# INFOGENT: An Agent-Based Framework for Web Information Aggregation

Anonymous ACL submission


## Abstract

Despite seemingly performant web agents on the task-completion benchmarks, most existing methods evaluate the agents based on a presupposition: the web navigation task consists of linear sequence of actions with an end state that marks task completion. In contrast, our work focuses on web navigation for *information aggregation*, wherein the agent must explore different websites to gather information for a complex query. We consider web information aggregation from two different perspectives: (i) *Direct API-driven Access* relies on a text-only view of the Web, leveraging external tools such as Google Search API to navigate the web and a scraper to extract website contents. (ii) *Interactive Visual Access* uses screenshots of the webpages and requires interaction with the browser to navigate and access information. Motivated by these diverse information access settings, we introduce INFOGENT, a novel modular framework for web information aggregation involving three distinct components: Navigator, Extractor and Aggregator. Experiments on different information access settings demonstrate INFOGENT beats an existing SOTA multi-agent search framework by 7% under Direct API-Driven Access on FRAMES, and improves over an existing information-seeking web agent by 4.3% under Interactive Visual Access on AssistantBench.


## 1 Introduction

Despite the well-documented success of autonomous web agents (Nakano et al., 2021; Yang et al., 2023; Zhou et al., 2023; Deng et al., 2024), the proposed tasks usually perform goal-oriented web-based tasks involving navigating within a website, interacting with elements like buttons and executing complex workflows. (e.g., booking a flight or scheduling a meeting). However, a critical aspect of web-based tasks, *information aggregation* has received relatively less attention. Tasks involving gathering and presenting relevant data from diverse web sources are central to many real-world applications. For instance, humans often visit multiple websites, using search engines to find relevant content, and browsing articles, reviews, or forums.

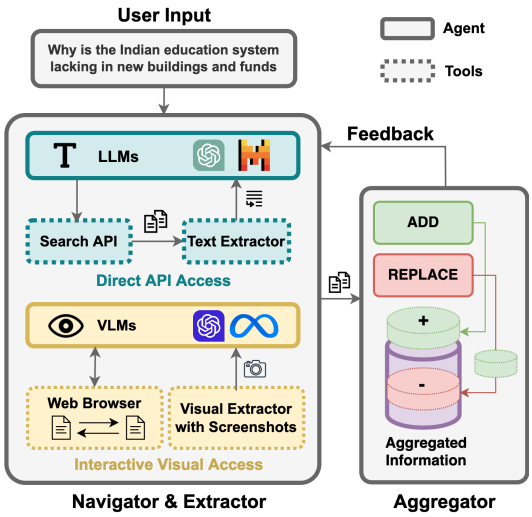


Figure 1: Overview of INFOGENT under the *Direct API Access* and *Interactive Visual Access* settings: The Navigator uses a tool-based LLM and a browser-controlling VLM as the web agent respectively, with the Aggregator's textual feedback guiding further navigation.

Existing web navigation benchmarks and methods (Zhou et al., 2023; Deng et al., 2024; Lù et al., 2024; Zheng et al., 2024b; Koh et al., 2024) primarily focus on linear, goal-oriented tasks, such as booking a flight from Chicago to London, where sequential actions lead directly to a predefined outcome without significant backtracking or exploration. These approaches address tasks with clear, predefined goals but overlook the challenge of aggregating information from multiple sources. In contrast, open-ended information-seeking tasks, such as investigating why the Indian education system lacks funding and infrastructure, require agents to explore multiple sources, consider diverse viewpoints, and determine when sufficient information has been gathered for a comprehensive answer.

Building agents for information-seeking tasks

shifts the focus from linearized action sequences for goal completion to the quality and coverage of the aggregated information, highlighting a gap in current methods that do not consider such exploratory behaviors. Specifically, we identify two critical limitations in current web agents: (1) *Lack of Information Aggregation*: they cannot aggregate information from multiple webpages; and (2) *Inability to Backtrack*: they are constrained to forward navigation, unable to revisit previous pages or explore alternative search results. These constraints hinder their effectiveness in information-seeking tasks that require iterative exploration.

Motivated by the challenges, we introduce INFOGENT, a novel framework for information aggregation on the web which accomplishes the task using three specialized components: a *Navigator* responsible for searching the web and identifying relevant websites, an *Extractor* for identifying relevant information from the selected web pages, and an *Aggregator* for selectively retaining the extracted information, and deciding what to include in the final aggregated output. To address the shortcomings of the current web agents, INFOGENT augments a task-completion agent with additional capabilities required to be an effective web navigator for information aggregation. Specifically, we introduce two key modifications: (1) *Enhanced Action Set* that enables the navigator to backtrack and transfer control to other components when aggregation is to be performed; and (2) *Feedback-Driven Navigation*, where navigator's decision-making process incorporates feedback from aggregator, ensuring that navigation strategies are dynamically informed by both the input query and the current state of information aggregation.

INFOGENT is modular, with a clear division of responsibilities between the three components, designed to operate effectively in real-world information aggregation settings. Specifically, we address two scenarios for accessing information from websites: *Direct API-driven Access*, where agents are enabled access only to the textual web data extracted via APIs without visual interaction, and *Interactive Visual Access*, which requires agents to simulate visually-dependent human browsing to access web information, which can often be obstructed by paywalls, logins, or other necessary user interactions that can only be bypassed with visual context understanding. By evaluating on realistic, multi-website aggregation tasks–AssistantBench (Yoran et al., 2024), FRAMES (Krishna et al., 2024) and FanOutQA (Zhu et al., 2024)–we demonstrate INFOGENT's ability to effectively handle both information access settings. In summary, our contributions are as follows:

- We introduce INFOGENT, a novel modular and feedback-driven framework for web information aggregation through the use of three distinct components: Navigator, Extractor, and Aggregator (illustrated in Figure 1).
- We demonstrate that INFOGENT can be employed under both *Direct API-Driven* Access and *Interactive Visual Access* settings.
- On various web aggregation tasks, we empirically show that INFOGENT outperforms existing state-of-the-art multi-agent search frameworks and information-seeking web agents.

## 2 Related work

**Web Navigation with LLMs:** Web navigation agents were originally explored in simulated web environments (Shi et al., 2017) and (Liu et al., 2018) which predominantly focused on completing goal-oriented tasks. The simulated environments came equipped with a range of task primitives such as selecting value from a drop down or entering text into an input box, which could be used to achieve the end goal. Subsequent work has focused on extending to more realistic settings (Nakano et al., 2021), such as WebShop (Yao et al., 2022) for e-commerce and RUSS (Xu et al., 2021) for web support. However, these efforts are still limited to a narrow set of domains and websites. In contrast, WebArena (Zhou et al., 2023) and Mind2Web (Deng et al., 2024) were introduced as benchmarks for autonomous web agents that can generalize to a wide variety of tasks on real-world websites. Nevertheless, these approaches were still limited to predominantly language-guided agents, that solely relied on the text elements present within the website raw HTML. Follow-up works, such as VisualWebArena (Koh et al., 2024), SeeACT (Zheng et al., 2024a) and WebVoyager (He et al., 2024), use multimodal agents (Achiam et al., 2023; Team et al., 2023) that leverage screenshots of websites as input for identifying the appropriate HTML elements to act upon. The motivation is that raw HTML contents are too noisy, and context is often too long, while screenshots provide a less noisier view of the webpage. While these methods involve an autonomous agent solving the task using an initial instruction, more recently, WebLinx (Lù et al.,

2024) introduces the problem of *conversational web navigation*, wherein the agent controls a real-world web browser and follows user instructions to solve tasks in a multi-turn dialogue fashion.

**Web Information Aggregation:** Recently, there has been a growing interest for more complex information aggregation tasks, which have been studied independently within the Information Extraction field (Reddy et al., 2023). In the context of Web Agents, information aggregation requires broader exploration and backtracking to effectively generate the answer. MindSearch (Chen et al., 2024) explores this, modeling the task as an iterative graph construction. AssistantBench (Yoran et al., 2024) enhances SeeAct with the go back action and a planning module, and tackles time consuming tasks on the web. In this work, we propose INFOGENT, a modular framework featuring specialized aggregation and feedback modules that achieves state-of-the-art performance in both Direct API-driven Access and Interactive Visual Access scenarios.

## 3 Information Aggregation Task

We conceptualize information aggregation for a query as an iterative process involving identifying relevant websites and gathering pertinent information within them, repeated until sufficient data is collected. Actively tracking the aggregated information guides subsequent searches, ensuring comprehensiveness while avoiding redundancy. The success of the process is dependent on the quality and diversity of the collected information.

We note that accessibility of web information varies significantly. Some data is easily obtainable through APIs or by scraping web pages (e.g., retrieving "Billboard Top 100 songs" from Wikipedia ). However, other information, such as salary data on Glassdoor, is not directly accessible due to paywalls, or other restrictions. Therefore, we categorize information aggregation tasks into two settings based on the type of access: *Direct API-Driven Access* and *Interactive Visual Access*.

The former involves retrieving data via APIs or automated tools without interacting with the website, making it efficient when APIs are available. In contrast, Interactive Visual Access requires simulating human browsing to retrieve information from screenshots of webpages that prohibit automatic scraping. We hypothesize that these two approaches together encompass a wide range of practical scenarios for information aggregation, and any comprehensive solution should handle both paradigms. Moreover, while we primarily focus on web-based aggregation, the concept of Interactive Visual Access extends to other desktop and mobile applications, such as Slack or iMessage, where API access is restricted and visual interaction is necessary (Ge et al., 2023; Kapoor et al., 2024).

**Alg. 1:** Information Aggregation with INFOGENT

**Input:** $\mathcal{T}$: User Task, $K$: Max websites to extract, $N$: Max time steps
**Output:** $\mathcal{S}$: Aggregated information stack
$\mathcal{W}_0 \leftarrow$ "Search Home" // Starting Webpage
$\mathcal{S}_0 \leftarrow$ "Empty Stack" // Information Stack
$\mathcal{F} \leftarrow$ "None" // Aggregator Feedback
$k \leftarrow 0$ // Iteration Counter
$t \leftarrow 0$ // Action Counter
**while** $a_t! =$ TERMINATE ***and*** $k < K$ ***and*** $t < N$ **do**
  $a_{t+1} = \mathcal{NG}(\mathcal{W}_t, \mathcal{T}, \mathcal{F}, \{a_1, a_2, \ldots, a_t\})$
  **if** $a_{t+1} =$ AGGREGATE **then**
    $\mathcal{P} \leftarrow \mathcal{ET}(\mathcal{W}_t, \mathcal{T}, \mathcal{F})$ // Extract Info.
    $\mathcal{S}_{k+1}, \mathcal{F} \leftarrow \mathcal{AG}(\mathcal{S}_k, \mathcal{P}, \mathcal{T})$ // Update $\mathcal{S}_k$
    $k \leftarrow k + 1$ // Update Counter
    $W_{t+1} \leftarrow W_t$
  **end**
  **else**
    $\mathcal{W}_{t+1} = \text{Act}(\mathcal{W}_t, a_{t+1})$ // Make Action
  **end**
  $t \leftarrow t + 1$ // Update Counter
**end**

## 4 INFOGENT

INFOGENT, as illustrated in Fig. 1, consists of three core components: A *Navigator* $\mathcal{NG}$, an *Extractor* $\mathcal{ET}$, and an *Aggregator* $\mathcal{AG}$. Given an information-seeking query, the Navigator $\mathcal{NG}$ initiates the process by searching the web for relevant sources. Upon identifying a suitable webpage, the Extractor $\mathcal{ET}$ takes over the control, which extracts relevant content and forwards it to the Aggregator $\mathcal{AG}$. $\mathcal{AG}$ evaluates this content with respect to the information aggregated so far and decides whether to include it. Importantly, $\mathcal{AG}$ provides feedback to $\mathcal{NG}$ about gaps in the aggregated information, guiding subsequent searches to address deficiencies. $\mathcal{NG}$ lacks direct access to the aggregated information, thereby relies on $\mathcal{AG}$'s feedback for directions in subsequent iterations. This iterative process continues until $\mathcal{AG}$ determines that sufficient information has been gathered and instructs $\mathcal{NG}$ to halt. Thus, INFOGENT employs a modular, feedback-driven approach to information aggregation, making it suitable for complex queries requiring diverse sources. Fig. 2 illustrates the feedback-driven navigation with example.

Let's denote the action space of $\mathcal{NG}$ as $\mathcal{A}$, the

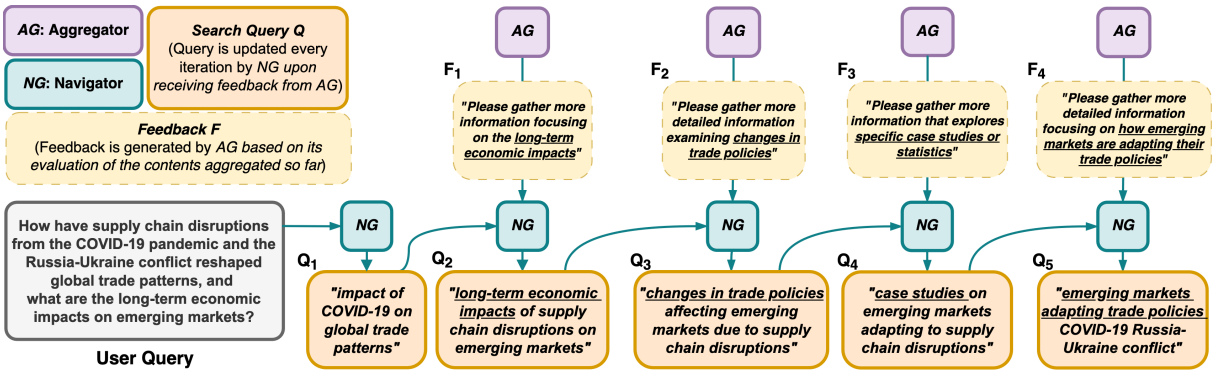


Figure 2: A working example of INFOGENT. $\mathcal{NG}$ iteratively generates an updated query given feedback from $\mathcal{AG}$.

task at hand as $\mathcal{T}$, the Aggregator feedback as $\mathcal{F}$ and the current website under consideration as $\mathcal{W}$. Further, there is a stack $\mathcal{S}$ of diverse information aggregated in the form of a list of paragraphs, which is returned upon task completion. $\mathcal{NG}$ is responsible for navigating the internet to identify relevant web pages. Formally, at time step $t$, for a given website $\mathcal{W}$, $\mathcal{NG}$ samples an action $a_t \in \mathcal{A}$ from its action space (shown in Table 1), which varies depending on the information access setting.

$$a_t = \mathcal{NG}(\mathcal{W}, \mathcal{T}, \mathcal{F}, \{a_1, a_2, \ldots, a_{t-1}\})$$

If the action $a_t$ is AGGREGATE, $\mathcal{ET}$ extracts relevant information from $\mathcal{W}$ for the task $\mathcal{T}$, to provide a list of passages $\mathcal{P} = \mathcal{ET}(\mathcal{W}, \mathcal{T}, \mathcal{F})$. $\mathcal{AG}$ then evaluates the relevance of $\mathcal{P}$ according to the current information stack $\mathcal{S}$ and the task $\mathcal{T}$, updates $\mathcal{S}$. and returns natural language feedback $\mathcal{F}$, to guide $\mathcal{NG}$'s subsequent actions. Using $\mathcal{F}$, $\mathcal{AG}$ can also instruct $\mathcal{NG}$ to finish the process once sufficient information has been aggregated. Algo. 1 shows a schematic of INFOGENT's working process.

INFOGENT's modular architecture is optimized for information aggregation and enhances adaptability across diverse scenarios by dividing responsibilities among distinct components. $\mathcal{NG}$ and $\mathcal{ET}$ can utilize either language-only or multimodal models, depending on the nature of web information access discussed in §3. Given our primary focus on textual information aggregation, both access types employ the same aggregator component. Further details on $\mathcal{NG}$, $\mathcal{ET}$, and $\mathcal{AG}$ follow.

### 4.1 Navigator $\mathcal{NG}$

Recent studies (Yang et al., 2023; Wang et al., 2024) have demonstrated the capabilities of LLMs and LMMs to autonomously plan and execute sequences of thoughts and actions (Yao et al., 2023) based on a high-level directive. Building on this capability, we conceptualize the navigator as an autonomous agent tasked with exploring the web to identify relevant websites. The action space available to the navigator agent, shown in Table 1, depends on the information access setting. Specifically, under the Direct API-Driven Access setting, INFOGENT employs a tool-based LLM agent (Yang et al., 2023) as the Navigator, which leverages a search API as a tool. Conversely, in the Interactive Visual Access setting, a multimodal web navigation agent (Zheng et al., 2024b) is utilized to interact with a real-world browser and access relevant content within the webpages. The Navigator here simulates human-like browsing behavior, allowing the agent to navigate through web interfaces that may not be accessible via APIs alone.

(a) Direct API-Driven Access

| Action | Description |
|---|---|
| SEARCH (query) | Return top-5 (url, snippet) pairs |
| AGGREGATE ($\mathcal{W}$) | calls $\mathcal{ET}$ and $\mathcal{AG}$ in sequence |
| TERMINATE | Terminate navigation |

(b) Interactive Visual Access

| Action | Description |
|---|---|
| CLICK (element) | element.click() |
| SELECT (element) | element.select() |
| TYPE (element, text) | Type text in selected element |
| PRESS ENTER | Press enter |
| GO BACK | Go back to previous page |
| AGGREGATE ($\mathcal{W}$) | calls $\mathcal{ET}$ and $\mathcal{AG}$ in sequence |
| TERMINATE | Terminate navigation |

Table 1: Action space $\mathcal{A}$ of the Navigator $\mathcal{NG}$.

#### 4.1.1 Direct API-Driven Access

In this setting, web information can be accessed by querying a search API, which returns a list of relevant urls; the corresponding website content can be retrieved using automated scraping tools. In this context, $\mathcal{NG}$ is an autonomous agent (Yang

et al., 2023), based on the ReACT framework (Yao et al., 2023), which combines chain-of-thought (Wei et al., 2022) with tool calls to generate sequence of thought and actions.

The action space $\mathcal{A}$ of $\mathcal{NG}$ under this setting (shown in Table 1a), consists of two tools, namely SEARCH and AGGREGATE. Given the user task, $\mathcal{NG}$ employs SEARCH[1] with an appropriate query, resulting in a set of URLs accompanied by brief descriptive snippets. $\mathcal{NG}$ then chooses a relevant URL from this set to invoke the AGGREGATE tool, which encompasses both $\mathcal{ET}$ and $\mathcal{AG}$. $\mathcal{ET}$ first scrapes the URL and extracts relevant content $\mathcal{P}$. Next, $\mathcal{AG}$ updates $\mathcal{S}$ using $\mathcal{P}$, and returns textual feedback $\mathcal{F}$. Based on $\mathcal{F}$, $\mathcal{NG}$ adjusts its strategy accordingly: if the extracted content is affirmed as relevant and useful, it continues to explore additional websites from the initial search results; if the content is deemed irrelevant or redundant, it initiates a new search with a revised query informed by $\mathcal{AG}$'s feedback.

#### 4.1.2 Interactive Visual Access

Under this setting, information cannot be directly scraped, meaning $\mathcal{NG}$ needs to explore the web in a manner similar to human interactions with a browser. Recent work (He et al., 2024; Zheng et al., 2024a) has demonstrated promising results in leveraging powerful Large Multimodal Models (LMMs) (OpenAI, 2023) for web navigation. The navigator here is based on SeeAct (Zheng et al., 2024a), a task-completion agent, capable of finishing web tasks by planning and executing interactive actions on webpages by utilizing screenshots and candidate HTML elements. SeeAct first performs *Action Generation* to create natural language descriptions of the necessary actions to accomplish a task (e.g., "Click on search button"). Subsequently, it engages in *Action Grounding* to identify appropriate HTML elements (e.g., "[input] Departure City") and determines the corresponding operations (such as CLICK, TYPE etc.) to execute. For more details on SeeAct, please refer to Zheng et al. (2024a).

We augment SeeAct with additional capabilities required to be an effective web navigator for information aggregation. We add GO BACK and AGGREGATE actions, enabling the agent to perform backtracking and to transfer control to $\mathcal{ET}$ respectively. The full list of actions is provided in Table 1b. Further, we modify the Action Generation procedure to also condition on the textual feedback $\mathcal{F}$ from $\mathcal{AG}$. The navigation begins from the search engine home page, with $\mathcal{NG}$ leveraging the CLICK, SELECT, TYPE, and PRESS ENTER actions to get the search results and explore the web pages further. The AGGREGATE action is used to invoke $\mathcal{ET}$ and $\mathcal{AG}$ when the webpage is deemed relevant. Subsequently, based on the feedback $\mathcal{F}$, $\mathcal{NG}$ leverages the GO BACK action to retrace its steps to explore other search results, or instead perform another search using a different revised query.

[1]Our experiments use Google Search as the search API.

### 4.2 Extractor $\mathcal{ET}$

Once $\mathcal{NG}$ selects a relevant website, $\mathcal{ET}$ identifies and extracts up to $k$ relevant paragraphs for the task. Since webpages are often lengthy, using a smaller, cost-efficient model for content processing is more practical. Extraction is favored over summarization for two key reasons: smaller models tend to produce low-quality summaries due to limited capacity, and they are prone to hallucination, introducing information not present in the source. Direct extraction ensures accurate attribution and maintains reliability of the aggregated data.

In the Direct API-Driven Access setting, given a website URL, $\mathcal{ET}$ scrapes the content and feeds it into an LLM, which is prompted to identify the relevant paragraphs based on the user's task. In contrast, under the Interactive Visual Access setting, where website content cannot be directly scraped due to access restrictions, $\mathcal{ET}$ navigates the webpage by scrolling from top to bottom, capturing multiple screenshots. These screenshots are then processed by a multimodal model (OpenAI, 2023), which identifies and extracts the relevant paragraphs. This approach facilitates extraction from web interfaces that are otherwise inaccessible through traditional scraping techniques. For detailed prompts, refer to Table 9 in the Appendix.

### 4.3 Aggregator $\mathcal{AG}$

Given the content extracted by $\mathcal{ET}$, presented as a list of paragraphs $\mathcal{P}$, $\mathcal{AG}$'s task is to determine whether to incorporate any of the paragraphs into the aggregated information stack $\mathcal{S}$. For each passage $p_i$ in $\mathcal{P}$, $\mathcal{AG}$ can choose to either add $p_i$ as a new item (ADD($p_i$)), replace an existing item $s_j$ in $\mathcal{S}_t$ with $p_i$ (REPLACE($s_j$, $p_i$)) or just ignore $p_i$ if it is irrelevant or redundant. This decision-making process is achieved by prompting an LLM, with detailed prompts in Table 9 in the Appendix. Furthermore, $\mathcal{AG}$ provides textual feedback $\mathcal{F}$ to $\mathcal{NG}$ regarding what information to seek next, which guides the $\mathcal{NG}$'s subsequent actions by highlight-

ing information gaps in $\mathcal{S}$. By incorporating a feedback-driven interaction between $\mathcal{AG}$ and $\mathcal{NG}$, INFOGENT ensures the information-seeking process is adaptive to the aggregated information.

## 5 Experiments

We test INFOGENT's ability to address complex queries that require accumulating information over multiple webpages. Evaluation is based on the final answer generated by the downstream LLM, leveraging the information aggregated by INFOGENT. We consider evaluation separately for Direct API-Driven access and Interactive Visual Access.

### 5.1 Direct API-Driven Access

Here, we employ a tool-based LLM as $\mathcal{NG}$, built upon AutoGPT. To mitigate issues arising from the dynamic and potentially conflicting information on the web, we restrict our search to Wikipedia pages, following prior work Zhu et al. (2024).

#### 5.1.1 Setup

**Datasets and Metrics:** We evaluate our method on the FanOutQA (Zhu et al., 2024) and FRAMES (Krishna et al., 2024) datasets, both comprising complex queries that require accumulating information from multiple webpages. FanOutQA includes 310 multi-hop questions involving multiple entities (for e.g. *What is the population of the five smallest countries by GDP in Europe?*). FRAMES contains complex questions requiring various reasoning types: numerical (counting, comparisons, calculations), tabular (using statistics from tables or infoboxes), constraints (multiple conditions leading to a unique answer), temporal (timeline reasoning) and post-processing (specific steps after gathering all necessary facts). Excluding numerical questions–whose performance depended significantly on the final answering LLM rather than the aggregation approach–we retained 531 examples. We use the official evaluation metrics for both datasets: FanOutQA employs string accuracy and ROUGE (Chin-Yew, 2004), while FRAMES uses language model to assess whether the generated output matches the gold answer, utilizing the prompt shown in Table 7 in the Appendix.

**Baselines:** We compare INFOGENT with MindSearch (Chen et al., 2024), a multi-agent search framework involving a planner and a searcher. MindSearch models information seeking as a dynamic graph construction process via code-driven decomposition of the user query into atomic sub-questions represented as nodes. It then iteratively builds the graph for the subsequent steps, based on answers to the sub-questions. The output is then passed to a downstream LLM for answer generation, similar to INFOGENT. We also include a closed-book model as a baseline. All approaches employ GPT-4o-mini as the underlying LLM.

| Approach | All | Tabular | Temporal | Constr. | Process |
|---|---|---|---|---|---|
| Closed-Book | 23.5 | 16.4 | 19.9 | 22.7 | 11.6 |
| MindSearch | 46.3 | 41.4 | **46.6** | 47.5 | 30.0 |
| INFOGENT | **53.3** | **45.7** | 43.8 | **55.2** | **46.5** |

Table 2: Results (in %) on the Frames dataset for queries with different reasoning types under Direct API-Driven Access setting. Constr. corresponds to Constraints.

| Approach | Acc. | R-1 | R-2 | R-L |
|---|---|---|---|---|
| Closed-Book | 46.6 | 44.5 | 24.2 | 38.2 |
| MindSearch | 47.3 | 49.3 | 28.4 | 44.2 |
| INFOGENT | **51.1** | **53.3** | **33.0** | **48.5** |

Table 3: Results (in %) on the FanoutQA dev set under the Direct API-Driven Access setting.

#### 5.1.2 Results

Table 2 reports results on FRAMES across different reasoning types. Low performance of the closed-book approach highlights the complexity and recency of the questions. INFOGENT significantly outperforms MindSearch on most reasoning types; however, on temporal reasoning, MindSearch performs better, likely due to its code-driven planning in graph construction. Table 3 presents results on the FanOutQA benchmark. Both INFOGENT and MindSearch outperform the closed-book method, demonstrating the benefit of web search, with INFOGENT consistently surpassing MindSearch. The relatively high performance of the closed-book model may be due to the dataset's release date (Nov 2023) being close to the LLM's knowledge cutoff, suggesting that the LLM's parametric knowledge might already contain the required facts.

### 5.2 Interactive Visual Access

Our Navigator $\mathcal{NG}$ in this setting uses the same web browser simulation tool as in SEEACT (Zheng et al., 2024b), built on top of Playwright. The navigator initiates search from the Google homepage.

| Type | Approach | Model | Dev | Test |
|---|---|---|---|---|
| RAG | RALM-Inst | GPT-4T | 15.5 | 11.7 |
| | RALM-1S | GPT-4T | 13.6 | 10.6 |
| Web Agent | SEEACT | GPT-4T | 0.0 | 4.2 |
| | SPA | GPT-4T | 12.7 | 11.0 |
| Web Agent | INFOGENT | GPT-4o | 19.2 | **15.3** |
| | | GPT-4T | **22.0** | – |

Table 4: Accuracy (in %) on AssistantBench in Interactive Visual Access Setting. Baseline numbers are taken from Yoran et al. (2024).

### 5.2.1 Setup

**Datasets and Metrics:** We use AssistantBench (Yoran et al., 2024), a dataset for evaluating web agents on time-consuming online information-seeking tasks, such as monitoring real estate markets or locating relevant nearby businesses. It comprises 214 realistic tasks (33 dev and 181 test) that require interacting with multiple websites. To assess performance on information-dense websites (Wikipedia) under the interactive visual access setting, we use a human-curated subset of FanOutQA released by Yoran et al. (2024), containing 31 queries with updated answers where closed-book models fail. Following Yoran et al. (2024), answer accuracy is the eval metric for both datasets.

**Baselines:** Baselines are same as in in Yoran et al. (2024). RALM-Inst and RALM-1S are zero and one-shot versions of a retrieval-augmented LM that is prompted to use Google Search as a tool (Yao et al., 2023). For web-agent baselines, we consider SEEACT (Zheng et al., 2024a), designed for web task-completion. Our primary comparison is with SPA (See-*Plan*-Act) (Yoran et al., 2024), which extends SEEACT for information-seeking tasks by incorporating planning and memory modules for information transfer between steps.

### 5.2.2 Results

Table 4 presents results on AssistantBench, where INFOGENT outperforms SPA by 6.5% and 4.5% on the dev and test sets respectively, even when using the smaller GPT-4o as backbone. Due to cost considerations, we report results on dev set with GPT-4T, observing a performance gain of 9.3% over SPA. The poor performance of SEEACT confirms that task-completion web agents struggle with web information-seeking tasks. Table 5 summarizes our results on FanOutQA, where INFOGENT improves upon the SPA baseline by 19%. Since navigator is often the point of failure in web tasks, the Extractor and Aggregator in INFOGENT lower the burden on the Navigator, unlike in SPA, where a single agent handles navigation, planning and memory management. Thus, INFOGENT's modular approach, with a clear division of responsibility between the components, contributes to its superior performance.

| Type | Approach | Acc. % |
|---|---|---|
| RAG | RALM-Inst | 9.6 |
| | RALM-1S | 27.3 |
| Web Agent | SEEACT | 7.5 |
| | SPA | 30.0 |
| Web Agent | INFOGENT | **49.0** |

Table 5: Results on the human-curated subset of FanOutQA under the Interactive Visual Access Setting. All methods use GPT-4T as underlying model, and numbers for the baselines are taken from Yoran et al. (2024).

| $\mathcal{NG}$ | $\mathcal{ET}$ | $\mathcal{AG}$ | Acc. % |
|---|---|---|---|
| GPT-4o | GPT-4o | GPT-4o | 19.2 |
| GPT-4o mini | GPT-4o | GPT-4o | 0.0 (↓19.2) |
| GPT-4o | GPT-4o mini | GPT-4o | 16.5 (↓2.7) |
| GPT-4o | GPT-4o | GPT-4o mini | 17.1 (↓2.1) |

Table 6: Performance impact of using different models for $\mathcal{NG}$, $\mathcal{ET}$, and $\mathcal{AG}$ under the Interactive Visual Access setting evaluated on AssistantBench dev split.

## 5.3 Analysis

### 5.3.1 Different Models for $\mathcal{NG}$, $\mathcal{ET}$ and $\mathcal{AG}$

We conduct ablation experiments on INFOGENT under the interactive visual access setting to investigate which component, $\mathcal{NG}$, $\mathcal{ET}$ or $\mathcal{AG}$, is most dependent on the underlying model's capabilities. For this study, we evaluate the performance when using GPT-4o mini instead of GPT-4o for each component separately. Table 6 shows the results on the AssistantBench dev set. We see that the navigator is most reliant on the underlying model, with final accuracy dropping to zero when GPT-4o mini is used for $\mathcal{NG}$. In comparison, using GPT-4o mini for both $\mathcal{ET}$ and $\mathcal{AG}$ results in relatively smaller performance drops of 2.7% and 2.1% respectively.

### 5.3.2 Distribution of Actions Taken by $\mathcal{NG}$

Analyzing the action frequencies of $\mathcal{NG}$ in the Interactive Visual Setting on the AssistantBench test set, we found that 61% instances successfully terminated navigation, while the remainder resulted in timeouts/failures. The top five actions per task, with their average usage, were `CLICK` (3.40), `AGGREGATE` (3.02), `GO BACK` (2.25), `TYPE` (2.01), and `PRESS ENTER` (1.32).

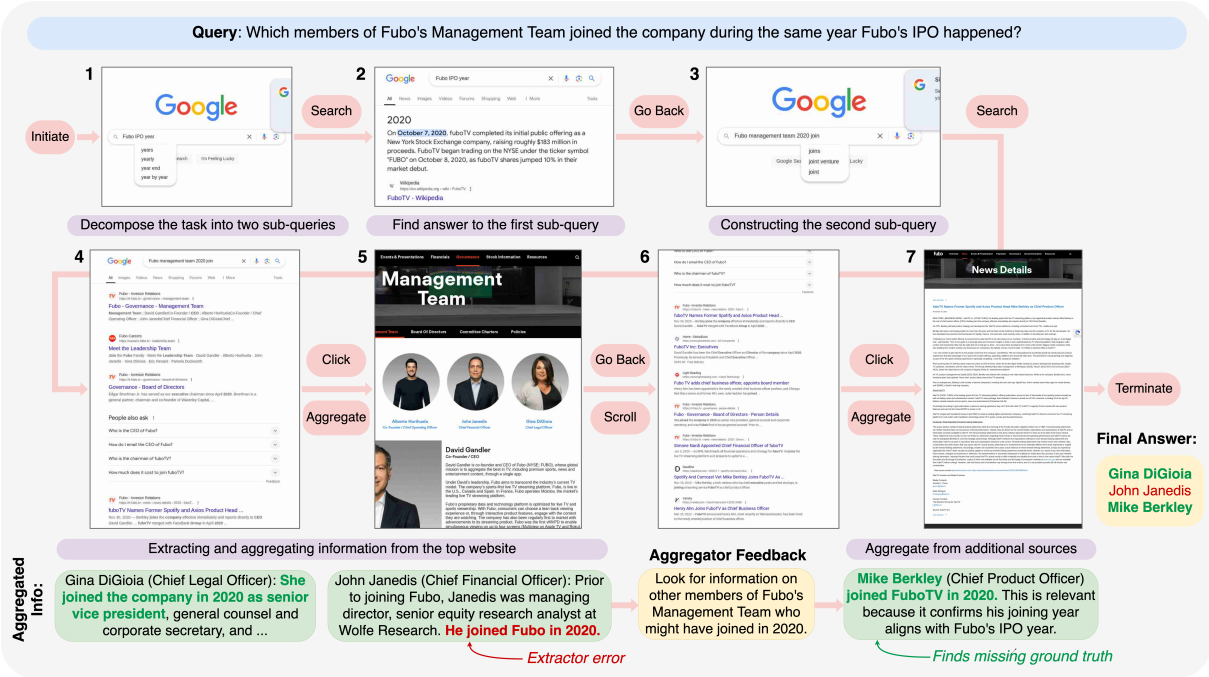


Figure 3: An illustrative example of INFOGENT in the *Interactive Visual Access* setting for a query from AssistantBench. In steps 1→4, $\mathcal{AG}$ accurately the identifies the IPO year (2020) and searches for the management team from that year. In step 5, while $\mathcal{ET}$ correctly identifies Gina DiGioia, it incorrectly extrapolates that John Janedia joined in 2020, even though his past affiliations were only mentioned up to that year. However, $\mathcal{AG}$'s feedback to "look for other members" improves the answer coverage by discovering Mike Berkley, whose name was not listed on Fubo's current web page, in an external news article (in step 7) noting his appointment as Chief Product Officer in 2020.

In the Direct API-Driven setting, SEARCH initiates a new query, and AGGREGATE involves website scraping for extraction and aggregation. For FanOutQA, SEARCH and AGGREGATE were used an average of 7.44 and 5.65 times per task, respectively; for FRAMES, these actions averaged 10.4 and 5 times per task, respectively. The higher frequency of SEARCH over AGGREGATE indicates that the navigator more often updates its search query,due to irrelevant results or because the required information is directly available in snippets, rather than extracting information.

### 5.3.3 Qualitative Analysis

Manual inspection of ten navigation traces in the Interactive Visual Access setting revealed some failure modes in the three components in INFOGENT. $\mathcal{NG}$ failed to predict correct actions in 6 out of 10 instances, exhibiting issues such as invalid assumptions during Google searches, ignoring aggregator feedback, and repeatedly triggering identical actions (see Appendices A.1 and A.2). $\mathcal{ET}$ incorrectly judges information in web page screenshots as task-relevant for 3 out of 10 examples, particularly on information-dense pages with distractions (see Figure 3 for a detailed walkthrough for one such example). $\mathcal{AG}$ often provides open-ended feedback, complicating further navigation; in 3 out of 10 cases, it gave incorrect feedback or omitted relevant information from memory. Conversely, Figure 3 illustrates how effective aggregator feedback (between steps 5 and 6) can improve answer coverage by appropriately directing the navigator.

## 6 Conclusion and Future Work

In this work, we introduce INFOGENT, a novel modular framework for web information aggregation. Through the use of separate Navigator, Extractor and Aggregator components, our approach can incorporate both tool-based LLMs and interactive web agents to handle different information access settings. Experiments demonstrate INFOGENT's superior performance over a state-of-the-art multi-agent search framework under Direct API-Driven Access and existing information-seeking web agents under Interactive Visual Access settings. Future work will incorporate evaluation on a wider variety of web information aggregation tasks. We also plan to explore measuring the diversity and coverage of aggregated information, and to assess "information sufficiency" as a criterion for terminating the information-seeking process.

## Limitations

**Navigation Challenges:** Navigation plays a pivotal role in the success of INFOGENT. As highlighted in Table 6, replacing GPT-4o with GPT-4o-mini led to a complete drop in accuracy, emphasizing the need for more effective navigation models. Existing models also struggle with diverse bottlenecks that arise during web navigation, such as solving captchas, indicating room for improvement in their robustness.

**Dependency on GPT-4:** While INFOGENT demonstrates effective collaboration between agents when leveraging high-performing models like GPT-4, the significant performance decline with GPT-4o-mini reveals an over-reliance on GPT-4's capabilities. This underscores the importance of developing open-source models capable of replicating such web navigation proficiency.

**Dataset Limitations:** Although INFOGENT operates as a fully automated framework, the process of information aggregation on the web remains inherently subjective. In our work, we had to rely on multi-hop QA datasets due to the absence of real-world datasets that capture the nuances of subjective information aggregation. Designing appropriate evaluation metrics for such tasks remains a complex challenge, warranting further exploration.

**Web's Dynamic Nature:** The constantly evolving nature of the web adds another layer of complexity to information aggregation. Time-sensitive information is prone to changes, and documents are often not updated in a timely manner. Without good SEO practices, outdated content can surface frequently. For large language models (LLMs) to aggregate reliable information, they must account for the relevance and recency of the content they encounter.

## Ethics Statement:

Automating web navigation introduces several ethical and security challenges. Agents interacting with websites may unintentionally breach terms of service or activate security measures, such as captchas, as previously mentioned. Additionally, there is a risk of accessing or utilizing sensitive or restricted information inadvertently, underscoring the need for stronger ethical guidelines and security protocols within the INFOGENT framework.

## References


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Jiangning Liu, Wenwei Zhang, Kai Chen, and Feng Zhao. 2024. Mindsearch: Mimicking human minds elicits deep ai searcher. *Preprint*, arXiv:2407.20183.

Lin Chin-Yew. 2004. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the Workshop on Text Summarization Branches Out, 2004*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

Yingqiang Ge, Yujie Ren, Wenyue Hua, Shuyuan Xu, Juntao Tan, and Yongfeng Zhang. 2023. Llm as os, agents as apps: Envisioning aios, agents and the aios-agent ecosystem. *arXiv e-prints*, pages arXiv–2312.

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

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. 2024. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*.

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

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. 2024. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. *arXiv preprint arXiv:2409.12941*.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*.

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

OpenAI. 2023. GPT-4V(ision) System Card.

Revanth Gangi Reddy, Yi R Fung, Qi Zeng, Manling Li, Ziqi Wang, Paul Sullivan, and Heng Ji. 2023. Smartbook: Ai-assisted situation report generation. *arXiv preprint arXiv:2303.14337*.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Nancy Xu, Sam Masling, Michael Du, Giovanni Campagna, Larry Heck, James Landay, and Monica Lam. 2021. Grounding open-domain instructions to automate web support tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1022–1032.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

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

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

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024b. Gpt-4v(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*.

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

Andrew Zhu, Alyssa Hwang, Liam Dugan, and Chris Callison-Burch. 2024. FanOutQA: A multi-hop, multi-document question answering benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 18–37, Bangkok, Thailand. Association for Computational Linguistics.

# A Appendix

## A.1 Navigation Failures

The Navigator is a critical component of INFOGENT. The dynamic nature of the web, especially with its constant updates and varying structures, makes this a particularly challenging task. Navigation failures manifest in multiple forms, including but not limited to pop-ups, AI-generated overviews, captchas, and other interactive elements. While these features are designed to enhance user experience, they also introduce significant barriers for a web agent attempting to navigate efficiently. These obstacles can disrupt the flow of information gathering, making it difficult to access or retrieve data accurately. Samples of web navigation failures are shown in Figure 4.

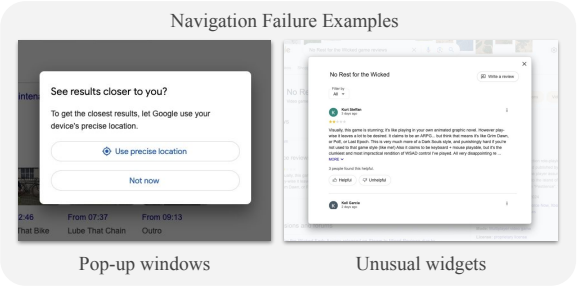


Figure 4: INFOGENT navigation error examples. The navigator falls in dead loops when encountered unusual web elements, such as pop-up windows asking for sharing locations (left) or "answer cards" occasionally appeared at the top of Google search results (right).

## A.2 Geo-Navigational Queries

We particularly observed that INFOGENT struggles with handling geo-navigational queries in AssistantBench. These queries often require precise spatial awareness and the ability to interact with dynamic map interfaces like Google Maps. For example, a query such as "Which gyms near Tompkins Square Park (within 200m) offer fitness classes before 7am?" demands not only the retrieval of location-based data but also filtering of relevant details based on distance and time constraints.

In such cases, the model must effectively parse geographic information and interact with Google Maps to identify specific venues within the given parameters. However, this task relies heavily on the Navigator to accurately traverse and manipulate the map interface, which proves to be a significant challenge for current models. Google Maps' dynamic and interactive nature makes it difficult for web agents like INFOGENT to seamlessly navigate and extract relevant data without human-like intuition. Consequently, handling geo-navigational queries requires sophisticated mechanisms for interpreting spatial data and overcoming the navigational hurdles posed by interactive web platforms. Particularly these queries cause pop-ups like the left one in Figure 4.

**Task**

I need your help in evaluating an answer provided by an LLM against a ground truth answer for a given question. Your task is to determine if the ground truth answer is present in the LLM's response. Please analyze the provided data and make a decision.

**Instructions**

1. Carefully compare the "Predicted Answer" with the "Ground Truth Answer."
2. Consider the substance of the answers - look for equivalent information or correct answers. Do not focus on exact wording unless the exact wording is crucial to the meaning.
3. Your final decision should be based on whether the meaning and the vital facts of the "Ground Truth Answer" are present in the "Predicted Answer."

**Input Data**

- Question: {question}
- Predicted Answer: {predicted}
- Ground Truth Answer: {answer}

**Output Format**

You should only respond in JSON format as described below and ensure the response can be parsed by Python json.loads.
Response Format:
{{
"Explanation": "(How you made the decision?)",
"Decision": "TRUE" or "FALSE"
}}

Table 7: Evaluation task for comparing an LLM's predicted answer with a ground truth answer.

**Navigator**

You are an assistant aiding an information aggregation process designed to gather relevant information from the web given a user task. You are provided access to a search tool that you can use to access the web. Your goal is to ensure diversity in the gathered information, so you might want to look at multiple websites in the search results.

You will work in conjunction with an aggregator assistant (which runs as part of the “extract” tool) that keeps track of information aggregated and will give feedback to you. It will also let you know how many iterations of calling “extract” are left and how many passages it has aggregated so far. You should only visit websites that you think will contain information relevant to user task. If a website does not contain any relevant information, you can skip it. DO NOT visit a website that you have already visited before.

You can leverage the web search multiple times, so that information can be aggregated information over multiple queries. You can decide to stop if aggregator assistant tells you so or if you keep running into a loop. You can simply terminate at the end with a message saying aggregation is done.

Below is the user task.
Task: {user_task}

**Extractor**

Website Data: {data}

From the above text, extract relevant information for the following task: {user_task}.

You must return the extracted information in the form of a list of paragraphs. Each paragraph should NOT be longer than 8 sentences. Only include the information that is relevant to the provided task. You can extract upto 2 paragraphs ONLY. If the text does not contain any relevant information, you can just return an empty list.

You should only respond in JSON format as described below and ensure the response can be parsed by Python json.loads.
Response Format:
{{
“paragraphs”: [list of paragraphs relevant to the task]
}}

**Aggregator**

You are an information aggregation assistant designed to aggregate information relevant to the given user task. Your goal is to ensure diversity in the gathered information while ensuring they are ALL relevant to the user task. Make sure to not gather duplicate information, i.e. do not add redundant information to what you have already aggregated. You can decide to stop aggregating when you decide you have information to address the user task. Also, you can aggregate only {num_to_aggregate} items in the list and should signal to stop when you have aggregated {num_to_aggregate} items.

From the above text, extract relevant information for the following task: {user_task}.

You will be provided with a set of passages collected from a website by a navigator assistant. You need to decide whether any of the provided information should be added to the aggregated information list. You have the option to ignore and not add any of the provided passages to the aggregated information list. Also, you should provide feedback to the navigator assistant on how to proceed next. The navigator assistant cannot see the information aggregated, so be clear and specific in your feedback. You should instruct the navigator to terminate if enough information has been aggregated. You have a maximum of {num_iterations} iterations overall, after which the information aggregated will be automatically returned.

Current Iteration Counter: {counter}
User Task: {user_task}
Information Aggregated so far: {aggregated_list}
Provided information: {provided_list}

You should only respond in JSON format as described below and ensure the response can be parsed by Python json.loads
Response Format:
{{
“thoughts”: Your step-by-step reasoning for what actions to perform based on the provided information,
“actions”: [list of actions (generated as a string) to perform. Allowed actions are: REPLACE(existing_id, provided_id) if passage existing_id in aggregated information should be replaced by passage provided_id from provided information and ADD(provided_id) if passage provided_id should be added to aggregated information],
“feedback”: Feedback to return to the navigator assistant on how to proceed next. Also, let the navigator assist know how many more iterations are left.
}}

Table 8: Input prompts for the *Navigator* (top), *Extractor* (middle), and *Aggregator* (bottom) components for the Direct API-Driven Access setting.

**Navigator**

The screenshot below shows the webpage you see. Follow the following guidance to think step by step before outlining the next action step at the current stage:

(Current Webpage Identification)
Firstly, think about what the current webpage is.

(Previous Response and Feedback Analysis)
Secondly, if provided, consider the current response generated for the task along with the feedback. If the response is insufficient, you may need to provide more details to complete the task. For instance, consider revisiting previous search results and exploring other websites to gather additional information.

(Previous Action Analysis)
Then, combined with the screenshot, analyze each step of the previous action history and their intention one by one. Pay more attention to the last step, which may be more related to what you should do next. If the last action involved a TYPE, always evaluate whether it necessitates a confirmation step.

(Screenshot Details Analysis)
Closely examine the screenshot to check the status of every part of the webpage to understand what you can operate with and what has been set or completed. Evaluate the status of every part of the webpage.

(Next Action Based on Webpage and Analysis)
Then, based on your analysis, in conjunction with human web browsing habits and the logic of web design, decide on the following action. Clearly outline which element in the webpage users will operate with as the first next target element, its detailed location, and the corresponding operation.

To be successful, it is important to follow the following rules:
1. You should only issue a valid action given the current observation.
2. If the current webpage has relevant information for the task, trigger AGGREGATE INFORMATION.
3. AGGREGATE INFORMATION is to be used when you think there is factual information that might be useful.
4. You should only issue one action at a time.
5. Press enter after typing a query if needed.
6. Prioritize visiting Wikipedia links over others.
7. Scroll is strictly not an allowed action.
8. Replan if taking the same action repeatedly.

**Extractor**

Website Data: {data}

From the above text, extract relevant information for the following task: {user_task}.

You must return the extracted information in the form of a list of paragraphs. Each paragraph should NOT be longer than 8 sentences. Only include the information that is relevant to the provided task. You can extract up to 2 paragraphs ONLY. If the text does not contain any relevant information, you can just return an empty list.

You should only respond in JSON format as described below and ensure the response can be parsed by Python json.loads.
Response Format:
{{
"paragraphs": [list of paragraphs relevant to the task]
}}

**Aggregator**

You are an information aggregation assistant designed to aggregate information relevant to the given user task. Your goal is to ensure diversity in the gathered information while ensuring they are ALL relevant to the user task. Make sure to not gather duplicate information, i.e. do not add redundant information to what you have already aggregated. You can decide to stop aggregating when you decide you have enough information to address the user task. Also, you can aggregate only {num_to_aggregate} items in the list and should signal to stop when you have aggregated {num_to_aggregate} items.

From the above text, extract relevant information for the following task: {user_task}.

You will be provided with a set of passages collected from a website by a navigator assistant. You need to decide whether any of the provided information should be added to the aggregated information list. You have the option to ignore and not add any of the provided passages to the aggregated information list. Also, you should provide feedback to the navigator assistant on how to proceed next. The navigator assistant cannot see the information aggregated, so be clear and specific in your feedback. You should instruct the navigator to terminate if enough information has been aggregated. You have a maximum of {num_iterations} iterations overall, after which the information aggregated will be automatically returned.

Current Iteration Counter: {counter}
User Task: {user_task}
Information Aggregated so far: {aggregated_list}
Provided information: {provided_list}

You should only respond in JSON format as described below and ensure the response can be parsed by Python json.loads
Response Format:
{{
"thoughts": Your step-by-step reasoning for what actions to perform based on the provided information,
"actions": [list of actions (generated as a string) to perform. Allowed actions are: REPLACE(existing_id, provided_id) if passage existing_id in aggregated information should be replaced by passage provided_id from provided information and ADD(provided_id) if passage provided_id should be added to aggregated information],
"feedback": Feedback to return to the navigator assistant on how to proceed next. Also, let the navigator assist know how many more iterations are left.
}}

Table 9: Input prompts for the *Navigator* (top), *Extractor* (middle), and *Aggregator* (bottom) components for the Interactive Visual Access setting.