PagePilot for Web Automation Based on Multi-Agent Architecture

Chi-Ju Yeh[1], **Chia-Hui Chang**[1]

[1]National Central University, Taiwan
tobyeider.work@gmail.com, chiahui@g.ncu.edu.tw

Abstract

Recent advances in reasoning and multi-modal analysis have enabled large language models (LLMs) to automate tasks like extracting data and filling out forms for web navigation. However, existing systems struggle with complex, long-form web interactions, such as parsing lengthy articles or optimizing multi-step workflows, due to limitations in processing dynamic content and scalable architecture. To address this gap, we introduce PagePilot, an automated web control system that integrates DOM source code analysis and multi-agent collaboration for robust task execution. Building on WebVoyager, PagePilot incorporates dynamic loading to handle infinite-scroll pages and asynchronous content, alongside a hierarchical reasoning framework for efficient decision-making. Evaluations demonstrate state-of-the-art performance: a 76% task success rate on WebVoyager (vs. 63% baseline) and 47% on GAIA (vs. 38% for GPT-4). To validate generalization, we curated benchmarks from Mind2Web and a novel Chinese web dataset, achieving 52% and 70% success rates, respectively, outperforming prior methods in linguistically diverse scenarios. Ablation studies confirm critical design choices, showing a 22% increase in task completion and 27% reduction in redundant actions compared to fragmentary approaches.

Code — https://github.com/fdsf53451001/PagePilot

Introduction

The launch of ChatGPT in 2022 marked a watershed moment in AI development, demonstrating a system capable of engaging in human conversation.

This breakthrough opens exciting possibilities for how we interact with computers. Traditional information systems rely on graphical user interfaces (GUIs) to provide a wide range of functions, forcing users to navigate complex menus and perform multiple steps to complete their desired tasks. Now, with advances in natural language understanding and computer vision, we can fundamentally redesign this interaction. Large Language Models can not only advise users but also perform operations directly.

For example, CogAgent (Hong et al. 2024) proposed at CVPR 2024 represents a significant advance in the field of conversational interface navigation. Built on the visual language model CogVLM, this system can process screenshots directly rather than relying on HTML text extraction. This approach has proven to be highly effective, with CogAgent outperforming traditional LLM-based methods in both the Mind2Web (Deng et al. 2024) and AITW (Rawles et al. 2024) benchmarks for personal computer and Android GUI navigation.

In addition, two papers on computer use and website browsing are accepted in ACL 2024: SeeClick and WebVoyager. SeeClick (Cheng et al. 2024) is a visual GUI agent that can automatically execute complex digital device tasks relying only on screenshots, while WebVoyager (He et al. 2024) is an agent created using GPT-4V's multimodal understanding capabilities to interact with real-world website. This growing field also includes other notable research projects such as WebAgent (Gur et al. 2024a), WebArena (Zhou et al. 2024), ToolLLM (Qin et al. 2023), OS-Pilot (Wu et al. 2024), etc. More research work could be found in the following two survey papers by Zhang et al. (2025) and Nguyen et al. (2025).

As research on computer use and website interaction expands, we can categorize these GUI agents into two types: **standalone applications** like web agents and **cross-application** OS-level agents on Windows or Mac.

Despite achieving a 59.1% success rate, WebVoyager still faces four significant challenges.

- Navigation Stuck (44.4% of cases): This occurs when the model fails to navigate to specified locations according to task instructions.
- Visual Grounding Issues (24.8% of cases): These problems arise when the model struggles to distinguish between similar or very small visual elements.
- Hallucination (21.8% of cases): This happens when the model prematurely or incorrectly completes tasks without following instructions.
- Prompt Misalignment (9% of cases): This involves miscommunication between user instructions and agent interpretation.

To address the above issues, we observe the following phenomena:

- Full web page augmentation: WebVoyager relies on screenshots to navigate websites, which creates inher-

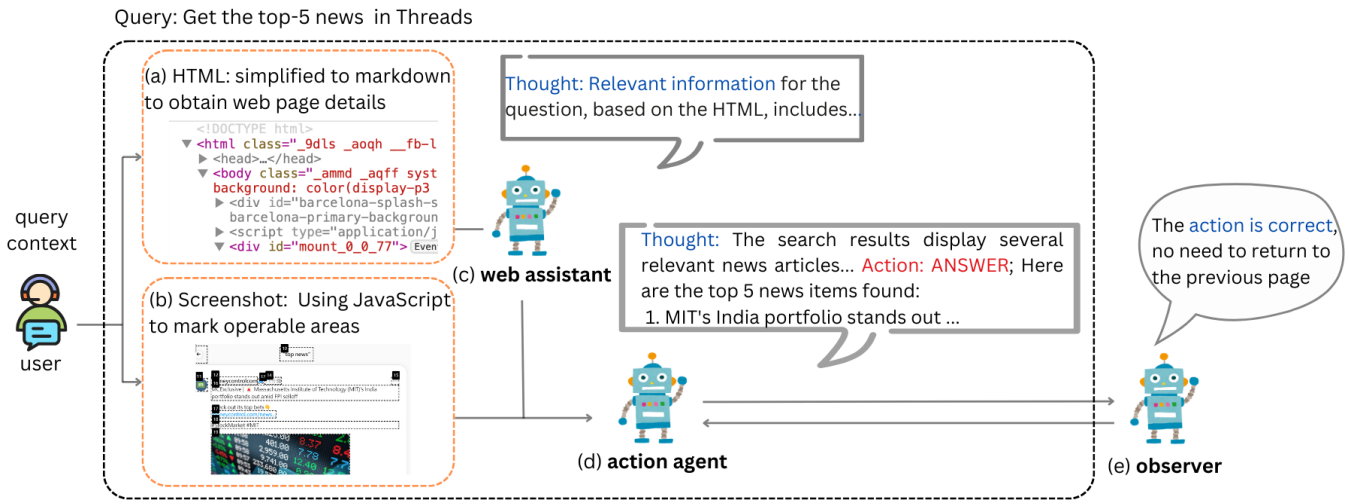

Figure 1: PagePilot architecture: We utilize (a) HTML Processing, (b) Screenshot Processing, and (c) Web Assistant modules to analyze the content of the web page. Next, we implement "Dynamic Loading" to expedite the processing of web pages. The (d) Action Agent module then takes over to perform actions within the browser. Throughout this process, the (e) Observer Agent module monitors and ensures that all actions are executed correctly.

ent limitations. When an agent needs information beyond the current screen, it must scroll multiple times—an inefficient and error-prone process. As the full webpage source code is available at the initial page load and contains comprehensive information beyond what's visible on screen, we augment this information to PagePilot agent for relevant information retrieval.

- Multi-Agent Oversight System: To relieve the system from stuck during navigation, we adopt a multi-agent design featuring an Action Agent that performs operations and an Observer Agent that monitors for navigation problems. When the Action Agent becomes stuck, the Observer Agent can detect this situation and implement corrective measures.
- Dynamic Content Loading Integration: While source code augmentation addresses some challenges, modern websites extensively use dynamic loading technologies that progressively reveal content. PagePilot incorporates specialized dynamic loading functionality that works in harmony with our source code augmentation.

Related Work

With the development of large language models (LLMs), there have been significant advancements in task planning, reasoning capabilities, code generation, and API calls, leading to increased research into the automation of LLM agents. The methods used can generally be divided into API calls, usage of Code and Bash tools, and simulated operations (Table 1). For instance, Gorilla (Patil et al. 2023) and ToolLLaMA (Qin et al. 2023) focus on interactions between LLMs and numerous APIs, completing tasks by invoking feature-rich predefined APIs. Meanwhile, Tulip Agent (Ocker et al. 2024) emphasizes the application and generation of Python tools to accomplish tasks through task decomposition and code generation. OS-Copilot (Wu et al. 2024) employs Python, Bash, and simulated keyboard and mouse operations to perform system-level tasks.

Visual Agents However, automation based on code and text input is difficult to apply to general and untrained interfaces or closed-source commercial software. Consequently, some research on Visual Agents employs visual input and simulates keyboard, mouse, or touch operations as output. These approaches are more adaptable to mobile devices (Android, iOS) and more operating systems (Linux, macOS, Windows). For instance, SeeClick uses pixel-level grounding to enable the model to interact with output pixels (Cheng et al. 2024); however, this method still faces issues with untrained icons and fine-grained control. CogAgent addresses this by training a Visual Language Model (VLM) that takes high-resolution images as input to achieve pixel-level control and visual question answering (Hong et al. 2024), yet it still encounters challenges with precision and generalization.

Automated Agent for Web Tasks Due to the complexity of automated operations, some research focuses on developing general automation agents for relatively simple web domains. These approaches include using code parsers or leveraging visual inputs through multimodal models. However, language models face challenges like hallucinations and incorrect outputs when dealing with image operations. SeeAct proposes alignment methods such as Grounding via Element Attributes, Grounding via Textual Choices, and Grounding via Image Annotation (Zheng et al. 2024), using HTML element tags, text tags, and image annotations to enable correct interaction between large language models (LLMs) and web pages. To create a general agent for real-world environments and address operational problems of language models, WebVoyager employs HTML and JS parsing and utilizes the Set-of-Mark method (Yang et al. 2023)

	Input		Interaction				Tasks/Environment
	text	vision	API	Bash	Code	Simulation	
Gorilla (Patil et al. 2023)	X		X	X			tasks with existing APIs
Tulip Agent (Ocker et al. 2024)	X		X		X		mathematics, robots
WebAgent (Gur et al. 2024b)	X				X		web tasks
Wilbur (Lutz et al. 2024)	X				X		web tasks
WebVoyager (He et al. 2024)	X[2]	X				X	web tasks
Browser Use(Müller and Žunič 2024)	X	X				X	web tasks
Manus(Manus AI 2025)	X	X		X	X	X	web tasks
OS-Copilot (Wu et al. 2024)	X	X[1]	X	X	X	X	computer tasks
CogAgent (Hong et al. 2024)	X	X				X	computer, mobile, web
SeeClick (Cheng et al. 2024)		X				X	computer, mobile, web
Claude3.5 Sonnet (Anthropic 2024)		X		X	X	X	computer tasks
PagePilot	X[3]	X				X	web tasks

[1] OS-Copilot uses OCR to convert images into text as input.
[2] WebVoyager uses vision and the HTML tag of the current field of view as input.
[3] PagePilot uses vision and complete HTML as input for processing.

Table 1: Related research on LLM agent automation, the general input method is text or image, and the operation methods from simple to complex are API call, Bash, Code, Simulate input (such as keyboard, mouse).

to digitally label interactive areas on web pages (e.g., input fields, buttons, scrollable sections) (He et al. 2024), allowing the language model to receive image inputs and output numerical labels with corresponding actions. This method outperforms the baseline performance of GPT-4v on commonly used English web pages but still faces challenges such as Navigation Stuck, Visual Grounding Issues, Hallucination, and Prompt Misalignment that need to be addressed.

Task Planning For automated agents, guiding models to plan, decompose, and complete tasks according to requirements remains a challenge. OS-Copilot uses directed acyclic graphs to plan task flows, dividing tasks into parallel subtasks to enhance execution efficiency (Wu et al. 2024). WebAgent predicts the next subtask based on HTML source code and task descriptions and interacts using the Python programming language (Gur et al. 2024b). WebVoyager, inspired by the ReACT method (Yao et al. 2023), requires language models to think before providing answers (He et al. 2024), thereby enhancing the agent's ability to analyze the environment and plan actions.Wilbur utilizes a retrieval method to learn from previous cases (Lutz et al. 2024), retrieving goal-specific and website-specific examples. These examples are synthesized through a language model to serve as references for generating actions. Additionally, a backtracking mechanism is employed to prevent repeating errors. Despite these methods, automated agents still encounter issues like Navigation Stuck and Prompt Misalignment, indicating that there is still room for improvement in task planning and execution.

Evolution of Language Models With the increasing number of parameters and training data in large language models, their ability to reason and analyze data continues to grow. CogAgent uses a vision-language model with 18 billion parameters, while GPT-3 features 175 billion parameters (Wei et al. 2024), and state-of-the-art models like GPT-4o have even more parameters. Therefore, developing a general automated agent based on pre-trained language models may enhance image processing and reasoning capabilities. Moreover, some models are natively equipped with the ability to operate computers. For instance, the latest Claude 3.5 Sonnet (Anthropic 2024) can accept multiple image inputs, task instructions, roles, etc., and its output can be in the form of Bash commands to simulate input and directly control a computer. This approach has made significant progress in general computer control but has room for improvement in tasks specifically related to web browser control.

Methods

Current web automation agents can be categorized into task-oriented and information extraction agents. The former operates on web pages based on task requirements, such as check-ins, posting messages, and shopping, demanding precise control over web elements. In contrast, the latter needs to read a large volume of web pages to identify key content, such as summarizing web pages, searching for information, and analyzing stances, requiring a comprehensive understanding of web content. Most current automation research focuses on task-oriented agents, while there is less research on information-extraction agents. Therefore, this paper will focus on information extraction agents, aiming to advance in the web mining field.

The architecture of our PagePilot web assistant, as depicted in Figure 1, is implemented using Selenium to simulate a normal browser environment and connect to the target website for a given task. We process the loaded website's source code (HTML) and its screenshot separately. Prior research typically relied solely on source code or visual information, resulting in incomplete webpage understanding and operational errors. Our architecture leverages both elements to enhance the web assistant's control over the webpage. The details of each module will be introduced below.

An example execution is shown in Figure 2, where the

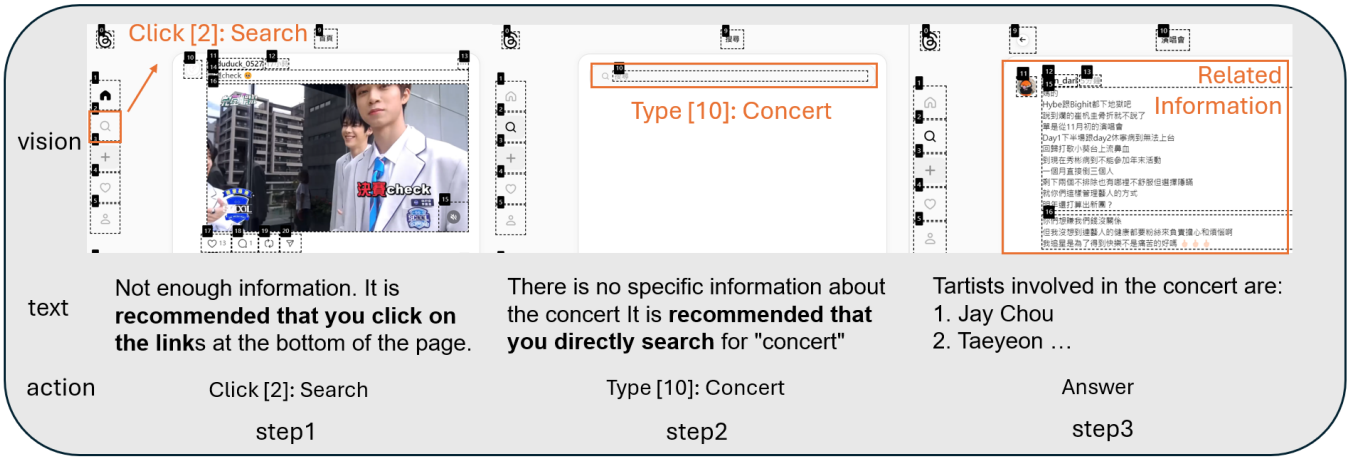

Figure 2: Example tasks - Thread , task is ”search concert, look at the top five news and tell me which artists you saw in concert”. The agent can receive suggestions from the source code of the web page, automatically operate the web page and obtain answers.

task involves searching for information on upcoming concerts and artists. Initially, upon entering the thread, neither visual nor textual information is sufficient to answer the query. At this point, the text assistant suggests search strategies. Once the search is completed, the text assistant analyzes the webpage source code to extract the required information, eliminating the need for step-by-step scrolling. This optimization allows for quickly finding answers, avoiding unnecessary browsing.

Multi-Agent Architecture Design

In view of the design of web automation in related studies, we adopted a Multi-Agent architecture similar to WebPilot (Zhang et al. 2024) (see Figure 1). Specifically, two dedicated agents are used to handle the web source code and the web screenshot, respectively. The web page source code is cleaned and dynamically loaded to ensure its completeness, after which the Web Assistant Agent extracts the task-related segments. For the processing of web page screenshots, we follow the approach of WebVoyager (He et al. 2024), marking relevant areas and passing them to the Action Agent to generate the next action. Together, these two components constitute the control section of the web assistant.

Previous studies have also demonstrated the importance of rollback mechanisms when errors occur (Lutz et al. 2024). Therefore, we introduce an Observer Agent that monitors the actions of the Action Agent as well as the responses from the environment to determine whether critical errors have occurred. When an error is detected, the Observer Agent provides a function to return to the previous page, thereby allowing the system to exit the current task loop. Thus, incorporating the Observer Agent into PagePilot achieves a balance between exploration and stability, enabling more effective handling of various web-based tasks.

Source Code Processing and Web Assistant

For the source code, we use the open-source package Crawler4ai to clean it and generate an intermediary Markdown file. Markdown is a lightweight markup language that can convert webpage text, links, and images into a structure that is beneficial for human and language model reading. It can remove irrelevant tags and windows from the webpage, retaining only content related to the topic. Then, in the web assistant steps, uses a language model to extract the parts relevant to the task. Filtering through the language model can significantly reduce the article’s length, avoiding the impact of excessive context length on action judgment. The web assistant’s output can be the answer to a question or suggestions such as scrolling through the webpage or clicking links.

Dynamic Loading

For information retrieval tasks, many websites employ dynamic loading techniques (such as posts on Facebook and Thread, or comments on YouTube, which only load when you scroll down). Using static web crawlers often results in retrieving only the first few pieces of content. When encountering such websites, PagePilot simulates user interactions by scrolling down to load more content. Experiments have shown that on YouTube, without any additional actions, comments are not loaded by default. However, by simulating dynamic loading, up to 50 comments can be loaded at once, significantly improving efficiency for information retrieval tasks.

Screenshot Processing

For processing webpage screenshots, we follow the visual workflow of WebVoyager(He et al. 2024), utilizing the Set-of-Mark method (Yang et al. 2023). This involves using JavaScript to mark interactive regions within the webpage and numbering them at the upper-left corner. This method facilitates easier interaction between the language model

and the webpage, avoiding issues related to precise location grounding.

Action Agent

After processing, the web page's text suggestions and screenshots are obtained, allowing the language model to determine the next course of action, including actions such as {click, wait, type, scroll, go back, google, answer}. The balance between text suggestions and screenshots represents a trade-off. When the emphasis is placed more on text suggestions, there's a risk of overlooking constraints such as price sorting, item selection, or specific requirements, as these settings typically rely on visual interactions. Conversely, when the focus is more on the visual aspect, it becomes necessary to continuously scroll through long web pages or data beyond the immediate view, requiring additional steps to browse the entire page.

Through experiments, we adopted a structure prioritizing visual input with supplementary text suggestions with the following prompt:

> **observation: [web assistant suggestion]** please analyze the attached screenshot (Currently visible range:0.00% - 42.29%) and give the Thought and Action. I've provided the tag name of each element and the text it contains ... **Please focus more on the screenshot and then refer to the textual information.**

The complete prompt is detailed in Appendix . In this structure, the language model first uses visual input to verify whether conditions are met, such as checking if options and ranges are selected or if the answer is in view. Only when full or partial verification is achieved it is considered as the final answer. This approach optimally utilizes both textual and visual information to achieve the task's objectives.

Observer Agent

Experimental data shows that when faced with complex operations, the action agent may perform repetitive or erroneous actions, such as clicking the wrong or irrelevant buttons or navigating to unrelated pages, leading the agent into a meaningless loop that persists until the action round ends. To address this, we introduced an observer agent that activates after the action agent performs four operations, assessing whether these steps are consistent with the mission. If they are aligned, the process continues; if not, the observer agent will prompt a return to the previous page to bypass infinite loops.

In the example shown in Figure 3, the action agent mistakenly clicks the login button, causing the model to get stuck on the login page. At this point, the observer agent determines that a return to the previous page is necessary, allowing the model to return to the task page and continue with the original mission.

Incorporating the observer agent enhances the model's ability to correct mistakes. While this may slightly increase the number of required actions (as performing additional actions is needed after returning to the previous page to complete the task), it effectively improves the task completion rate. According to the experimental results in Table 5, an 8% improvement can be achieved on the mind2web subset.

Experiments

For the evaluation methods in this experiment, we employed both manual evaluation and an inherited LLM automatic evaluation method from WebVoyager(He et al. 2024), with improvements specific to the architecture of PagePilot. In the evaluation process, in addition to using the dataset provided by WebVoyager and part of the GAIA dataset, we also extracted a dataset based on Mind2web(Deng et al. 2024). Additionally, for Chinese corpora, we established a task dataset based on Chinese language websites.

Mind2web Dataset

Mind2web is a dataset used to evaluate general web agents, collected from tasks on 137 real websites(Deng et al. 2024). It provides web page source code and task objectives to observe whether the agent's interaction trajectory successfully completes the task. Multimodal-Mind2Web is the multimodal version of the Mind2web dataset. We extracted 61 tasks from the test set for testing. Since Mind2web does not provide the web page URLs for the tasks, we consistently start from the official website of the task. Due to the uncertainty of automatic operations, we did not use Mind2web's evaluation method; instead, we applied the automatic evaluation method described in Section .

Chinese Dataset

We referenced the traffic of Taiwanese websites to construct a web testing dataset based on the Chinese language. These websites exclude those with reCAPTCHA, Cloudflare verifications, and login requirements, as current automated agents are still unable to bypass these robot verifications. The distribution of the websites is shown in Figure 4. The dataset covers domains such as information, news, shopping, social media, and YouTube (excluding videos). The tasks include web operations and information extraction, and the dataset has been expanded using language models. It consists of a total of 45 test data entries.

Evaluation Criteria

The manual evaluation focuses on webpage screenshots, text suggestions, action flows, and final answer judgments. The evaluation criteria are based on the degree of task completion, classified as {Failure, Partially Correct, Completely Correct}. The automated evaluation uses the GPT-4 Turbo language model, which is divided into {Failure, Correct} based on task completion. Along with the final answer, webpage screenshots and webpage Markdown serve as evidence to determine task completion, assisting the model in evaluating whether the requirements are met. Please refer to Appendix for the evaluation prompt. Each task undergoes evaluation three times to eliminate uncertainty.

Results and Analysis

The experimental evaluation results are as follows: For the Web Voyager dataset, the performance is presented in Ta-

Figure 3: Observer example, through the observer agent, the action agent's errors can be corrected. For example, skip the login page that enters incorrectly.

Figure 4: PagePilot Dataset composition

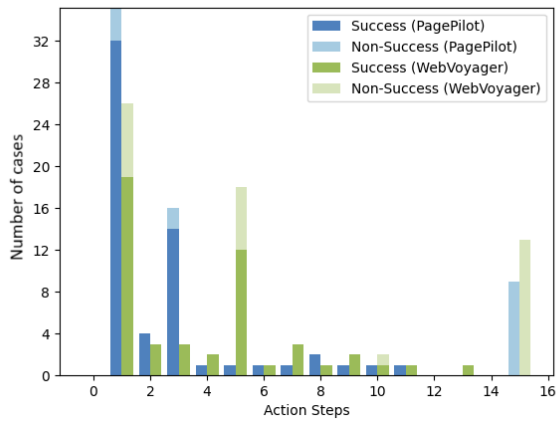

Figure 5: Success Rate by Action Steps on WebVoyager ArXiv subset. Dark colors are success cases, light colors are failure cases. PagePilot in the figure only requires a smaller number of actions and can achieve a higher success rate. The upper limit of the number of actions is 15, which will automatically terminate when reached.

ble 2. PagePilot shows improvements over the original versions in all tested web pages, significantly outperforming the GPT-4 baseline performance. Notably, for the ESPN website, it achieved enhancements of 33% and 45%. The results indicate that adding textual suggestions to the agent has minimal impact on the model's decision-making ability, allowing it to effectively complete tasks. For a detailed analysis of the ArXiv subset, as demonstrated in the success rate and step distribution Figure 5, PagePilot not only had a higher success rate but also required fewer average steps. This shows that textual suggestions can help avoid cumbersome screen interactions and quickly locate the appropriate content. Specifically, the task completion rate for Web Voyager was 65.1%, while PagePilot achieved 76.7%, with an average number of steps required to complete a task being 4.86 and 3.27, respectively. PagePilot was able to complete more tasks with fewer steps.

We also tested the GAIA subset selected by WebVoyager. The automatic analysis results are shown in Table 3. In both level 1 and level 2 tasks, PagePilot achieved significant improvements, surpassing the performance of GPT-4 and WebVoyager. On average, only 2.95 steps were needed to complete the tasks. In the Mind2web (Deng et al. 2024) subset, the automatic analysis results is that the task success rate was 51.7%, with an average of 4.2 steps required to complete a task. Since the Mind2web dataset requires more detailed operations (such as querying locations, addresses, order numbers, etc.), it is more challenging and requires more steps. Nevertheless, PagePilot maintained a 51.7% success rate.

For the custom Chinese corpus dataset, the experimental results are shown in Table 4. The results indicate that for information retrieval tasks in Chinese, PagePilot has a task completion rate of 69.5%. Manually verified annotations show that the success rate only slightly decreases to 60%, effectively handling most tasks and requiring an average of 3.7 action steps. Upon closer examination of the data, the manual evaluation of the original dataset showed a task completion rate of 65%, which is slightly lower compared to the automatic evaluation. Additionally, 15% of the tasks were partially correct. For datasets generated by LLM, the tasks were more challenging, with a success rate of 56% and 12% partially completed. The total task completion rate is 60%. The experimental results indicate that PagePilot still

	Allrecipes	Amazon	Apple	ArXiv	ESPN	Coursera
GPT-4 (All Tools)	11.1%	17.1%	44.2%	14.0%	31.8%	31.0%
WebVoyager$_{\text{GPT-4o}}$	56.3%	53.7%	56.6%	60.5%	44.0%	65.1%
Wilbur$_{\text{GPT-4 turbo}}$	60.0%	43.9%	60.5%	51.2%	59.1%	51.1%
PagePilot$_{\text{GPT-4o}}$	**64.4%**	**80.5%**	**76.7%**	**76.7%**	**77.3%**	**76.2%**
	Cambridge Dictionary	BBC News	Google Map	Google Search	Huggingface	Wolfram Alpha
GPT-4 (All Tools)	25.6%	9.5%	53.7%	60.5%	37.2%	52.2%
WebVoyager$_{\text{GPT-4o}}$	82.2%	54.8%	56.9%	63.6%	42.6%	65.2%
Wilbur$_{\text{GPT-4 turbo}}$	**86.0%**	**81.0%**	39.0%	67.4%	53.5%	65.2%
PagePilot$_{\text{GPT-4o}}$	83.7%	78.6%	**68.3%**	**90.7%**	**60.5%**	**82.6%**

Table 2: Evaluation on the WebVoyager dataset: WebVoyager and Wilbur were evaluated using GPT-4V, while PagePilot was evaluated using GPT-4 Turbo (since the former has been deprecated). The results for GPT-4, WebVoyager, and Wilbur are sourced from the original paper. The evaluation metric used is the task success rate. Among these, Booking and Google Flights cannot be reproduced due to date constraints, and GitHub cannot be reproduced for technical reasons.

Figure 6: Example task - NCU, task: Help me find the webpage of ”Research Center for Advanced Science and Technology”. In this example, the visual part was misled by the text suggestion and the link was not clicked correctly.

	GAIA level 1	GAIA level 2
GPT-4 (All Tools)	23.1%	12.5%
WebVoyager	38.5%	15.6%
PagePilot$_{\text{GPT-4o}}$	73.1%	41%

Table 3: Evaluation on GAIA subset

	Origin	LLM Expansion	Entire
Auto Evaluation	75%	64%	69.5%
Human Evaluate	65%	56%	60%

Table 4: Evaluation on Chinese dataset

tends to neglect certain instructions (such as selecting options) and encounters navigation errors.

It is worth noting that in the example of an information retrieval task, such as task-NCU (as shown in Figure 6), the visual model was misled by text suggestions and skipped clicking on the page. This resulted in only partial success of the task, highlighting the significant importance of the balance between visual and textual inputs.

Ablation Study

We conducted an ablation study to verify the functionality of different components of the system, as shown in Table 5. The results indicate that the basic system achieves only a 30% task completion rate, with an average of 5.28 steps required. When webpage markdown is integrated, the task completion rate significantly increases to 43%. Furthermore, incorporating the web assistant reduces the required operational steps to 4.46. This demonstrates that the combination of these two systems significantly enhances the effectiveness of the automated control system. Adding the dynamic load feature further elevates the task completion rate to 52% and reduces the operational steps to 4.16, aligning with previous predictions regarding the dynamic load feature. Additionally, the observer component can further increase the task completion rate to 60%, while slightly raising the number of actions to 4.81, indicating that the multi-agent architecture is beneficial for task processing.

Case Analysis

We analyze cases of automated tasks to study the performance of automation agents across various tasks. The observations are as follows:

Efficient in Form Filling: For tasks provided with sufficient information, agents can effectively fill out forms across

Methods	Success Rate	Avg Step
Basic	0.3	5.28
Basic+MD	0.43	6
Basic+MD+AS	0.43	4.46
Basic+MD+AS+DL	0.52	4.16
Basic+MD+AS+DL+OB	**0.6**	**4.81**

Table 5: Ablation study on part of the mind2web dataset (due to budget constraints, we use the first portion of the test domain as the evaluation standard), where MD stands for web page markdown, AS stands for web assistant, DL stands for dynamic loading, and OB stands for observer.

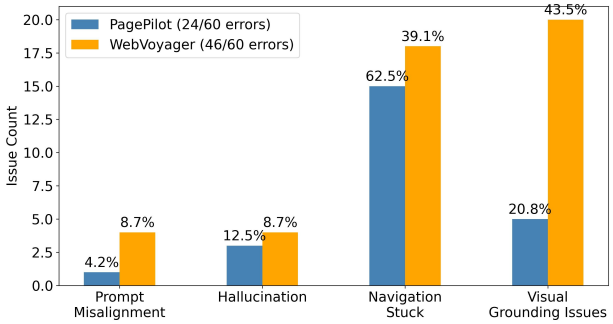

Figure 7: Error Rate Analysis on the Mind2Web Subset

multiple steps and handle the output results. For example, in the mind2web-19 task (Figure 8-a), the agent can input details such as animal, location, and distance progressively, then organize and output the results. This demonstrates that with clear instructions, agents can complete tasks involving data entry and extraction, such as shopping and information retrieval.

Efficient in Handling Long Articles: When it comes to lengthy articles, the Web Assistant can effectively extract relevant portions in response to queries, retaining sufficient information despite using minimal context. For instance, in the youtube-1 task (Figure 8-b), it can read 53 comments at once, classify them as supportive, neutral, opposing, and provide brief descriptions. This capability significantly reduces the time required for summarizing articles and performing sentiment analysis.

Limited by Complex Search Keywords: During searches, agents tend to incorporate item requirements directly into search keywords. For example, in the pchome24h-3 task (Figure 8-c), the search keyword used was "iPhone 16 Pro eSIM original warranty." The agent's inclusion of terms like eSIM and original warranty as search keywords led to unsuccessful product searches.

Challenges with Repetitive and Complex Actions: The decision-making process of automation agents is comparatively slower than humans, often requiring substantial input of images, prompts, and context for processing. Repetitive tasks, therefore, become expensive and less effective. For instance, in the huggingface-7 task (Figure 8-d), where the objective was to find the dataset with the highest download rate, the search page did not display download statistics, requiring manual entry into each dataset for comparison. Such tasks can confuse the model or exceed the action limit (15 actions) and thus forcefully terminate the task.

Error Analysis

We conducted an error case analysis for both PagePilot and WebVoyager on the Mind2Web subset. In our experiments, the PagePilot system produced 24 error cases, while WebVoyager generated 46. As illustrated in Figure 7, the majority of errors for WebVoyager fell into the categories of Navigation Stuck (39.1%) and Visual Grounding Issue (43.5%). In contrast, for PagePilot, most errors (62.5%) were due to Navigation Stuck, with a notably lower proportion of Visual Grounding Issues, attributable to the significant reduction in operation count. Navigation Stuck errors, which represent the majority in both systems, are often difficult to address at the architectural level because they are typically caused by webpage design limitations or ambiguous user prompts. Therefore, improving web navigation will remain a critical challenge for future research in web automation.

Model and System Scalability

To validate the performance of the PagePilot architecture against other models and open-source models, we conducted tests on the LLAMA and Gemini models (see Table 6). In the relatively straightforward GAIA level 1 dataset, Gemini 2.5 Pro performed well with a task completion rate of 61.5%, closely approaching GPT-4o, while LLAMA4 Maverick and Scout both achieved a task completion rate of 50%, slightly below GPT-4o. In the more complex GAIA level 2 dataset, Gemini 2.5 Pro and LLAMA 3.1 Nemotron Ultra both demonstrated strong performance, with task completion rates nearing that of GPT-4o.

The above experiments demonstrate that PagePilot has the capability to run on open-source language models, requiring only that the model possesses image input capabilities. The performance of the current open-source models, LLAMA 4 and LLAMA 3.1 Nemotron, showed only slight declines, which is sufficient to tackle complex tasks. Additionally, it exhibits good scalability for future, more advanced open-source models.

In terms of system scalability, although PagePilot is currently designed primarily for research purposes and has not been optimized for production environments, it can only handle a single request at a time. However, we believe that by adopting distributed systems or implementing multi-process techniques, the system can be scaled to accommodate multiple user requests concurrently. Additionally, integrating a web operation interface could enable functionalities similar to those provided by the Manus system. Further development in these areas is left for future work.

Cost Analysis

We further analyze the operational costs of the PagePilot and WebVoyager systems. The results are summarized in Table 7. Specifically, PagePilot employs GPT-4o for the Action Agent and Gemma3 for both the WebAssistant and Observer

Method	Model	Params	GAIA level1		GAIA level2	
			Success Rate	Action Step	Success Rate	Action Step
BrowserUse	GPT-4o	-	73.1	5.16	**48.4**	6.23
PagePilot	GPT-4o	-	**73.1**	**2.53**	45.3	3.38
	Gemini 2.5 Pro Exp	-	61.5	3.56	40.6	4.35
	Llama3.1 Nemotron Ultra	253B	42.3	2.64	34.4	3.73
	Llama4 maverick	400B	50	4.92	29.7	**2.79**
	Llama4 scout	109B	50	4.38	29.7	3.74

Table 6: Model Scalability Evaluation on GAIA subset

Method	Action Agent	Web Assistant& Observer Agent	Success Rate	Action Step	Cost (USD)
PagePilot	GPT-4o	Gemma3-27b-it-pt[1]	0.53	5.22	20.18341
WebVoyager	GPT-4o	-	0.23	4.5	18.48419

[1] Gemma3 uses pre-trained open source models, which are not included in the cost

Table 7: Cost Analysis in Mind2Web Subset

Agents to optimize cost efficiency, while WebVoyager utilizes only the GPT-4o model. The results demonstrate that PagePilot achieves a 53% task completion rate with nearly the same cost as WebVoyager, which attains only a 23% completion rate. This significant improvement highlights the cost-effectiveness and advantage introduced by the Multi-Agent architecture adopted in PagePilot.

Conclusion

This study explored the performance of the LLM agent in automating tasks, including computer usage and problem-solving, with a specific focus on web page control. Building on previous research, the PagePilot framework was proposed, effectively integrating web page images and source code. By utilizing dynamic loading capabilities, the LLM agent can perform automated controls to complete designated tasks. We validated its feasibility on datasets such as web voyager, GAIA, mind2web, as well as our self-constructed Chinese dataset. The experiments demonstrated that the task completion rate could be increased by 22% with 27% fewer operations, thus setting a new standard for web automation tasks. Case analyses showed that while the PagePilot agent can efficiently handle common tasks like form filling and information extraction, there is room for improvement in handling repetitive and complex tasks. However, this aspect is related to the agent's reasoning and context processing abilities, which remain areas for future research.

Limitation

PagePilot is an automated agent based on large language models, with its analytical and decision-making capabilities relying on externally provided language models. There can be significant differences depending on the model used. For example, differences in context length can affect the amount of context or number of images that can be processed. The ability to analyze environments and make plans also varies. Thus, the feasibility of integrating other open-source language models is still subject to testing and optimization.

Since it requires external models, there are some limitations in automation capabilities, such as logging in with a username and password, handling private information, or viewing confidential data. Additionally, operational security must be considered, and users are advised to monitor personal account activities, such as posting, commenting, or filling out forms. Therefore, it is currently recommended to use the automated agent only in clean and controllable environments.

Ethical Considerations

The purpose of this study is to explore the feasibility of automated web agents. When deploying such agents, it is essential to ensure that automated actions are permitted by the target websites and are not leveraged for inappropriate activities, such as expressing specific political views or initiating abnormal accesses. Due to the ease of automation, both intentionally malicious behaviors by humans and those inadvertently caused by language models become more likely. Furthermore, given the unpredictability of language models, even when provided with comprehensive task objectives, the interactions between agents and webpages must be monitored in real time to prevent unintended behaviors. Examples include unexpected account operations, unverified purchases, or incorrect form submissions, all of which require human supervision and timely intervention.

Acknowledgements

The research is partially supported by the National Science and Technology Council, Taiwan, under grant NSTC-114-2222-E-008-005-MY3.

Click [3]: Cats | Type [5]: Location, "94587, CA" | Click [5]: Distance, "100 miles"

(a) Mind2web-19, Find cats available for adoption with 10 miles of zip code 94587, Young or Adult age cat, sorted by Oldest Addition

Scroll Down | Scroll Down | Answer: Comments...

(b) Youtube-1, Please analyze the position of the comments below the video, and list the number of supporting and opposing comments and a brief description.

Type [8]: "iPhone 16 Pro eSIM original warranty" | Click [10]: 手機/平板 | Answer: 找不到iPhone 16 Pro eSIM original warranty

(c) Pchome24h-3, I'm looking for an iPhone 16 pro that supports esim and provides original warranty. Tell me the price.

Type [1]: Search, "audio datasets" | Scroll Down and Click | Answer: 找到syntheory dataset

(d) Huggingface-7, Which is the most downloaded audio related dataset on Hugging face currently.

Figure 8: Case Study

References

Anthropic. 2024. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku.

Cheng, K.; Sun, Q.; Chu, Y.; Xu, F.; YanTao, L.; Zhang, J.; and Wu, Z. 2024. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 9313–9332. Bangkok, Thailand: Association for Computational Linguistics.

Deng, X.; Gu, Y.; Zheng, B.; Chen, S.; Stevens, S.; Wang, B.; Sun, H.; and Su, Y. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

FORCE11. 2020. The FAIR Data principles. https://force11.org/info/the-fair-data-principles/.

Gebru, T.; Morgenstern, J.; Vecchione, B.; Vaughan, J. W.; Wallach, H.; Iii, H. D.; and Crawford, K. 2021. Datasheets for datasets. *Communications of the ACM*, 64(12): 86–92.

Gur, I.; Furuta, H.; Huang, A. V.; Safdari, M.; Matsuo, Y.; Eck, D.; and Faust, A. 2024a. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Gur, I.; Furuta, H.; Huang, A. V.; Safdari, M.; Matsuo, Y.; Eck, D.; and Faust, A. 2024b. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

He, H.; Yao, W.; Ma, K.; Yu, W.; Dai, Y.; Zhang, H.; Lan, Z.; and Yu, D. 2024. WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 6864–6890. Bangkok, Thailand: Association for Computational Linguistics.

Hong, W.; Wang, W.; Lv, Q.; Xu, J.; Yu, W.; Ji, J.; Wang, Y.; Wang, Z.; Dong, Y.; Ding, M.; et al. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14281–14290.

Lutz, M.; Bohra, A.; Saroyan, M.; Harutyunyan, A.; and Campagna, G. 2024. WILBUR: Adaptive In-Context Learning for Robust and Accurate Web Agents. arXiv:2404.05902.

Manus AI. 2025. https://manus.im/. Accessed: 2025-06-02.

Müller, M.; and Žunič, G. 2024. Browser Use: Enable AI to control your browser.

Nguyen, D.; Chen, J.; Wang, Y.; Wu, G.; Park, N.; Hu, Z.; Lyu, H.; Wu, J.; Aponte, R.; Xia, Y.; Li, X.; Shi, J.; Chen, H.; Lai, V. D.; Xie, Z.; Kim, S.; Zhang, R.; Yu, T.; Tanjim, M.; Ahmed, N. K.; Mathur, P.; Yoon, S.; Yao, L.; Kveton, B.; Kil, J.; Nguyen, T. H.; Bui, T.; Zhou, T.; Rossi, R. A.; and Dernoncourt, F. 2025. GUI Agents: A Survey. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Findings of the Association for Computational Linguistics: ACL 2025*, 22522–22538. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-256-5.

Ocker, F.; Tanneberg, D.; Eggert, J.; and Gienger, M. 2024. Tulip Agent – Enabling LLM-Based Agents to Solve Tasks Using Large Tool Libraries. arXiv:2407.21778.

Patil, S. G.; Zhang, T.; Wang, X.; and Gonzalez, J. E. 2023. Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint arXiv:2305.15334*.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; Zhao, S.; Tian, R.; Xie, R.; Zhou, J.; Gerstein, M.; Li, D.; Liu, Z.; and Sun, M. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. arXiv:2307.16789.

Rawles, C.; Li, A.; Rodriguez, D.; Riva, O.; and Lillicrap, T. 2024. Android in the wild: a large-scale dataset for android device control. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23. Red Hook, NY, USA: Curran Associates Inc.

Wei, C.; Wang, Y.-C.; Wang, B.; and Kuo, C.-C. J. 2024. An Overview of Language Models: Recent Developments and Outlook. *APSIPA Transactions on Signal and Information Processing*, 13(2).

Wu, Z.; Han, C.; Ding, Z.; Weng, Z.; Liu, Z.; Yao, S.; Yu, T.; and Kong, L. 2024. OS-Copilot: Towards Generalist Computer Agents with Self-Improvement. arXiv:2402.07456.

Yang, J.; Zhang, H.; Li, F.; Zou, X.; Li, C.; and Gao, J. 2023. Set-of-Mark Prompting Unleashes Extraordinary Visual Grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

Zhang, C.; He, S.; Qian, J.; Li, B.; Li, L.; Qin, S.; Kang, Y.; Ma, M.; Liu, G.; Lin, Q.; Rajmohan, S.; Zhang, D.; and Zhang, Q. 2025. Large Language Model-Brained GUI Agents: A Survey. arXiv:2411.18279.

Zhang, Y.; Ma, Z.; Ma, Y.; Han, Z.; Wu, Y.; and Tresp, V. 2024. WebPilot: A Versatile and Autonomous Multi-Agent System for Web Task Execution with Strategic Exploration. arXiv:2408.15978.

Zheng, B.; Gou, B.; Kil, J.; Sun, H.; and Su, Y. 2024. GPT-4V(ision) is a Generalist Web Agent, if Grounded. In *Forty-first International Conference on Machine Learning*.

Zhou, S.; Xu, F. F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Ou, T.; Bisk, Y.; Fried, D.; Alon, U.; and Neubig, G. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *The Twelfth International Conference on Learning Representations*.

Paper Checklist

1. For most authors...

(a) Would answering this research question advance science without violating social contracts, such as violating privacy norms, perpetuating unfair profiling, exacerbating the socio-economic divide, or implying disrespect to societies or cultures? Yes.

(b) Do your main claims in the abstract and introduction accurately reflect the paper's contributions and scope? Yes.

(c) Do you clarify how the proposed methodological approach is appropriate for the claims made? Yes.

(d) Do you clarify what are possible artifacts in the data used, given population-specific distributions? NA

(e) Did you describe the limitations of your work? Yes.

(f) Did you discuss any potential negative societal impacts of your work? Yes, see Ethical Considerations section.

(g) Did you discuss any potential misuse of your work? Yes, see Ethical Considerations section.

(h) Did you describe steps taken to prevent or mitigate potential negative outcomes of the research, such as data and model documentation, data anonymization, responsible release, access control, and the reproducibility of findings? Yes.

(i) Have you read the ethics review guidelines and ensured that your paper conforms to them? Yes.

2. Additionally, if your study involves hypotheses testing...

(a) Did you clearly state the assumptions underlying all theoretical results? NA

(b) Have you provided justifications for all theoretical results? NA

(c) Did you discuss competing hypotheses or theories that might challenge or complement your theoretical results? NA

(d) Have you considered alternative mechanisms or explanations that might account for the same outcomes observed in your study? NA

(e) Did you address potential biases or limitations in your theoretical framework? NA

(f) Have you related your theoretical results to the existing literature in social science? NA

(g) Did you discuss the implications of your theoretical results for policy, practice, or further research in the social science domain? NA

3. Additionally, if you are including theoretical proofs...

(a) Did you state the full set of assumptions of all theoretical results? NA

(b) Did you include complete proofs of all theoretical results? NA

4. Additionally, if you ran machine learning experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? Yes, see the Methods and Appendix section.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? Yes, This experiment did not involve training model, other parameters can be referenced in the methods and appendix sections.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? No, We eliminated uncertainty through repeated experiments and consensus.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? Yes, We only used the public APIs described in the article.

(e) Do you justify how the proposed evaluation is sufficient and appropriate to the claims made? Yes, see the Evaluations section.

(f) Do you discuss what is "the cost" of misclassification and fault (in)tolerance? Yes, see the Limitation section.

5. Additionally, if you are using existing assets (e.g., code, data, models) or curating/releasing new assets, **without compromising anonymity**...

(a) If your work uses existing assets, did you cite the creators? Yes. The foundational code for this experiment is derived from WebVoyager, and all other packages comply with their usage agreements. Please see the Methods and Reference section.

(b) Did you mention the license of the assets? Yes, WebVoyager is licensed under the Apache-2.0 license, and we have conducted secondary development based on it.

(c) Did you include any new assets in the supplemental material or as a URL? Yes, We have published the PagePilot code in comments.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? No, This experiment only uses public datasets.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? No, This experiment does not include personal or sensitive information.

(f) If you are curating or releasing new datasets, did you discuss how you intend to make your datasets FAIR (see FORCE11 (2020))? Yes, The dataset will be made available online using a standard format and will be reproducible.

(g) If you are curating or releasing new datasets, did you create a Datasheet for the Dataset (see Gebru et al. (2021))? Yes, Datasheet will be included in the public link.

6. Additionally, if you used crowdsourcing or conducted research with human subjects, **without compromising anonymity**...

(a) Did you include the full text of instructions given to participants and screenshots? NA

(b) Did you describe any potential participant risks, with mentions of Institutional Review Board (IRB) approvals? NA

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? NA

(d) Did you discuss how data is stored, shared, and deidentified? NA

Web Assistant Prompt

```
You are a helpful assistant who analyzes web content and gives answers appropriate to a given task. Please strictly check whether you meet the task requirements before answering. If there is not enough information, answer \"Insufficient information\" and provide suggestions, such as clicking on the link at the bottom of the page, etc.

web content:{webtext_md}
task:{task}
```

Action Agent Prompt

```
Observation:
web assistant:Sorry, I cannot directly assist you in making an appointment for your passport application. However, you can visit the [USPS official website](https://www.usps.com) and use the tools on the website to find the nearest post office and make an appointment to apply for a passport. Typically, you will need to provide personal information and select your available times. I recommend checking mission times in advance to ensure a successful appointment. If you have any questions, you can contact the Postal Service for additional assistance.
prompt: please analyze the attached screenshot (Currently visible range :0.00% - 42.29%) and give the Thought and Action. I've provided the tag name of each element and the text it contains (if text exists). Note that <textarea> or <input> may be textbox, but not exactly. Please focus more on the screenshot and then refer to the textual information.
tags:[0]: "Current language: English"; [1]: "Locations"; [2]: "Support"; [3]: "Informed Delivery"; [4]: " Register / Sign In"; [6]: "Quick Tools"; [7]: "Send"; [8]: "Receive"; [9]: "Shop"; [10]: "Business"; [11]: "International"; [12]: "Help"; [13]: "Search USPS.com"; [14]: <button> " Use Current Location"; [15]: <input> ""; [16]: <button> "Post Office National Retailer Collection Boxes Contract Postal Unit Self-Service Kiosks gopost Village Post Office"; [17]: <button> "Search"; [18]: < button> "Passport Appointments"; [19]: <button> "Filter \& Sort 1"; [23]: <button> "Street", "Map street view"; [24]: <button> "Satellite", " Map satellite view"; [25]: <button> "+", "Map zoom in"; [26]: <button> "-", "Map zoom out"; [27]: <button> " Feedback";
```

Observer Agent Prompt

```
SYSTEM PROMPT: You are an observer evaluating an agent that is performing a web page control task. Your goal is to determine whether the agent is following the correct step-by-step execution.
Base your judgment only on the current step, without predicting future actions.
Answer YES if the agent's action aligns with the expected task flow.
Answer NO only if the agent is significantly deviating from the task goal, such as repeating unnecessary actions or entering an incorrect loop. If you answer NO, provide a brief explanation and suggestions for correction.
This evaluation should focus on whether the agent is making meaningful progress rather than minor inefficiencies.
Output format: {YES/NO}, Suggestion: <your suggestion here>
action trajectory:{action trajectory}
task is {task}, Please say YES or NO first, and then say your suggestion.
```

Auto Evaluation Prompt

```
SYSTEM_PROMPT
As an evaluator, you will be presented with three primary components to assist you in your role:
1. Web Task Instruction: This is a clear and specific directive provided in natural language, detailing the online activity to be carried out. These requirements may include conducting searches, verifying information, comparing prices, checking availability, or any other action relevant to the specified web service (such as Amazon, Apple, ArXiv, BBC News, Booking etc).
2. Web Page Markdown: This is the content cleaned from the source code of the web page, showing the current status of the web page. It serves as textual proof of the actions taken in response to the instruction.
```

```
3. Result Screenshots: This is a visual representation of the screen showing the result or intermediate state of performing a web task. It serves as visual proof of the actions taken in response to the instruction.
4. Result Response: This is a textual response obtained after the execution of the web task. It serves as textual result in response to the instruction.
You DO NOT NEED to interact with web pages or perform actions such as booking flights or conducting searches on websites.
You SHOULD NOT make assumptions based on information not presented in the screenshot when comparing it to the instructions.
Your primary responsibility is to conduct a thorough assessment of the web task instruction against the outcome depicted in the screenshot and in the response, evaluating whether the actions taken align with the given instructions.
NOTE that the instruction may involve more than one task, for example, locating the garage and summarizing the review. Failing to complete either task, such as not providing a summary, should be considered unsuccessful.
NOTE that the screenshot is authentic, but the response provided by LLM is generated at the end of web browsing, and there may be discrepancies between the text and the screenshots.
Note the difference: 1) Result response may contradict the screenshot, then the content of the screenshot prevails, 2) The content in the Result response is not mentioned on the screenshot, choose to believe the content.
You should elaborate on how you arrived at your final evaluation and then provide a definitive verdict on whether the task has been successfully accomplished, either as 'SUCCESS' or 'NOT SUCCESS'.
USER_PROMPT
TASK: <task> Web Page Markdown: <markdown> Result Response: <answer> <num> screenshots at the end: <screenshot>
```