifying core pages and obtaining their fully rendered snapshots, the Page Content Annotator will be given a browsing-prior-aware instruction $\mathcal{I}$, together with a multimodal representation $\mathbf{h}$ of the page, generating structured semantic annotations $(S, \mathcal{W}, \mathcal{D})$ to provide actionable browsing priors for web agents.

Multimodal representation. For each core page, we jointly leverage textual metadata and visual layout information. Let T denote page-level texts (e.g., URL, title, and content), and let $I \in \mathbb{R}^{H\times W\times C}$ denote the full-page screenshot capturing the complete layout. They are encoded by dedicated encoders and fused into a unified representation:

$$\mathbf{h} = \mathrm{Fuse}\big(\mathrm{Enc}_T(T), \mathrm{Enc}_I(I)\big), \tag{6}$$

where Enc_T and Enc_I denote text and image encoders, and this represention captures both semantic intent and spatial structure of the webpage.

Structured annotation. Conditioned on the multimodal representation $\mathbf{h}$ and instruction $\mathcal{I}$, the annotator produces three types of structured outputs

critical for autonomous browsing:

$$(S, \mathcal{W}, \mathcal{D}) \sim P_\theta(\cdot \mid \mathbf{h}, \mathcal{I}), \tag{7}$$

where S is a **page-level summary** describing the primary function of the page, $\mathcal{W} = \{a_1, \ldots, a_{N_w}\}$ denotes typical **user workflows**, and $\mathcal{D} = \{d_1, \ldots, d_M\}$ represents **block-level descriptions** that semantically characterize distinct content blocks identified via visual and structural cues.

These multi-level annotations populate the local-level fields of the WASP json schema and serve as explicit semantic and interaction priors. By bridging raw webpage content and structured understanding, the Page Content Annotator enables more accurate grounding, efficient navigation, and robust task completion for web agents.

4.3 Integration

During inference, WASP serves as a queryable knowledge interface that web agents can flexibly access throughout task execution. Given the current page URL u and optionally the task description q, the agent retrieves structured website knowledge from the sitemap $\mathcal{S}$:

$$\mathcal{K}(u, q) = \text{Lookup}(\mathcal{S}, u, q), \tag{8}$$

which augments standard observations with both site-level structure and page-level semantic and interaction priors.

Task-aware retrieval and refinement. Beyond direct URL-based lookup, the agent can also issue task-driven queries to retrieve a candidate set of relevant pages $\mathcal{C}$ from the sitemap. Given the agent's internal state s, which encodes task progress, browsing history, and intermediate findings, the agent dynamically refines this candidate set via a state-conditioned reranking function:

$$\hat{p} = \arg\max_{p \in \mathcal{C}} f_{\text{rerank}}(p \mid s). \tag{9}$$

This enables adaptive prioritization of sitemap entries as the task evolves, allowing the agent to seamlessly alternate between exploration and fine-grained interaction.

5 Experiments

In this section, we present the effectiveness of WASP through a series of soundness experiments on real-world scenarios.

5.1 Benchmarks

We evaluate the effectiveness of WASP in two real-world scenario autonomous browsing benchmarks as follows:

WebWalker (Wu et al., 2025). WebWalker is a challenging benchmark designed to evaluate the ability of LLM-based agents in web traversal. For network access availability, we select two subsets within the education and conference domains, comprising 136 and 139 tasks respectively.

WebVoyager (He et al., 2024). WebVoyager is an autonomous browsing benchmark comprising 643 tasks from 15 popular websites. For network access availability, we select a subset of 127 tasks across three websites for our experiments.

5.2 Baselines

Due to real-world autonomous web browsing is a challenging task for LLMs, We mainly select two open-source models that have potential to reasoning in long-horizon web browsing tasks, `Qwen-2.5-72B` and `Qwen-2.5-VL-72B` as backbones for web agents. To valid the effectiveness of sitemap, we implement two classic agent architectures as baselines. The first **directly generates actions** based on the current state of website and history, while the second employs the **ReAct (Yao et al., 2023)** framework, performing iterative observation, reasoning and action output. WASP only differs by integrating the knowledge of the sitemap into agent context during browsing.

5.3 Evaluation

We follow the Online-Mind2Web (Xue et al., 2025b) and WebVoyager (He et al., 2024), employing an automatic evaluation method via LLM-as-a-Judge to assess the success rate of autonomous browsing tasks, which demonstrate over 80% consistency with human evaluation. As discussed in Section 3.1, screenshots of the entire trajectory and the final response to the user are fed into a powerful multimodal language model for evaluation.

5.4 Implementation

Web Agent Setup. We mainly implement web agents using the open-source `browser-use` framework (Müller and Žunič, 2024) and playwright [2], building a simulated browser environment with a structured action space (Table 5). During inference, the agent generates one grounded action per step

[2] https://playwright.dev/

Table 2: Main results on WebWalker benchmark.

Methods	Education			Conference			Total
	Single source	Multi source	Average	Single source	Multi source	Average	Average
Accessible tree prompt							
Qwen2.5-72B	15.7	5.7	11.8	23.2	18.6	20.9	16.4
+ ReAct	16.9	3.8	11.8	21.7	18.6	20.1	16.0
+ WASP (ours)	**25.3**	**5.7**	**17.6**	**30.4**	**24.3**	**27.3**	**22.5**
Accessible tree prompt + Screenshot with Set of Marks							
Qwen2.5-VL-72B	18.1	3.8	12.5	21.7	11.4	16.5	14.5
+ ReAct	20.5	7.5	15.4	23.2	18.6	20.9	18.2
+ WASP (ours)	**28.9**	**7.5**	**20.6**	**33.3**	**28.6**	**30.9**	**25.8**

Table 3: Main results on WebVoyager benchmark.

Methods	WebVoyager			Total
	Apple	ArXiv	Github	Avg.
Accessible tree prompt				
Qwen2.5-72B	32.6	53.5	41.5	42.5
+ ReAct	39.5	51.2	41.5	44.1
+ WASP (ours)	**53.5**	**60.5**	**43.9**	**52.8**
Accessible tree prompt + Screenshot with Set of Marks				
Qwen2.5-VL-72B	44.2	55.8	41.5	47.2
+ ReAct	44.2	**60.5**	41.5	45.7
+ WASP (ours)	**65.1**	58.1	**51.2**	**58.3**

Table 4: Ablation studies on Qwen2.5-VL-72B.

Methods	Edu.	Conf.	WebV.	Avg.
WASP	**20.6**	**30.9**	**58.3**	**36.1**
w/o Global	13.2	15.1	55.1	27.1
w/o Local	17.8	26.6	46.5	29.9

with decoding temperature set to 0. Each task is limited to a maximum of 20 interaction steps.

Sitemap construction and inference with sitemap. We use LightASM to build sitemaps for each website. The Page-level and block-level annotations are generated by GPT-4o-mini. For each task, the agent retrieves the top-20 relevant pages from the sitemap to guide navigation and grounding. For each step, sitemap provides block-level descriptions of current located page (if exist).

5.5 Result

The main results of WASP are reported in Table 2 and 3. Across both challenging real-world benchmarks, LLM-based web agents augmented with sitemap knowledge achieve the best overall performance. We highlight two key observations.

Sitemaps substantially improve task success. Under both the Accessible Tree prompt setting and the multimodal input combining screenshots with sets of marks, sitemap-enhanced agents consistently outperform their baselines. Compared to ReAct, sitemap integration provides richer browsing priors, leading to significantly improved navigation and page-level interaction, and ultimately higher task success rates.

External knowledge is more effective than inference-time optimization. While inference-time strategies such as ReAct improve performance by refining the internal policy through iterative reasoning, their gains remain limited. In contrast, integrating external sitemap knowledge yields substantially larger improvements, highlighting the importance of structured website priors.

Overall, these results strongly support our central hypothesis: **current web agents lack sufficient familiarity with real-world websites**, and incorporating explicit structural and semantic knowledge is crucial for robust autonomous web browsing.

6 Analysis

In this section, we further analyze the effectiveness of web agent sitemap through a series of soundness analysis experiments.

6.1 Ablation Study

We conduct our WASP ablation studies with Qwen-2.5-VL-72B as backbone, with results summarized in the Table 4. Several key insights from the ablation study are:

(1) **Global knowledge enhances navigation capabilities.** For the tasks of WebWalker, the absence of global knowledge leads to a more pronounced performance decline. This is because tasks in these datasets primarily involve collecting and integrating information from specific sub-sections of websites, which typically require agents to possess a deep understanding of the site's overall structure

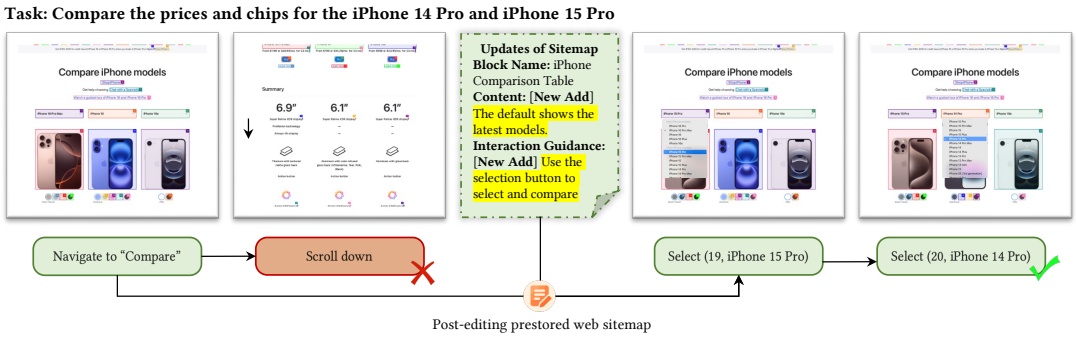

Figure 3: Overview of WASP that contains a sitemap construction framework and autonomous browsing with sitemap.

and functionality of each sub-section.

(2) **Local knowledge improves fine-grained interactions.** For the WebVoyager benchmark, tasks require executing a series of interactions on one page. For example, task "*Compare the prices and chips for the iPhone 14 Pro and iPhone 15 Pro models*", involving **clicks**, **selections**, and **scrolling** on the comparison page. Such tasks demand agents to have detailed interaction experience, which is often lacking in general (vision) language models.

6.2 Efficiency Analysis

When tasks involve multiple consecutive navigation steps, sitemap guidance can significantly enhance the agent's ability to efficiently locate target pages, minimizing trial-and-error interactions and allowing it to focus on critical task actions. As demonstrated in Figure 4, this effect is particularly evident in the WebVoyager benchmark, where agents augmented with sitemap knowledge achieve higher success rates (52.8 vs. 44.1 and 58.3 vs. 47.2) while completing tasks with considerably fewer steps (5.26 vs 6.18) compared to baseline methods. This improvement in interaction efficiency is crucial for building robust and scalable autonomous web automation systems.

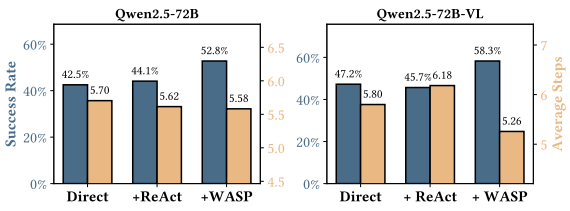

Figure 4: The average number of steps and successful rate across different methods.

6.3 Updates of Sitemap

Figure 3 demonstrates how updating the sitemap with page content annotator improves performance. Initially, the agent failed to locate relevant data by simply navigating to the "Compare"section and scrolling down. This failure was due to the agent's inability to identify the selection drop-down menus from the accessible trees and its reliance on limited visual information from the screenshot. After revising the sitemap block *iPhone Comparison Table* to include explicit interaction guidance of 'selection', the agent does not scroll down initially. This highlights the **extensibility** of the sitemap, demonstrating the critical role of precise and actionable sitemap updates in enhancing the robustness and performance of web agents during task execution.

7 Conclusion

In this paper, we analyze the poor performance of LLM-driven web agents in automation tasks from two key perspectives: (1) insufficient understanding of website structures and content, and (2) the lack of prior web browsing knowledge. To address these challenges and enable robust, intelligent web automation, we propose the **Web Agent Sitemap Protocol (WASP)**, designed to enhance agent-website interactions by incorporating web browsing prior knowledge through a dual-granularity design principle. Additionally, we introduce LightASM, a lightweight framework for efficiently constructing these agent-oriented sitemaps at low cost. This method requires no extra training for specific scenarios, and the update and maintenance of the sitemap are easy to carry out. Extensive empirical evaluations demonstrate the effectiveness of this sitemap service in building robust and deployable autonomous web browsing agents for real-world applications.

8 Limitation

In this work, we primarily discuss how external website structural and interaction knowledge can enhance the performance of Web Agents. However, there is limited discussion on aspects such as workflow priors and advanced browsing techniques, which suggests directions for future improvements.

9 Acknowledgment

The work was partially done at the Beijing Key Laboratory of Research on Large Models and Intelligent Governance and Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE.

References

Arvind Arasu and Hector Garcia-Molina. 2003. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2024. Web agents with world models: Learning and leveraging environment dynamics in web navigation. *CoRR*, abs/2410.13232.

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 Information Processing Systems*, volume 36, pages 28091–28114. Curran Associates, Inc.

Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. 2025. Webevolver: Enhancing web agent self-improvement with coevolving world model.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. 2025. Is your LLM secretly a world model of the internet? model-based planning for web agents. *Trans. Mach. Learn. Res.*, 2025.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 6864–6890. Association for Computational Linguistics.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024. Tree search for language model agents. *CoRR*, abs/2407.01476.

Nicholas Kushmerick. 2000. Wrapper verification. *World Wide Web*, 3(2):79–94.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, pages 5295–5306. ACM.

Magnus Müller and Gregor Žunič. 2024. Browser use: Enable ai to control your browser.

OpenAI. 2024a. Browsecomp: a benchmark for browsing agents. Accessed: 2025-05-05.

OpenAI. 2024b. Using a computer with an agent. Accessed: 2025-05-05.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. 2024. Webrl: Training LLM web agents via self-evolving online curriculum reinforcement learning. *CoRR*, abs/2411.02337.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR.

Paloma Sodhi, SRK Branavan, Yoav Artzi, and Ryan McDonald. 2023. Step: Stacked llm policies for web actions. *arXiv preprint arXiv:2310.03720*.

Yueqi Song, Frank Xu, Shuyan Zhou, and Graham Neubig. 2025. Beyond browsing: Api-based web agents.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025. Agent workflow memory. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. 2025. Webwalker: Benchmarking llms in web traversal. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 10290–10305. Association for Computational Linguistics.

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

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

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

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 *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A survey of large language models.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. 2025. Skillweaver: Web agents can self-improve by discovering and honing skills.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Appendix

A Use of AI Assistants

In this paper, we use ChatGPT to improve the presentations of this paper.[3]

B JSON-Schema of Web Agent Sitemap Protocol

We provide the complete JSON schema specification of the Web Agent Sitemap Protocol (WASP), which formally defines the structure, fields, and constraints of agent-oriented sitemaps.

Listing 1: Complete JSON Schema of WASP.

```
{
  "$schema": "https://json-schema.org/draft
      /2020-12/schema",
  "$id": "https://example.org/wasp.schema.json",
  "title": "Web Agent Sitemap Protocol (WASP)",
  "type": "object",
  "required": ["url", "title", "depth", "
      children", "content"],
  "properties": {
    "url": {
      "type": "string",
      "description": "Canonical URL of the web
          page."
    },
    "title": {
      "type": "string",
      "description": "Agent-readable title of the
           page."
    },
    "depth": {
      "type": "integer",
      "minimum": 0,
      "description": "Click depth from the
          website homepage."
    },
    "children": {
      "type": "array",
      "description": "Child pages in the site
          hierarchy.",
      "items": {
        "type": "string"
      }
    },
    "content": {
      "type": "object",
      "required": ["page_summary", "
          key_user_flows", "blocks"],
      "properties": {
        "page_summary": {
          "type": "string",
          "description": "Functional summary
              describing the purpose of the page
              ."
        },
        "key_user_flows": {
          "type": "array",
          "description": "Typical user workflows
              associated with the page.",
          "items": {
            "type": "string"
          }
        },
        "blocks": {
          "type": "array",
          "description": "Semantic content blocks
               within the page.",
          "items": {
            "type": "object",
            "required": [
              "name",
              "vertical_position",
              "content",
              "interaction_guidance"
            ],
            "properties": {
              "name": {
                "type": "string",
                "description": "Name or functional
                     role of the content block (e.
                    g., header, sidebar)."
              },
              "vertical_position": {
                "type": "string",
                "enum": ["top", "middle", "bottom
                    "],
                "description": "Approximate
                    vertical position of the
                    block in the page layout."
              },
              "content": {
                "type": "string",
                "description": "Semantic
                    description of the block
                    content."
              },
              "interaction_guidance": {
                "type": "string",
                "description": "Actionable
                    guidance for interacting with
                     this block."
              }
            }
          }
        }
      }
    }
  }
}
```

[3] https://chatgpt.com/

C The System Prompt of Web Agent

The system prompt of our implemented web agents is as follows:

SYSTEM PROMPT

You are a precise browser automation agent that interacts with websites through structured commands. Your role is:

1. Analyze the provided webpage elements and structure
2. Use the given information to accomplish the ultimate task
3. Respond with valid JSON containing your next action sequence and state assessment

D The Action Space of Web Agent

We details the action space available to our implemented web agent during autonomous browsing tasks in Table 5. The agent interacts with web pages through a set of **discrete actions**, including element manipulation (clicking, selecting, typing), page navigation (scrolling, URL navigation, tab management), content extraction, and task completion signaling.

Table 5: Action space of a web agent.

Action	Description
`click(id)`	Click a DOM element by its id.
`select(id, option)`	Select an option from a dropdown.
`type(id, text)`	Input text into a text field.
`scroll(pixel)`	Scroll the page vertically by pixels.
`go_to_url(url)`	Navigate the browser to a URL.
`open_new_tab()`	Open a new browser tab.
`switch_tab(id)`	Switch focus to a different tab.
`extract_content()`	Extract content from page.
`done(response)`	Submit the final answer or response.

During inference, these actions are formatted and fed into the LLM's context as structured commands. For each action, we provide an example output with a description. The format is as follows:

ACTION SPACE PROMPT

Complete task with response to user:
{done: {'text': {'type': 'string'}}}
Navigate to URL in the current tab:
{go_to_url: {'url': {'type': ''string'}}}
Click element:
{click_element: {'index': {'type': 'integer'}, 'xpath': {'anyOf': [{'type': 'string'}, {'type': 'null'}], 'default': None}}}
... (other actions)

This structured prompting enables the LLM to produce valid and contextually appropriate action commands, facilitating precise grounding and execution in the web environment.

E The Observation of Web Agent

In this section, we describe how we build observation prompt for our implemented Web Agent.

Global browsing context. To construct the observation prompt, we first extract global contextual information from the fully rendered webpage, including the current URL and the list of open browser tabs.

Interactive element identification. We traverse the DOM tree to identify interactive elements that expose actionable affordances, such as buttons, input fields, links, and selectable components. Each interactive element is assigned a unique numeric index, which serves as a stable reference for precise action grounding across interaction steps.

Actionable element normalization. For each interactive element, we extract its HTML tag type and visible text, or generate a concise description when textual content is unavailable. These elements are serialized into a unified textual format. Interactive elements will be assigned with a numeric index while non-interactive elements are retained without numeric indices to preserve semantic context, explicitly preventing the agent from attempting invalid interactions. The overall prompt is presented as follows:

OBSERVATION PROMPT

1. Current URL: The webpage you're currently on
2. Available Tabs: List of open browser tabs
3. Interactive Elements: List in the format:
index[:]<element_type>element_text</element_type>
- index: Numeric identifier for interaction
- element_type: HTML element type (button, input, etc.)
- element_text: Visible text or element description
Example:
[33]<button>Submit Form</button>
[] Non-interactive text
Notes:
- Only elements with numeric indexes inside [] are interactive
- [] elements provide context but cannot be interacted with

F The Filtering Rules of Crawler

The filering rules we use for Crawler is listed in Table 6. These heuristics include restricting crawling to specific URL path prefixes to focus on relevant content sections, enforcing domain restrictions to avoid external or advertisement links, and detecting pagination patterns to efficiently traverse multi-page listings.

G Benchmark Statistics

We provides an overview of the key statistics and characteristics of the benchmarks we used in experiments to evaluate our web agents.

Table 6: Filtering rules and URL patterns for tree-structured crawler.

Rule Category	Description
Ad URLs	External URLs irrelevant to main content
Login URLs	URLs related to user login/logout flows
Session/Tracking Parameters	URLs differing only by query parameters like session IDs or UTM tags
Resource URLs	Links to images, CSS, JS files, videos, and other non-HTML resources
Fragments	URLs with anchors (e.g., `#section`) pointing to page subsections
Domain heuristics	Crawl limited to specific directories or path prefixes (e.g., `/news/`, `/products/`)

G.1 WebWalker Subset

Table 7: Statistics for the WebWalker subset (accessible websites only).

Statistic	Value
Total Tasks	222
Unique Websites	27
Task Type Distribution	55.3% single_source, 44.7% multi_source
Domain Distribution	50.5% conference, 49.5% education
Top Website (by task count)	`cs.swust.edu.cn` (23 tasks)
Average Tasks per Website	Approx. 8.2

Table 7 summarizes the statistical characteristics of the WebWalker subset. The selected tasks are primarily drawn from university and academic conference websites, which are typically structured and information-dense.

G.2 WebVoyager Subset

Table 8 presents the statistical characteristics of the WebVoyager subset, which includes only websites that remained accessible and stable throughout our experimental period.

G.3 Representative Tasks

To illustrate the diversity and complexity of the tasks in our benchmark, we present some examples from both WebWalker (covering education and conference domains) and WebVoyager subsets. Each task involves multi-step reasoning and interaction with real-world websites.

WebWalker – Education Domain

Table 8: Statistics for the WebVoyager subset (accessible websites only).

Statistic	Value
Total Tasks	123
Unique Websites	3
Top Websites	`apple.com`, `arxiv.org`, `github.com`
Tasks per Top Website	Each with 41–43 tasks
Average Tasks per Website	Approx. 41.0

- **Task:** Which leaders attended the postgraduate entrance examination mobilization meeting for the 2025 graduates at the School of Computer Science and Technology, Southwest University of Science and Technology, and who chaired the meeting?
  Answer: The meeting was attended by Party Secretary Chen Yufang, Dean Zhang Hui, Vice Deans Jia Xiaolin and Huang Xiaofang, and Executive Deputy Director of the Elderly Workers Committee Lei Quanshui. It was hosted by Party Committee Deputy Secretary Xie Changyong.

- **Task:** Which professor gave a lecture at the 832nd session of the 'Xuecui Forum' at Harbin Engineering University on August 15, 2018, and what innovative solution did they propose?
  Answer: Dr. Xiaojiang (James) Du; e-SAFE (Secure, Efficient and Forensics-Enabled Access to Wireless Implantable Medical Devices)

- **Task:** What was the theme of Professor Du Xiaojiang's academic report held at Harbin Engineering University on December 19, 2016?
  Answer: Analysis of Clickjacking Attacks and An Effective Defense Scheme for Android Devices

WebWalker – Conference Domain

- **Task:** In which city and venue did the Joint Navigation Conference take place in 2025, and what were the dismantling hours for ION GNSS+ 2024 exhibitors?
  Answer: Greater Cincinnati Area at Northern Kentucky Convention Center; Thursday, 4:00 p.m. – 10:00 p.m.

- **Task:** Which scholars did Dr. Mathieu Joerger collaborate with when he received the

2014 and 2022 ION awards?
Answer: 2014: No collaborators; 2022: Elisa Gallon and Dr. Boris Pervan

- **Task:** What is the date when Dr. John Raquet received the Johannes Kepler Award at the ION GNSS+ 2024 conference?
Answer: September 20, 2024

WebVoyager

- **Task:** Compare the prices of the latest models of MacBook Air available on Apple's website.
- **Task:** Search for the latest preprints about "quantum computing" on arXiv.
- **Task:** Search for an open-source project related to "climate change data visualization" on GitHub and report the one with the most stars.

The detailed statistics of these datasets provide essential context for conducting our soundness experiments.

H Case Studies

To provide a concrete illustration of how WASP addresses the core challenges of existing web agents, we present two representative task execution trajectories, highlighting how sitemap-based prior knowledge enables agents to overcome:

- **Limited understanding of website structure and content**;
- **Lack of prior browsing knowledge within pre-trained language models**.

H.1 Case 1: WebVoyager arXiv Submission Guidelines Task

Task: Look up the submission guidelines on arXiv for submitting a paper and tell me the formats for figures.

- **Challenge:** The agent must locate deeply nested information on the arXiv website, which is not visible upon initial load and requires scrolling and multi-step navigation.
- **Naive Approach:** Without structural guidance, agents often rely on keyword searching and fall into a vicious circle.
- **WASP-Enhanced Strategy:** Leveraging sitemap-provided block-level context (e.g., "About arXiv" is at the bottom and contains links to submission guides), the agent efficiently scrolls, locates the correct link, navigates through a multi-level structure, and pinpoints the required information.

Figure 5 illustrates the enhanced reasoning trace and aligned actions empowered by WASP context.

H.2 Case 2: WebVoyage GitHub JavaScript Repository Retrieval Task

Task: Find a popular JavaScript repository created in the last 30 days on GitHub with a README file.

- **Challenge:** The agent must navigate to the trending page and interpret multiple dropdown filters and dynamic content rendering to query the correct list of repositories.
- **Naive Approach:** Pretrained agents often fail to recognize that the most recent repositories are listed on the Trending page, and struggle to identify and correctly interact with the relevant combinations of dropdown filters, such as language and date range.
- **WASP-Enhanced Strategy:** Sitemap provides descriptions of filter blocks, their default visibility, and expected behaviors (e.g., "updates results dynamically"), allowing the agent to plan a multi-step filter application and repository selection procedure.

Figure 6 demonstrates the structured plan and interaction sequence that resulted in a successful completion.

These two case studies show that WASP not only improves task completion success by grounding agent decisions in structured layout knowledge, but also bridges the gap between LLM priors and real-world web usage, resulting in significantly more robust and semantically informed interaction plans.

Figure 5: An example shows that web agent sitemap can enable structured navigation by providing location and interaction guidance, allowing the web agent to reason semantically over arXiv page's layout.

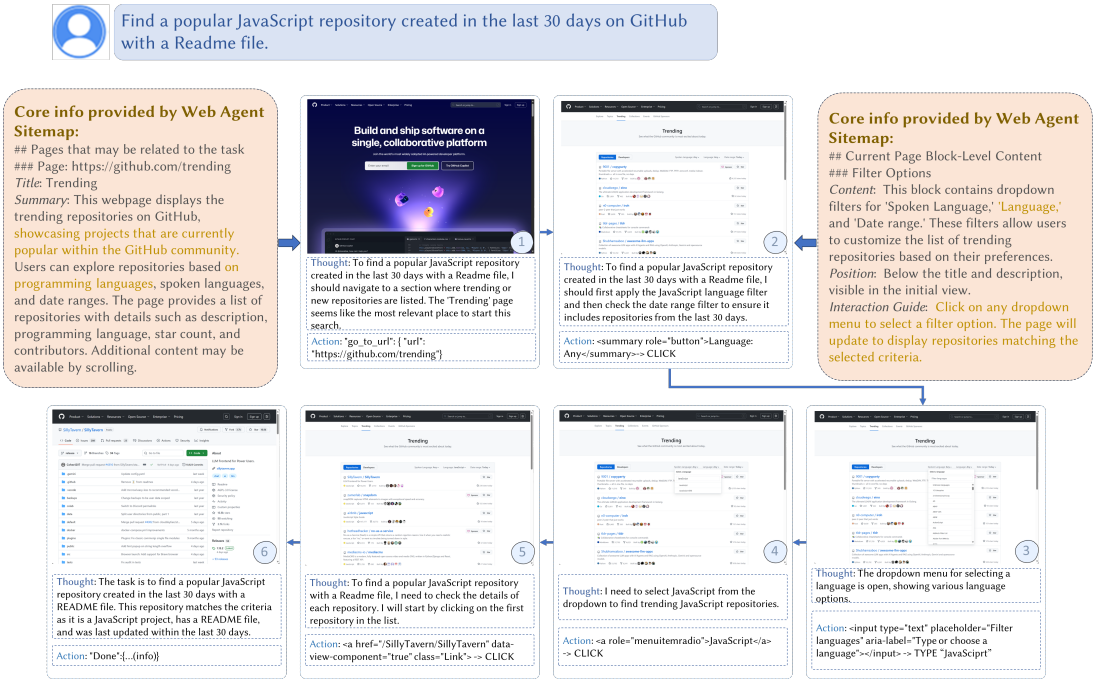

Figure 6: An example shows that web agent sitemap can help web agent to reason about drop-down usage and repository filtering by offering explicit knowledge of layout and expected interactivity.