WebEVA: A Web Agent with Reliable Element Selection

Anonymous ACL submission

Abstract

Existing web agents struggle with brittle planning, hallucinations caused by insufficient observations, and frequent execution failures due to unreliable element selection. We introduce WebEVA, a multimodal web agent that improves reliability across observation, decision, and execution through five key innovations: (1) generating high-level task requirements instead of low-level plans, (2) filtering elements via inner-text matching to reduce the number of candidate elements, (3) ranking icons and images via a fine-tuned ModernBERT model, (4) introducing an observation stage that analyzes the screenshot to assess task progress before deciding the next action, and (5) separating action selection (what to do next) from action parsing (how the action should be carried out), enabling clearer grounding prior to execution.

WebEVA sets a new state-of-the-art among open-source systems on the WebVoyager (He et al., 2024) and Online-Mind2Web (Xue et al., 2025) benchmarks. It achieves 82.1% in human evaluation and 90.8% in automated evaluation on WebVoyager, and 37.5% and 34.6% respectively on Online-Mind2Web.

We release WebEVA to support future research in web automation.[1]

1 Introduction

The release of ChatGPT in 2022 ignited global excitement around AI, enabling automation in tasks once deemed impossible (Xu et al., 2024). Since then, LMMs like GPT-4, Gemini, LLaVA, and Claude(OpenAI, 2024; Team, 2024; Liu et al., 2023; Anthropic, 2024) have shown impressive vision-language reasoning, achieving state-of-the-art results on various benchmarks (Saikh et al., 2022; Lu et al., 2022; Zhong et al., 2023; Yue et al., 2023; Zheng et al., 2024), fueling hopes for potential AGI (Xi et al., 2023; Eloundou et al., 2023).

This wave of progress has catalyzed a surge in real-world applications that leverage LMMs' multimodal reasoning capabilities. Among these, web agents have emerged as a particularly promising direction—AI systems that follow natural language instructions to autonomously complete tasks on real websites (Gur et al., 2023; Kagaya et al., 2024). Since the web powers many workflows (Azam et al., 2024), web agents leverage LMMs to sense webpage states (via DOM, accessibility trees, or screenshots) and act through structured actions like clicking or typing (Abuelsaad et al., 2024). This combination of constrained actions and broad domains makes the web an ideal testbed for advancing general-purpose AI agents.

A modern web agent alternates between observing and acting (Xi et al., 2023; Azam et al., 2024):

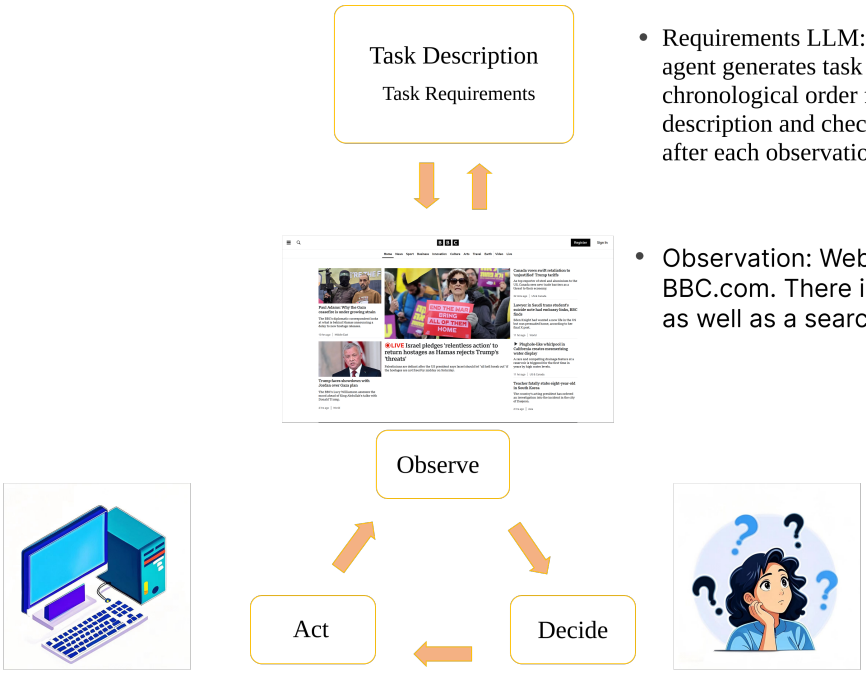
- **Observe**: The ability to accurately interpret a webpage's state, including textual content, structural layout, and interactive elements (Abuelsaad et al., 2024). At this step, the agent typically determines the next best action to achieve a given task based on the observed state, task description, and plan (if present) (Furuta et al., 2024; Minaee et al., 2024; Erdogan et al., 2025).

- **Act**: The ability to interact with webpage elements such as buttons, input fields, or links, and perform actions such as clicking, typing, or scrolling (Zheng et al., 2024).

Through continuous iteration, web agents are capable of adapting to different environments, solving complex tasks, and influencing their surroundings.

Developing web agents is challenging due to the dynamic and often cluttered nature of real-world webpages. During observation, agents may hallucinate outcomes of previous actions or produce

[1] https://anonymous.4open.science/r/WebEVA-6BC3



- Requirements LLM: The requirements agent generates task requirements in chronological order from the task description and checks for completion after each observation.
- Observation: WebEVA navigated to BBC.com. There is a news section as well as a search icon.
- Filters HTML elements to find the one with the inner text 'News' and performs a click.
- Get Action: Click on News link
- Parse Action: Type: Click
  Inner Text: News
  Icon?: False

Figure 1: WebEVA's architecture for web interaction. Given a task "Find the latest article regarding the economic implications of climate change in Europe as reported by BBC News," WebEVA navigates to BBC.com, determines the next action, identifies the relevant element, and executes it.

partially correct results—for example, selecting the cheapest visible item rather than applying the correct filters to sort for the lowest-priced option (He et al., 2024). Moreover, pages with a large amount of text and interactable elements can obscure task-relevant content and hinder accurate information retrieval (Abuelsaad et al., 2024).

When deciding the next action, such as determining which element to interact with, agents must process vast amounts of HTML content, with some webpages containing over 186,000 tokens (Zheng et al., 2024). Even with preprocessing and filtering, densely packed layouts and unfamiliar elements hinder accurate decision-making (He et al., 2024). Moreover, when following a predetermined plan, agents must frequently revise their strategies due to not knowing the layout beforehand, as well as the inherently dynamic and unpredictable nature of real-world environments (Prasad et al., 2024; Erdogan et al., 2025).

Executing actions is equally challenging. While LLMs can describe actions well, converting these descriptions into precise, executable steps remains difficult, often resulting in misidentified elements and incorrect interactions (Zheng et al., 2024; He et al., 2024).

To address these challenges, we propose WebEVA, a web agent designed to enhance the accuracy and reliability of observation, decision-making, and action execution across dynamic web environments. Figure 1 illustrates WebEVA's architecture: the agent first generates task requirements from the task description. It then enters a loop where it observes the current webpage to evaluate progress, decides on the next action using the latest observation, and executes the selected action—repeating until all requirements are satisfied.

WebEVA demonstrates state-of-the-art performance among open-source systems on both the WebVoyager (He et al., 2024) and Online-Mind2Web (Xue et al., 2025) benchmarks. On the WebVoyager dataset, WebEVA achieves 82.1% in human evaluation and 90.8% in WebVoyager's AI evaluation. On the Online-Mind2Web dataset, WebEVA achieves 37.5% in human evaluation and 34.6% in WebJudge, Online-Mind2Web's AI evaluator.

WebEVA contributes several innovations to the design of web agents:

- **Task Requirements Generation** – Instead of generating low-level, step-by-step plans, WebEVA produces a list of high-level, chronological requirements that define task completion goals.

- **Element Selection via Inner Text Matching** – For elements with visible text, WebEVA filters candidates by performing inner text matching, limiting the median number of elements to 1.

- **Icon and Image Ranking via ModernBERT** – For icons and images, WebEVA leverages a fine-tuned ModernBERT model to rank candidate elements, limiting the selection space to the top 5 most relevant elements.

- **Separation of Observation and Decision** – Previous web agents combine observation with decision, immediately deciding the next action when receiving HTML, screenshot, or both. WebEVA first analyzes the screenshot and notes the progress made toward the task requirements. This textual observation is then used as additional context for deciding the next action.

- **Separation of Get Action and Parse Action** – WebEVA separates getting the next action description (Get Action) from determining how to execute the next action (Parse Action). This separation allows the LLM to focus exclusively on determining the action details based on the screenshot and the action description during Parse Action, free from the details of earlier steps.

2 Related Works

Modern web agents employ large language models (LLMs) through one of two paradigms: *iterative execution* or *plan-and-execute* strategies (Prasad et al., 2024). Iterative agents act step by step, updating decisions after each observation. While adaptive, they struggle with long-horizon tasks due to limited compositional reasoning and planning capabilities (Dziri et al., 2023). Plan-and-execute agents generates a plan before execution, which can be efficient but risks failure if any subtask breaks down. Recent state-of-the-art open-source agents on the WebVoyager dataset favor an adaptive plan-and-execute strategy, revising plans when subtasks cannot be completed (Abuelsaad et al., 2024; Erdogan et al., 2025).

Sensing the state of a webpage is a fundamental challenge for web agents, as it determines how effectively an agent can perceive and interact with elements (Abuelsaad et al., 2024). Most existing approaches encode the Document Object Model (DOM) to represent webpage structure (Nakano et al., 2022; Lutz et al., 2024), while others use the accessibility tree for semantic information or rely on screenshots to provide visual context (He et al., 2024).

There has been previous attempts to improve web agents by focusing on processing HTML documents efficiently (Deng et al., 2023; Gur et al., 2023; Kim et al., 2023; Sridhar et al., 2023). Raw HTML documents are often massive, complex, and verbose, making them infeasible or cost-prohibitive to process directly using LMMs (Zheng et al., 2024). Recent approaches have focused on simplifying and structuring HTML to improve token efficiency while preserving key information (Nakano et al., 2022; Zhou et al., 2024; Deng et al., 2023).

A growing body of research explores visual grounding for web agents instead, where an AI model processes rendered webpages rather than relying solely on raw HTML structures (Shaw et al., 2023; Furuta et al., 2024; Hong et al., 2024). This is particularly crucial because rendered webpages are designed with user experience (UX) principles, making visual analysis more intuitive and structured than HTML parsing (He et al., 2024).

2.1 Element Grounding and Action Execution

Finding the correct element for interaction, via *element grounding*, remains a key challenge for web agents. After the agent describes the next action, such as "click on *Buy Now*", it needs to find the corresponding HTML element. Agent E extracts interactable elements from the HTML and provides them to the LMM for selection (Abuelsaad et al., 2024). MindAct and SEEACT enhance this process by using a ranking model, framing the task as a multi-choice problem based on the top-k HTML elements, while also presenting a visual screenshot of the current webpage for additional context (Zheng et al., 2024; Deng et al., 2023).

In visual-based agents, Set of Mark and Visual Prompting techniques (Kim et al., 2023; Yang et al., 2023a,b; He et al., 2024) overlay numbered bounding boxes or visual markers on interactable elements, asking the model to select from prede-

fined options, as shown in Figure 2. However, this method often struggles with *proximity errors*, where adjacent elements are misidentified, or misinterprets numerical labels, such as confusing calendar dates with numerical indicators (He et al., 2024; Zheng et al., 2024).

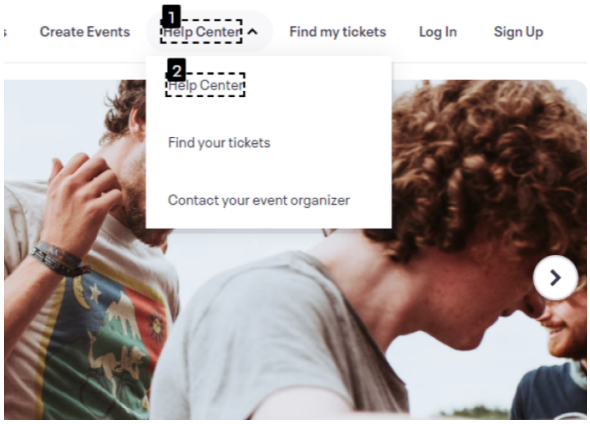


Figure 2: Example from an Online-Mind2Web task demonstrating WebEVA's use of Set of Mark prompting technique. Rather than overlaying bounding boxes on all interactable elements, WebEVA first filters elements using inner-text matching based on the action description, and then passes the screenshot to the LLM for final selection.

3 WebEVA

WebEVA begins each task by generating a list of high-level, chronological requirements that define task completion goals. Instead of low-level, specific actions, these high-level requirements offer robustness against the dynamic and unpredictable nature of real-world web interfaces, where interaction flows vary widely.

For example, given the task "Find a recipe for vegetarian lasagna with at least a four-star rating and zucchini as an ingredient," WebEVA would generate a requirement such as "Ensure the lasagna recipe has at least four stars," whereas a plan-based agent may specify "Find and click the filter icon." If no filter button exists, the plan-based agent may fail. WebEVA, by contrast, adapts by focusing on the intent rather than interface-specific steps. The system prompt used to generate task requirements is shown in Figure 5.

After generating the requirements, WebEVA follows a three-step framework: **Observe**, **Decide**, and **Act**, repeating this loop until all requirements are fulfilled.

3.1 Observe – Evaluate state

The Observe layer is responsible for analyzing the state of the task and consists of two functions:

- **Observation**: Drawing inspirations from Self-Refine (Madaan et al., 2023) and Change Observation (Abuelsaad et al., 2024), WebEVA tracks progress toward task requirements by interpreting the result of the previous action via screenshot. This enables the LMM to critique past actions, assess their contribution to the task, and summarize progress in textual form. To our knowledge, WebEVA is the first web agent to provide both the screenshot and the textual observation as joint context to the decision layer.
- **Completion**: Determines whether the task and its requirements have been completed. If so, WebEVA summarizes the findings and exits the loop.

3.2 Decide – Get and Parse Action

The Decide layer governs WebEVA's decision process and consists of two submodules:

- **Get Action (What should I do next?)**: Generates a natural language description of the next action and its rationale based on the task requirements and observations. A natural language description ensures transparency and provides clearer context for both observations and future decisions (Shinn et al., 2023; Wei et al., 2023).
- **Parse Action (How do I execute this action?)**: Maps the action description to WebEVA's action space (Click, Input, Scroll, Go Back) and dynamically refines it to match actual webpage elements. For example, if the agent plans to click "Buy now", but the webpage only displays a "Buy" button, this module adjusts the action accordingly. It determines the action type, the inner text of the element, input text, and whether the target is an icon.

3.3 Act – Execute action

The Act layer executes the parsed action on the webpage. WebEVA's action space includes:

- Clicking an element.
- Entering text into an input field.

- Scrolling.
- Going back to the previous page.

3.4 Interaction Formulation

WebEVA operates within an environment $\mathcal{E}$, leveraging a multimodal model $\mathcal{M}$, an Observation Space $\mathcal{O}$, and an Action Space $\mathcal{A}$. Each task $\mathcal{T}$ is accompanied by a set of task requirements $\mathcal{R} = \{r_1, \ldots, r_n\}$. At timestep t, the context includes the task $\mathcal{T}$, its requirements $\mathcal{R}$, and the history of previous observations and actions:

$$c_t = (\mathcal{T}, \mathcal{R}, \{(o_1, a_1), \ldots, (o_{t-1}, a_{t-1})\})$$

The agent uses this context to generate an action a_t:

$$a_t = \mathcal{M}(c_t),$$

which is executed in $\mathcal{E}$, producing a new observation o_{t+1} consisting of both a screenshot and text observation:

$$o_{t+1} = \mathcal{E}(o_t, a_t).$$

Actions $a \in \mathcal{A}$ are defined as (e, op, v), where e is the target element, op is the operation (e.g., Click, Input), and v is an optional parameter (e.g., text for input). This iterative process continues until the task is completed or a termination condition is reached.

3.5 Element Finding

The action step remains a major challenge for web agents. Even when the LMM predicts the correct action, agents frequently fail to locate the corresponding element on the page. As SEEACT notes, it is "challenging to convert the action description into an executable action" (Zheng et al., 2024). Similarly, Wilbur finds that "in many cases, even if the agent predicts the correct action, it is unable to perform it on the page" (Lutz et al., 2024). The difficulty increases on complex websites, with WebVoyager highlighting that "websites with more interactable elements are more challenging for agents" (He et al., 2024). Additionally, Agent E reports common "navigation issues" and the "inability to operate a button" (Abuelsaad et al., 2024).

In our Online-Mind2Web dataset run, webpages averaged 268.4 interactable elements. For a task like "Find a chocolate cupcakes recipe with over 1,000 ratings", even if the LMM predicts the correct action—"Click the 'Chocolate Cupcakes' link with 1,565 ratings"—selecting the right element among hundreds remain a major challenge and source of failure.

3.5.1 Filtering by Predicted Inner Text

To address the challenge of element finding, we first extract the predicted inner text from the LMM generated action description. In the example above, the predicted inner text would be "Chocolate Cupcakes". We then filter interactable elements by selecting those whose inner text contains the predicted phrase and that are currently visible on the screen.

This filtering significantly reduces the search space. In our Online-Mind2Web dataset run, exactly one matching element remained in 73.66% of cases, with an average of 1.62 remaining elements overall.

If multiple elements remain, we pass distilled HTML representations (tag name, attributes, and inner text) along with the current screenshot overlaid with numbered bounding boxes (Set of Mark prompting) to the LLM, which selects the target element by choosing among the numbered options. We illustrate our use of Set of Mark in Figure 2.

3.5.2 Elements Without Inner Text (Icon and Image Selection)

For parsed actions that lack inner text—such as clicking on icon and images—we first use a fine-tuned ModernBERT model to narrow down the candidates to the top $k = 5$ elements. We then send distilled HTML representations (tag name and attributes), along with a screenshot overlaid with numbered bounding boxes using Set of Mark prompting, to the LLM for final selection. Details of the model are provided in Appendix A.

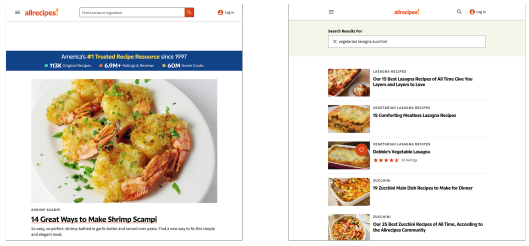
4 Evaluation

WebEVA is built on top of Playwright, an open-source automation framework designed for web scraping and browser interaction (Microsoft, 2024). We first evaluated WebEVA using the WebVoyager dataset, which includes 643 tasks spread across 15 live websites, each containing 40–46 tasks (He et al., 2024). Previous datasets often relied on simplified simulators or synthetic environments (Putta et al., 2024; Lutz et al., 2024; Zhou et al., 2024), or restricted exploration to fixed action sequences (Deng et al., 2023), which fail to capture the dy-

namic and full representation of real-world websites (Xue et al., 2025).

While WebVoyager is currently the most widely used dataset, it suffers from several limitations: limited coverage of websites and tasks, vague instructions that can lead to multiple interpretations of the correct answer, and outdated content due to changing page layouts (Xue et al., 2025).

For better coverage, we additionally evaluated WebEVA on the Online-Mind2Web dataset, the latest version of Mind2Web. Online-Mind2Web offers 300 diverse tasks spanning 136 websites and includes a built-in automatic evaluation system, WebJudge, which leverages LLM-as-a-judge to provide scalable, unbiased assessments of task success (Xue et al., 2025).



1. Search vegetarian lasagna zucchini
2. Click Debbie's Vegetable Lasagna
3. Scroll down to see ingredients
4. Verify Result. Return Answer

Figure 3: A WebVoyager task for allrecipes.com: "Find a recipe for vegetarian lasagna with at least a four-star rating and zucchini as an ingredient." We demonstrate how our web agent completes this task. For an example from Online-Mind2Web, see Appendix C.

We ran our agent at a resolution of 900 × 1600 for the WebVoyager dataset and 1280 × 1440 for the Online-Mind2Web dataset, with a 20-action limit per task. Due to time and budget constraints, our evaluation was conducted exclusively using GPT-4.o (for WebVoyager) and GPT-4.1 (for Online-Mind2Web). While alternative LMMs such as Claude and Gemini may offer competitive results, our focus remains on validating WebEVA's architecture and showcasing our methodology, rather than performing an extensive LMM benchmark study. Additional differences between the WebVoyager and Online-Mind2Web runs are listed in Appendix B.

Retries were not allowed, as simple redo mechanisms often improve success rates (Kapoor et al., 2024; Abuelsaad et al., 2024). The total development and execution cost was approximately $680, with each task typically requiring 3 to 12 minutes to complete, depending on the number of actions taken.

On the WebVoyager dataset, we made necessary modifications to ensure task feasibility. Many tasks originally specified past dates (e.g., "Search a hotel with free WiFi and air conditioning in Bali from Jan 1 to Jan 4, 2024"). To make the task feasible, we updated past dates to reasonable future equivalents. Additionally, a subset of tasks became impossible due to changes in website layout or content. After filtering these tasks, 588 valid tasks remained.

For the Online-Mind2Web dataset, we excluded 17 tasks in which our Amazon EC2 instance was either unable to access the site or became stuck on CAPTCHA challenges. Although the dataset is designed to avoid sites protected by CAPTCHA (Xue et al., 2025), some websites employ additional safeguards against automation tools like Playwright or restrict access when they detect cloud-based infrastructure.

4.1 Results

Five student annotators assessed the text observations and screenshot trajectories to determine whether a task was successfully completed. In cases of uncertainty, a secondary annotator reviewed the task to make the final judgment.

WebEVA establishes new benchmarks on both the WebVoyager and Online-Mind2Web datasets, surpassing previous open-source agents in human and LLM-based automatic evaluations. On the WebVoyager dataset, WebEVA achieves 82.1% success in human evaluation (Table 1) and 90.8% using WebVoyager's built-in AI evaluator. On the Online-Mind2Web dataset, it attains 37.5% in human evaluation and 34.6% with WebJudge, the dataset's autonomous evaluation framework (Table 2).

4.2 Failure Modes

Despite WebEVA achieving strong performance on the WebVoyager dataset, its performance on the Online-Mind2Web dataset was noticeably lower. This gap can be attributed to the Online-Mind2Web dataset's broader diversity of tasks and longer interaction trajectories. Tasks in this dataset are seperated by difficulty based on trajectory length: tasks with $N_{\text{step}} \leq 5$ are labeled as easy, $6 \leq N_{\text{step}} \leq 10$

Agent	Allrecipes	Amazon	Apple	Arxiv	BBC	Booking	Cambridge	Coursera
He et al., 2024 (text)	57.8	43.1	36.4	50.4	45.2	2.3	66.7	24.6
He et al., 2024 (multi)	51.1	52.9	62.8	52.0	60.3	32.6	71.3	57.9
Lutz et al., 2024	60.0	43.9	60.5	51.2	81.0	38.6	**86.0**	51.1
Abuelsaad et al., 2024	71.1	70.7	74.4	62.8	73.8	27.3	81.4	85.7
WebEVA	**93.0**	**83.3**	**81.3**	**69.2**	**92.3**	**62.5**	**86.0**	**90.2**

Agent	ESPN	Flights	Github	Google	Huggingface	Maps	Wolfram	Overall
He et al., 2024 (text)	28.6	7.1	63.4	75.2	31.0	62.6	60.2	44.3
He et al., 2024 (multi)	47.0	51.6	59.3	77.5	55.8	64.3	60.9	57.1
Lutz et al., 2024	59.1	0.0	22.0	67.4	53.5	39.0	65.2	52.6
Abuelsaad et al., 2024	77.3	35.7	82.9	**90.7**	**81.0**	**87.8**	**95.7**	73.1
WebEVA	**85.3**	**57.1**	**89.7**	82.9	75.0	85.4	**95.7**	**82.1**

Table 1: WebEVA outperforms prior baselines on 12 out of 15 websites in the WebVoyager dataset, based on human evaluation.

Agent	Human Eval. (%)	Human Eval. Date	Auto Eval. (%)	Auto Eval. Date
Operator (OpenAI)	61.3	2025-03-22	58.3	2025-05-11
Computer Use (Anthropic)	56.3	2025-04-20	47.3	2025-05-11
SeeAct (Zheng et al., 2024)	30.7	2025-03-22	30.0	2025-05-11
Browser Use	30.0	2025-03-22	26.0	2025-05-11
Agent-E (Abuelsaad et al., 2024)	28.0	2025-03-22	27.0	2025-05-11
WebEVA	37.5	2025-05-14	34.7	2025-05-14

Table 2: Human and WebJudge evaluation results on the Online-Mind2Web dataset. WebEVA achieves the highest performance among open-source agents, though it lags behind proprietary agents.

Failure Type	Cases
Navigation Stuck	59
Interaction Errors	17
Hallucination	22
Action Not Available	38
Lack of Constraint Enforcement	15
Partial Observation Error	26

Table 3: The table presents six types of errors and their corresponding number of occurrences.

as medium, and $N_{\text{step}} \geq 11$ as hard (Xue et al., 2025). In the manual evaluation, WebEVA completed 46 out of 78 easy tasks, 50 out of 135 medium tasks, and only 10 out of 70 hard tasks, highlighting the model's reduced effectiveness as task complexity increases.

To better understand the challenges faced, we categorize the failure cases into six main types, as shown in Table 3. Although each failure is labeled under a single category, many involve overlapping issues that span multiple types rather than fitting neatly into a single classification.

Navigation Stuck The agent occasionally gets trapped in a loop, repeatedly attempting ineffective actions without making progress (He et al., 2024).

Interaction Errors Interaction error is when the agent is unable to convert the action description into an executable action.

- The agent fails to select the correct element from the list of HTML elements (He et al., 2024).
- The correct HTML element is not included in the list of possible selections.

Hallucination The agent generates incorrect information that appears plausible (He et al., 2024). This error falls into two subcategories:

Action Hallucination: The agent assumes an action has been successfully executed based on the action description rather than the screenshot evidence.

Visual Hallucination: The agent assumes an action is available when it does not exist.

Action Not Available Certain tasks involve elements or interactions that WebEVA cannot handle effectively, leading to failure (Lutz et al., 2024). Common cases include:

- Failure to interact with new tabs.
- Failure to interact with modals (pop-ups).

Lack of Constraint Enforcement Despite WebEVA's use of task requirements, the agent sometimes fail to enforce task conditions and provides an answer that only partially meets the requirements (Abuelsaad et al., 2024)

Example: For the task *"Find baby shoes priced under $20 with a 5-star rating."* The agent knowingly returned baby shoes with a 4-star rating.

Partial Observation Error The agent believes it has fulfilled the task correctly but has only considered a subset of the relevant data.

Example: Given the task *"Find the cheapest used 8-cylinder bmw made between 2005-2015"* The agent returns the cheapest bmw from the first page but forgets to sort by lowest price first.

4.3 Discussion

Successes The number of hallucinations was minimal, with only 22 instances recorded. This low rate can be attributed to WebEVA's use of both textual and visual observations, which enables a more thorough understanding of the environment. By incorporating multimodal input, WebEVA reduces the likelihood of hallucination.

Interaction errors were rare, with only 17 instances observed. This reflects the strength of WebEVA's element selection mechanism. The use of inner text matching for filtering, fine-tuned ModernBERT model for ranking, and set-of-mark prompting for selection proved effective in helping to identify the correct target elements and translating action descriptions into precise executions.

Areas for Improvement WebEVA's limitations reveal several areas for future development. Hallucination and partial observation errors often stem from the agent's inability to fully perceive the current state of the environment. To address this, we can leverage the correspondence between changes in HTML and rendered visual elements (Zheng et al., 2024) to more effectively track state changes and reduce misinterpretations by the LMM.

Interaction errors mostly stemmed from challenges in visual grounding. The LMM occasionally misidentified images containing text as elements with inner text, rather than recognizing them as icons or images. A possible solution is to first check whether an element with the predicted inner text exists; if not, we can estimate the image's position and attempt interaction through hover and click.

Additionally, some interactable elements lack inner text, while adjacent non-interactable elements—such as labels or spans—contain the descriptive text. In these cases, the LMM may incorrectly associate the label's text as the interactable element's inner text and fail to match it directly. A potential solution is to group nearby non-interactable and interactable elements using DOM parent-sibling relationships. This would enable inner text matching to operate on the label while ensuring actions are performed on the corresponding interactable element.

Another key limitation is the lack of constraint enforcement. In some cases, WebEVA knowingly returns an incomplete answer, even though it has generated a list of task requirements. One potential improvement is to implement a checklist that tracks which requirements have been completed and which remain, forbidding completion until all requirements are met.

Finally, navigation failures remain a significant bottleneck, accounting for 59 errors. These failures are primarily due to context window limitations and the agent's difficulty in reasoning over long action-observation sequences (Hosseini et al., 2024). To alleviate this, a possible direction is to reduce contextual burden by summarizing past attempts in a structured format (Lutz et al., 2024).

5 Conclusion

In this work, we introduced **WebEVA**, a multimodal web agent that advances autonomous web interaction. Our strong performance on the WebVoyager and Online-Mind2Web datasets among open-source models is driven by several key innovations: generating task requirements, filtering elements based on inner text, ranking icon and image elements using a fine-tuned ModernBERT, clearly separating observation from decision, and decoupling action selection from action parsing. These contributions address key limitations of prior systems and provide a blueprint for building more reliable and capable web agents.

Future work can further improve performance by enhancing context retention for better task continuity, improving observation accuracy through HTML change tracking, and monitoring task requirements to ensure task completion.

6 Limitations

WebEVA has the following limitations:

Limited Action Space WebEVA operates within a constrained action space consisting of Click, Input, Scroll, and Go Back. It lacks support for more complex interactions such as: double-clicking, drag-and-drop, or cursor movement.

Human-Agent Interaction In real-world applications, users may not always have a well-defined objective when assigning tasks to an AI agent. WebEVA currently operates in a fully autonomous manner, with no mechanism for interacting with a human user during its execution. This lack of human-agent communication limits its ability to ask for clarification or request additional input when needed.

References

Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan, Aditya Vempaty, and Ravi Kokku. 2024. Agent-e: From autonomous web navigation to foundational design principles in agentic systems. *Preprint*, arXiv:2407.13032.

Anthropic. 2024. Claude 3 family: New advancements in ai. Accessed: 2025-02-10.

Ruhana Azam, Tamer Abuelsaad, Aditya Vempaty, and Ashish Jagmohan. 2024. Multimodal auto validation for self-refinement in web agents. *Preprint*, arXiv:2410.00689.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Preprint*, arXiv:2306.06070.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality. *Preprint*, arXiv:2305.18654.

Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. 2023. Gpts are gpts: An early look at the labor market impact potential of large language models. *Preprint*, arXiv:2303.10130.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. *Preprint*, arXiv:2503.09572.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. 2024. Multimodal web navigation with instruction-finetuned foundation models. *Preprint*, arXiv:2305.11854.

Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. 2023. Understanding html with large language models. *Preprint*, arXiv:2210.03945.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, Bangkok, Thailand. Association for Computational Linguistics.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

Peyman Hosseini, Ignacio Castro, Iacopo Ghinassi, and Matthew Purver. 2024. Efficient solutions for an intriguing failure of llms: Long context window does not mean llms can analyze long sequences flawlessly. *Preprint*, arXiv:2408.01866.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. 2024. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *Preprint*, arXiv:2402.03610.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. 2024. Ai agents that matter. *Preprint*, arXiv:2407.01502.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *Preprint*, arXiv:2303.17491.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *Preprint*, arXiv:2304.08485.

P. Lu, S. Mishra, T. Xia, L. Qiu, K.-W. Chang, S.-C. Zhu, O. Tafjord, P. Clark, and A. Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. *ArXiv*, abs/2209.09513.

Michael Lutz, Arth Bohra, Manvel Saroyan, Artem Harutyunyan, and Giovanni Campagna. 2024. Wilbur: Adaptive in-context learning for robust and accurate web agents. *Preprint*, arXiv:2404.05902.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Microsoft. 2024. Playwright: Fast and reliable end-to-end testing for modern web apps. Accessed: 2025-02-15.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *Preprint*, arXiv:2402.06196.

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

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. Adapt: As-needed decomposition and planning with language models. *Preprint*, arXiv:2311.05772.

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

T. Saikh, T. Ghosal, A. Mittal, A. Ekbal, and P. Bhattacharyya. 2022. Scienceqa: A novel resource for question answering on scholarly articles. *International Journal on Digital Libraries*, 23:289–301.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. 2023. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. *Preprint*, arXiv:2306.00245.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Abishek Sridhar, Robert Lo, Frank F. Xu, Hao Zhu, and Shuyan Zhou. 2023. Hierarchical prompting assists large language model on web navigation. *Preprint*, arXiv:2305.14257.

Gemini Team. 2024. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey. *Preprint*, arXiv:2309.07864.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. 2024. Theagentcompany: Benchmarking llm agents on consequential real world tasks. *Preprint*, arXiv:2412.14161.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. An illusion of progress? assessing the current state of web agents. *Preprint*, arXiv:2504.01382.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023a. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *Preprint*, arXiv:2310.11441.

Lingfeng Yang, Yueze Wang, Xiang Li, Xinlong Wang, and Jian Yang. 2023b. Fine-grained visual prompting. *Preprint*, arXiv:2306.04356.

X. Yue, Y. Ni, K. Zhang, T. Zheng, R. Liu, G. Zhang, S. Stevens, D. Jiang, W. Ren, Y. Sun, C. Wei, B. Yu, R. Yuan, R. Sun, M. Yin, B. Zheng, Z. Yang, Y. Liu, W. Huang, H. Sun, Y. Su, and W. Chen. 2023. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. *ArXiv*, abs/2311.16502.

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

W. Zhong, R. Cui, Y. Guo, Y. Liang, S. Lu, Y. Wang, A. S. S. Saied, W. Chen, and N. Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *ArXiv*, abs/2304.06364.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. *Preprint*, arXiv:2307.13854.

A Model Implementation Details

We fine-tuned `ModernBERT-base`[2], a Transformer model with 22 layers and approximately 149 million parameters, for a binary classification task that determines whether a given HTML element matches a natural language description of its function.

[2] https://huggingface.co/answerdotai/ModernBERT-base

Dataset. The dataset was constructed by collecting interactable HTML elements with no inner text from the top 100 most accessible websites, as identified by ChatGPT. For each element, a description of its purpose was generated using ChatGPT and validated by human annotators. Positive examples were formed by pairing elements with their correct descriptions; negative examples were generated by pairing the same element with mismatched descriptions sampled from other elements on the same page.

An example positive pair would be the HTML element "<button role="button" aria-label="Open menu"></button>" and the corresponding description: "Click on the menu icon (three horizontal lines) to open navigation options."

To guide negative sampling, we analyzed the WebVoyager dataset and found that the median number of visible interactable elements without inner text per page was 7 (standard deviation: 1.5). We constructed negative examples using up to two standard deviations from this baseline. The final dataset consists of 1,762 training examples and 440 test examples.

Training. Training was performed for 5 epochs using the AdamW optimizer with a learning rate of 2×10^{-5} and a batch size of 8. The entire training process completed in under 10 minutes on a free T4 GPU using Google Colab. The training script and dataset is available on our GitHub repository.

Results. The model achieved an element selection accuracy of 98.18% on the test set. When ranking candidates (ranging from 4 to 10 elements) by predicted probability and returning the top five, Recall@5 reached 100%. The results are averaged over three runs.

B Differences between WebVoyager and Online-Mind2Web Runs

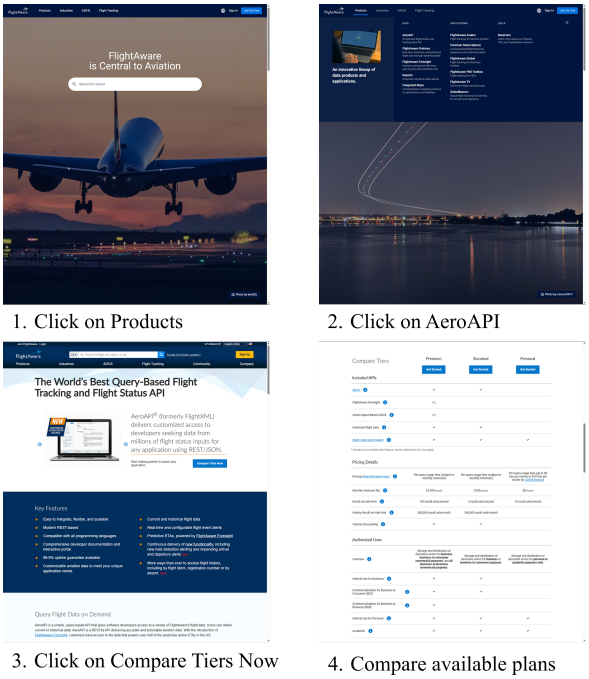
In addition to being conducted at different screen resolutions, the WebVoyager run included two additional functionalities that were later removed to reduce agent shortcuts, which may be considered a source of unfairness. First, the agent was originally capable of modifying URL parameters to achieve task goals more directly. Second, the agent previously closed popups and modals automatically, which was also disabled to prevent bypassing interactive steps.

WebVoyager also used a more lenient element-matching strategy. Specifically, it applied *inclusive inner text matching*, where an element was considered a match if either the element's inner text included the predicted inner text or vice versa. For the Online-Mind2Web run, we adopted a stricter criterion: a match only occurred when the element's inner text included the predicted inner text.

The WebVoyager experiments were conducted using a combination of Astrill VPN (with servers in Taiwan and Korea) and an Amazon EC2 instance located in California. The Online-Mind2Web experiments were conducted entirely on an EC2 instance, also located in California.

C Online-Mind2Web Task Example

Figure 4 shows an example task from the Online-Mind2Web dataset: "Compare available plans for the AeroAPI on FlightAware." This task is classified as *easy* and requires only four steps.



1. Click on Products
2. Click on AeroAPI
3. Click on Compare Tiers Now
4. Compare available plans

Figure 4: A sample Online-Mind2Web task trajectory.

D Artifacts

We utilized the following artifacts during development and evaluation (Table 4).

E Prompts

Instruction: You are given a user task involving web browsing. Your job is to extract a list of high-level, generalized conditions that must be satisfied in order for the task to be considered complete.

Guidelines:

- Do not assume any specific website structure.
- Focus on the semantic intent, not interface-specific steps.
- Return the conditions in chronological order, based on what must be verified or fulfilled.
- Express each condition as a clear, goal-oriented check (e.g., "A product costs less than $50", "Posts are sorted by date").

Input (Example 1):
Find me a hotel for May 4th to 7th, under $200 per night, with Wi-Fi and breakfast.
Website: `https://www.expedia.com/`
Output:

1. The hotel is available from May 4th to May 7th.
2. The nightly rate is under $200.
3. The hotel includes Wi-Fi.
4. The hotel includes breakfast.

Figure 5: System prompt for generating task requirements from a natural language instruction.

Artifact	**License**
WebEVA (Our Agent)	MIT
WebVoyager Dataset	Apache 2.0
Online-Mind2Web Dataset	CC-BY 4.0
Playwright	Apache 2.0
GPT-4	Proprietary
Doubao (ByteDance)	Proprietary
ModernBERT (Fine-tuned)	Apache 2.0

Table 4: Summary of artifacts used in this work.

Instruction: Your role is to analyze the user's current action to generate a new observation. The new observation should be short.

Input Details:

- Current task
- Conditions to be met (for task completion)
- Previous observations
- Current action
- Current screenshot

Your Role:

- Use the screenshot to provide analysis of the current action and its outcome.
- If the answer is located or partly located in the screenshot, include it in the observation.

Example:
Task: Find a chicken curry recipe on allrecipes.com
Conditions: The recipe should include chicken and curry in the title or ingredients.
Previous Observations:

- Opened allrecipes.com
- Searched for "chicken curry" in the search bar

Current Action: Clicked on a recipe card with the title "Indian Chicken Curry (Murgh Kari)"
Screenshot shows: A recipe page with the title "Indian Chicken Curry (Murgh Kari)", 4.5 stars, over 2000 reviews, and an ingredient list including "chicken breasts" and "curry powder".
Observation: The user navigated to a relevant chicken curry recipe that includes both "chicken" and "curry" in the title and ingredients. This recipe appears suitable for task completion.

Figure 6: System prompt for generating an observation based on the action and screenshot.

Instruction: You are a Robot tasked with browsing the web to complete a specific task.

Input Details:

- Current task
- Conditions to be met (for task completion)
- Previous actions
- Previous observations of the results of those actions
- Current screenshot

Your Role: Determine the **single optimal next action** to progress toward completing the task.

Response Requirements:

- `user_action_and_explanation`: A single string combining:
  1. The action to be taken.
  2. A specific explanation of why the action is optimal.
  3. Details about the exact element the action is performed on, including its **inner text**, **placeholder**, or **icon/image description**.

Guidelines:

- **Action Selection**:
  - Use inputs to perform a search and state the text to input.
  - The search query must contain only the essential keywords and omit all descriptive details, constraints, or additional criteria.
  - For text areas or text inputs, write `type` or `search`; you do not need to click on them.
  - For navigation or interaction elements like buttons and links, use `click`.
  - Use `scroll` if the needed information is likely not visible.
  - Use `go back` if returning to a previous page is necessary.
- **Action Constraints**:
  - Select only **one action** per observation.
  - Base the action solely on the task, the most recent observation, and the current screenshot.
  - **Do not** sign up or log in.
  - **Do not** combine actions or suggest sequences of actions.

Figure 7: System prompt for generating the next action.

Instruction: You are a Robot tasked with browsing the web. You are given:

- The *task*
- The *current action* to take
- A *webpage screenshot*

Based on the provided information, your job is to determine how to complete the current action.

Actions: Choose one of the following actions and provide the required fields:
`"action"`: One of: `input`, `click`, `scroll_up`, `scroll_down`, `go_back`
`"inner_text"`: The innerText or placeholder of the element for text or click actions. Make sure it is visible in the screenshot.
`"is_clickable_without_visible_text"`: `true/false`. Identifies if it's an icon/image type. Set to true for icons, X buttons, image buttons, and icon buttons.
`"input_value"`: The exact text to input into the field for input actions, using essential keywords only. Leave blank for other actions.

Guidelines:

- **Input Actions**:
  - Provide the text to input in `input_value`.
  - Use `inner_text` to specify the visible text or placeholder.
- **Click Actions**:
  - Use `click` only for clickable elements.
  - If the element lacks visible text (e.g., icons), set `is_clickable_without_visible_text` to `true`.
  - Do not click on input fields—use the `input` action instead.
- **Scroll Actions:**
  - `scroll_up` and `scroll_down` do not require `inner_text` or `input_value`.
- **Go Back**:
  - The `go_back` action does not require any additional fields.

Ensure the selected action aligns with the task and makes meaningful progress toward completion.

Figure 8: System prompt for determining how to execute a given action on the webpage.

Instruction: You are a web agent tasked with selecting the best element for the current action.
If none of the elements on the page are suitable, respond with `0`.

Inputs:

1. **Task**: The overall user goal.
2. **Action**: The specific user intent for this step.
3. **Elements**: A JSON array of elements on the page, each with properties and attributes.
4. **Screenshot**: The webpage image, with bounding boxes and index numbers in the top-left corner of each element.

Output Format: `"index": number`

Figure 9: System prompt for selecting an element using set of mark and element properties.