Cybernaut: Towards Reliable Web Automation

Ankur Tomar
axtomar@amazon.com
Applied AI, Amazon.com
Bellevue, Washington, USA

Hengyue Liang
hengyue@amazon.com
Applied AI, Amazon.com
Bellevue, Washington, USA

Indranil Bhattacharya
bindrani@amazon.com
Applied AI, Amazon.com
Bellevue, Washington, USA

Natalia Larios
natalild@amazon.com
Applied AI, Amazon.com
Bellevue, Washington, USA

Francesco Carbone
carbonef@amazon.lu
Applied AI, Amazon.com
Bellevue, Washington, USA

Abstract

The emergence of AI-driven web automation through Large Language Models (LLMs) offers unprecedented opportunities for optimizing digital workflows. However, deploying such systems within industry's real-world environments presents four core challenges: (1) ensuring consistent execution, (2) accurately identifying critical HTML elements, (3) meeting human-like accuracy in order to automate operations at scale and (4) the lack of comprehensive benchmarking data on internal web applications. Existing solutions are primarily tailored for well-designed, consumer-facing websites (e.g., Amazon.com, Apple.com) and fall short in addressing the complexity of poorly-designed internal web interfaces. To address these limitations, we present *Cybernaut*, a novel framework to ensure high execution consistency in web automation agents designed for robust enterprise use. Our contributions are threefold: (1) a Standard Operating Procedure (SOP) generator that converts user demonstrations into reliable automation instructions for linear browsing tasks, (2) a high-precision HTML DOM element recognition system tailored for the challenge of complex web interfaces, and (3) a quantitative metric to assess execution consistency. The empirical evaluation on our internal benchmark demonstrates that using our framework enables a 23.2% improvement (from 72% to 88.68%) in task execution success rate over the baseline [13]. Cybernaut identifies consistent execution patterns with 84.7% accuracy, enabling reliable confidence assessment and adaptive guidance during task execution in real-world systems. These results highlight Cybernaut's effectiveness in enterprise-scale web automation and lay a foundation for future advancements in web automation.

CCS Concepts

• **Applied computing** → *Business process management*; • **General and reference** → *Evaluation*; • **Computing methodologies** → **Planning under uncertainty**.

Keywords

AI Agents, Web-navigation, Consistency, Operations, Autonomous web-browsing

ACM Reference Format:
Ankur Tomar, Hengyue Liang, Indranil Bhattacharya, Natalia Larios, and Francesco Carbone. 2018. Cybernaut: Towards Reliable Web Automation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (KDD '25).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Recent advances in Large Language Models (LLMs) have provided them with remarkable reasoning capabilities that enable the development of autonomous agents. Autonomous agents can automate and optimize Knowledge Operations, which are repetitive and well defined tasks involving the creation, manipulation, or classification of knowledge. A common step in Knowledge Operation tasks consists of interacting with a web application to retrieve, submit and manipulate data. Instruction-based web browsing tools enable automation for these types of tasks. They represents a significant frontier in this domain [14], where LLMs are equipped with browsing tools (e.g., navigation, clicking, scrolling, form filling, etc.) to interpret website structures and execute required actions. While several solutions have emerged, including Operator [15], Computer use [2] and Browser-Use [13], their evaluation methodologies often rely on selecting optimal performances from multiple attempts, highlighting a critical limitation in consistency and reliability. Moreover, public benchmarks like WebVoyager [6] may not effectively represent scenarios where agent interacts with a limited set of websites but demand high consistency and quality. This disparity between benchmark performance and practical requirements underscores the need for more focused evaluation methodologies.

Enterprise web automation faces several challenges particular to real-world scenarios, especially when performing repetitive tasks with dynamic parameters — for example, retrieving hazardous material ratings for different ASINs. Existing solutions often depend on brittle, hard-coded, element-based approaches that are highly sensitive to UI changes. Also, accurately detecting *interactable* elements on web pages remains a significant challenge. While tools like *Selenium* and *Playwright* offer mechanisms to generate summarized snapshots of HTML elements, the heterogeneity of web interfaces often leads to detection failures, resulting in incomplete action spaces and reduced task accuracy. Furthermore, existing browsing agents rely heavily on detailed task descriptions from

KDD '25, Toronto, ON

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

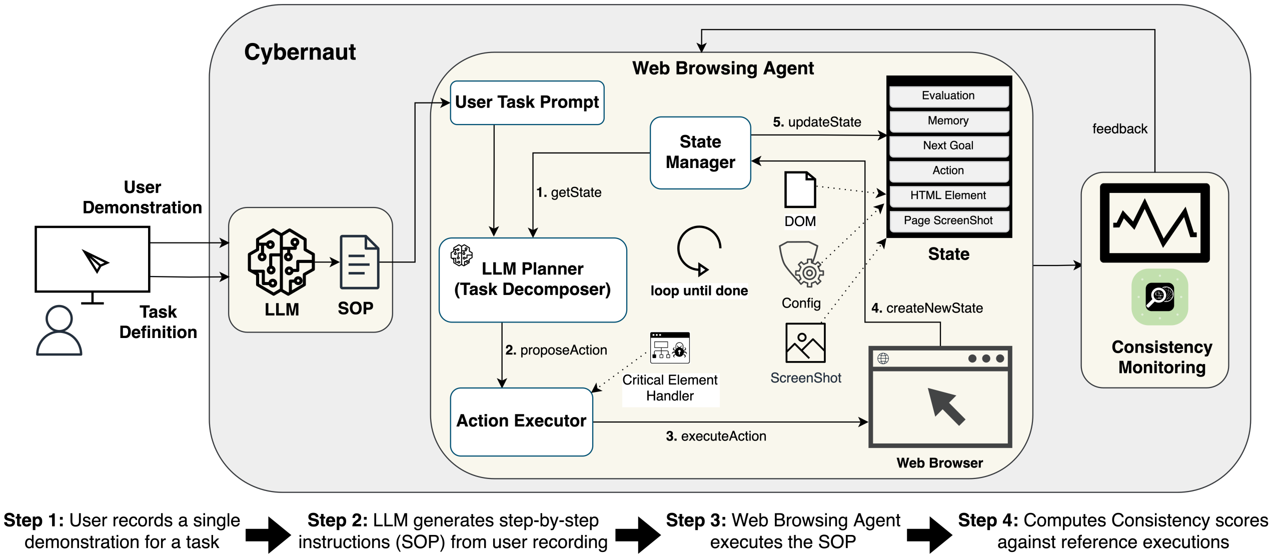

Figure 1: Architectural overview of Cybernaut's workflow pipeline, illustrating the transformation of user demonstrations into executable SOPs through LLM processing, web agent execution, and consistency monitoring feedback loops.

users. While comprehensive instructions can improve task specificity, they often result in brittle solutions that fail when websites undergo minor changes.

To address these limitations, we introduce *Cybernaut*, a framework that provides a robust mechanism for measuring and ensuring high consistency in repeated executions of web automation agents, while simultaneously meeting the stringent accuracy requirements. Cybernaut is built on the principles of demonstration-based learning [3], with a particular emphasis on the challenges encountered in real-world environments. Our solution addresses these limitations through three key innovations: (1) automated generation of high-level execution steps from user demonstrations, (2) robust element detection and interaction handling, and (3) quantitative consistency evaluation across multiple executions. Figure 1 illustrates the architectural overview of Cybernaut's workflow pipeline.

2 Related work

2.1 Web Navigation Agents

Recent advances in LLMs have led to diverse approaches in web agent design, categorized by their user interaction paradigms, including personalization, multi-modality, and grounding mechanisms. These range from conversational agents like WebLINX [12] that frame web navigation as a dialogue task, to API-centric approaches [17] offering structured interfaces between LLMs and web environments. Visual grounding has emerged as a promising direction, with studies like [22] demonstrating GPT-4V's effectiveness in grounding actions through element highlighting. Nova-Act [1] achieve best-in-class performance on benchmarks like ScreenSpot and GroundUI Web which most directly measure the ability of model to actuate the web. Our work builds upon these approaches, particularly focusing on grounding-aided interactive element detection. We introduce novel methods for converting single demonstrations into robust steps that the agent can follow consistently and measure the execution's consistency. This advancement addresses the critical need for consistency and reliability, pushing the field forward in terms of practical applicability and performance.

Browser-Use Framework: Although our framework is agnostic to the underlying web agent, for the purpose of this paper, we use Browser-Use [13], as it is an open-source project that has gained significant traction for web automation. It provides a sophisticated infrastructure for creating AI agents capable of autonomous web navigation, interaction, and information extraction. Our work extends the framework by effectively bridging the gap between user intent and consistent web agent execution.

Perception of Interactive Elements: Graphical User Interface (GUI) perception in LLM-based agents follows two main approaches. Single-modal LLMs [19, 9] use separate modules for GUI processing, while multi-modal LLMs [7, 18] handle visual and textual information in an integrated manner. Recent advancements in UI understanding include Ferret-UI's [21] "any-resolution" approach, OmniParser's [11] vision-based parsing, and Iris's [5] enhanced processing of heterogeneous GUI information, addressing key challenges in annotation bias, modality misalignment, and architectural limitations.

2.2 Benchmarks for Browser and Computer Agents

Web Navigation Benchmarks and Environments: Several benchmarks have been proposed to evaluate LLM-based web agents. WebVoyager [6] focuses on end-to-end web agents navigating real-world websites, and introduces advanced DOM element recognition techniques. VisualWebArena [8] extends this by incorporating multi-modal agents and visual grounding for web tasks. Mind2Web

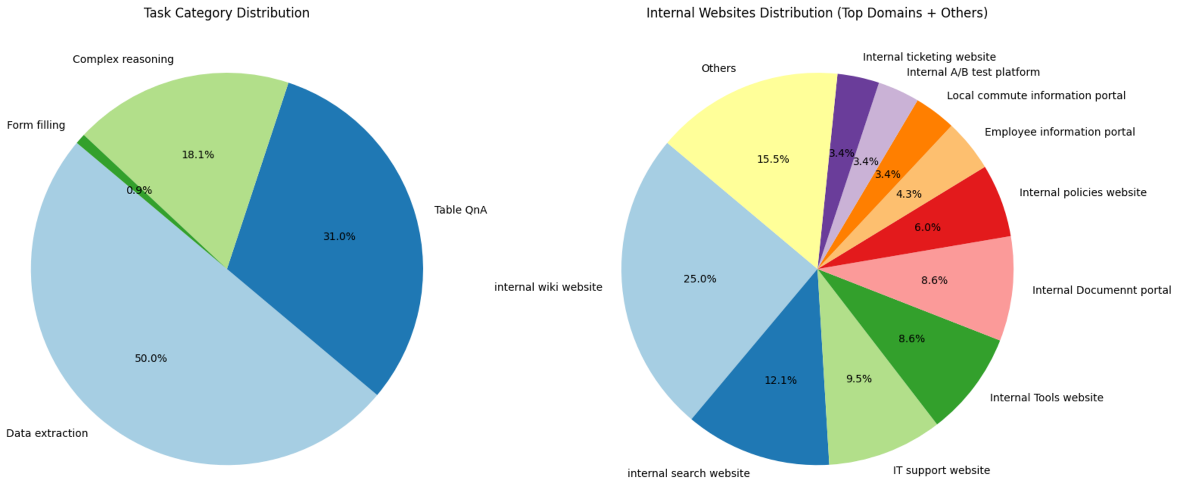

Figure 2: Distribution of tasks across categories and internal website types in the benchmarking dataset. The left pie chart shows the proportion of tasks across four categories: Data extraction, Table QnA, Complex reasoning, and Form filling. The right pie chart highlights the most frequent internal domains accessed, and aggregating low-frequency domains under "Others"

[4] provides a large-scale dataset spanning 137 websites and 31 domains, emphasizing generalist web agents. While these benchmarks primarily evaluate the successful completion of online tasks, some - like Mind2Web - also measure execution trace alignment with human demonstrations.

Mobile and Cross-Platform Environments: AndroidWorld [16] presents a comprehensive benchmark for mobile interfaces, featuring 116 programmatic tasks across 20 real-world Android apps. This environment enables testing under varying conditions through parameterized task generation. The CRAB benchmark [20] extends this to cross-platform scenarios, encompassing both desktop and mobile environments, and encouraging development of generalizable agents.

Execution-Based Evaluation Approaches: Existing approaches differ in how they evaluate agents. Benchmarks like WebVoyager and VisualWebArena focus on end-to-end task completion in live environments, while Mind2Web and AndroidWorld emphasize execution trace-based evaluation against ground-truth demonstrations. The step verification approach (STEVE) by [10] uses screenshots at every step to validate each action. CRAB [20] proposes a graph-based evaluation of the agent trajectory in addition to traditional goal-based evaluations. Our work aligns with approaches for execution-based evaluation by introducing novel metrics measuring automation consistency and reliability in real-world settings.

3 Proposed Methodology

3.1 Internal Web Benchmarking Data

We introduce an internal benchmarking dataset to evaluate agent performance on our company’s internal websites, which are primarily designed for internal operational needs over optimal HTML structure. This dataset enables other teams working on similar tasks to conduct standardized evaluations and make fair comparisons across different agent implementations. Our benchmarking dataset was constructed based on the following principles: selection of universally accessible internal websites with stable and consistent content; inclusion of historically stable tasks to ensure reproducibility; formulation of questions with unambiguous answers directly derivable from the web page; and coverage of diverse interaction types such as form-filling, summarization, table-based querying, and logical reasoning. The dataset consists of 117 tasks distributed across 25 internal domains, providing a representative testbed for evaluating enterprise-grade web automation agents in a zero-shot setting. A detailed breakdown of the dataset, including task category and domain distribution, is illustrated in Figure 2.

A majority of task interactions in the internal dataset are concentrated within high-frequency domains, which support core functions including policy access, IT support, and operational dashboards. Mid-frequency domains and others contribute additional task diversity grounded in everyday workflows. To ensure coverage beyond standardized interfaces, our benchmark includes numerous low-frequency domains (each $< 2\%$), comprising legacy systems, departmental portals, and domain-specific dashboards. Collectively, these long-tail domains ($\simeq 15.5\%$) are essential for evaluating agent generalization and robustness under web interface heterogeneity.

Tasks are grouped into *four* categories based on required reasoning and interaction complexity. (1) **Data Extraction** tasks involve retrieving factual information from structured pages. (2) **Complex Reasoning** tasks require multi-hop inference, such as analyzing hierarchical structures. (3) **Table QnA** targets tabular interfaces, ranging from simple lookups to multi-dimensional aggregation. (4) **Form Filling** assesses the agent’s ability to translate natural language commands into precise UI actions, such as form submissions and event creation. This taxonomy enables a comprehensive evaluation of both the reasoning and execution capabilities of web automation agents across realistic enterprise scenarios.

3.2 Demonstration Learning

Website users are uniquely positioned to understand the optimal sequence of steps $S = s_1, s_2, ..., s_n$ required to complete online tasks. Their practical expertise is invaluable in creating Standard Operating Procedures (SOPs) for Cybernaut's execution. While a straightforward approach would involve users writing detailed textual prompts P to describe the task, this method introduces certain limitations despite its simplicity and accessibility. Users often struggle to determine the appropriate level of detail, and their implicit assumptions or cognitive biases may lead to the omission of critical intermediate steps — steps that are essential for the reliable execution of tasks by LLMs.

To address these limitations, we propose a novel methodology that utilizes LLMs to analyze user-provided task demonstrations. Let $D = (T, E)$ represent a demonstration, where T is the task definition and $E = e_1, e_2, ..., e_n$ is the execution trace containing a sequence of user actions. In this approach, users manually follow their SOP to complete a task and record the sequence of actions taken - herein referred to as the *task execution trace* - for a given input. Alongside this trace, users also provide a high-level task definition that articulates the intended objective or outcome of the task. The LLM processes this demonstration to generate a generalizable SOP template $G(D)$ with placeholder variables $V = v_1, v_2, ..., v_k$. For each new task instance i, the model populates the placeholders with relevant contextual data C_i before initiating execution, resulting in a concrete execution plan $E_i = G(D, C_i)$.

Our approach leverages a custom browser extension that captures the details of web interaction in JSON format with an option to also enable audio and video recording. Although this format supports direct replay, it is brittle in practice — sensitive to dynamic changes in web page structure and unable to generalize across different inputs. To overcome these limitations, we process the recorded JSON through an LLM to generate robust, step-by-step SOP instructions. The transformation function $f : JSON \to SOP$ is implemented through our carefully designed prompting methodology, detailed in appendix B. At present, our framework supports only single demonstrations with linear browsing (non-branching sequential navigation through a website) task executions where $|E| = n$ for some finite n.

A crucial avenue for our future research is the expansion of our framework to incorporate audio and video demonstration data, alongside branched and conditional traces. In this enhanced model, the agent navigation actions will be represented as embeddings to compare execution videos of task instances. The execution paths will be represented as a directed acyclic graph, $G = (V, E)$. Finally, these representations will be fused to enable a more sophisticated and nuanced analysis of user demonstrations, capturing the intricacies of non-linear interactions and conditional behaviors.

3.3 Critical Element Identification

A critical component of Cybernaut's functionality is the generation of an HTML snapshot that accurately identifies all interactive elements on a web page. The identification process relies on matching HTML element roles or tags and applying visibility filters. However, this approach encounters significant limitations when attempting to detect elements within complex web structures, particularly when the target element is obscured by multiple layers of HTML code. A common challenge arises when essential elements are rendered with zero dimensions or are visually hidden due to the prioritization of presentational elements. For instance, input text boxes are frequently overlaid with decorative div and span elements for styling purposes, effectively masking their presence in the DOM hierarchy. This layering technique, while beneficial for achieving desired visual effects, can interfere with automated detection mechanisms. Modern web applications often employ sophisticated CSS and JavaScript frameworks that further complicate this issue by dynamically manipulating element visibility and positioning, making traditional detection methods less reliable.

To overcome these limitations, we developed an approach that leverages JSON data from the demonstration data. We use HTML element attributes and XPath DOM identifiers, to generate task-specific configurations. These configurations are designed to maintain element visibility and interactability, regardless of their initial rendering state in the application. However, using XPath and identifiers directly presents two significant challenges. First, XPath identifiers and element selectors are frequently generated dynamically, resulting in inconsistent values across different sessions. Second, there exists a fundamental mismatch between demonstration and execution phases: during demonstration, users typically interact with visible, parent elements in the interface, while the actual functional elements are often hidden child components.

To systematically identify and enable interactive elements, we employ a web agent to execute the task. At each step, the agent applies a three-stage procedure for element identification, as shown in Figure 3:

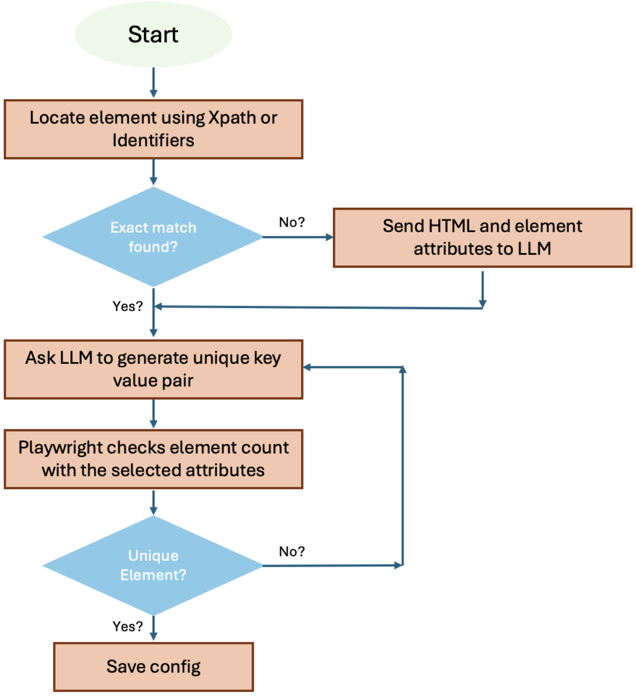
Start
Locate element using Xpath or Identifiers
Exact match found?
No?
Send HTML and element attributes to LLM
Yes?
Ask LLM to generate unique key value pair
Playwright checks element count with the selected attributes
Unique Element?
No?
Yes?
Save config

Figure 3: Critical element identification approach

1. Presence Verification: At current page, we attempt to locate element using XPath and identifiers. If no exact match is found, this means the state has changed and XPaths are not valid anymore. To get the HTML code of element, we submit both the recorded element attributes and the current HTML snapshot to an LLM to perform semantic matching and return HTML code of element that might match this element.

2. Key-Value Signature Assignment: Upon a successful match, now we need to define some unique attributes of the element that we can save as configuration. To do that we employ an LLM to extract a set of stable key-value attribute pairs that uniquely identify the element. Once the attributes are identified, we use playwright to check how many element exists on this page with these attributes. If unique, we save it otherwise LLM is asked to generate new attributes.

3. Configuration Persistence: Finally, we store validated element signatures in a persistent configuration file that is read during execution. This config ensures consistent and automatic visibility toggling in future executions of the same task.

This method enhances element robustness and reduces fragility in web automation, particularly in dynamic or conditionally rendered user interfaces. For example, for a search input element from IMDb's interface:

```
{
    "tag": "input",
    "attributes": {
        "type": "text",
        "aria-label": "Search IMDb",
        "class": "imdb-header-search__input",
        "id": "suggestion-search",
        "data-testid": "suggestion-search"
    }
}
```

Our system analyzes these attributes and generates a minimal yet robust configuration signature:

```
{
    "data-testid": "suggestion-search",
    "aria-label": "Search IMDb"
}
```

3.4 Task Execution Consistency

We define *consistency* as Cybernaut's ability to reproduce similar execution patterns when performing identical tasks with same or varying input parameters. Inconsistent behavior emerges from three key factors: (1) LLM output stochasticity (σ), (2) dynamic form dependencies where $state_{t+1} = f(state_t, action_t)$, and (3) temporal website changes. While existing solutions employ deterministic selectors, prompt engineering, or retry mechanisms, they either sacrifice flexibility for consistency or impose computational overhead. More importantly, they fail to capture the fundamental relationship between successful execution patterns and intended task objectives.

Execution consistency is crucial to optimize Knowledge Operations Work (KOW) with well-defined tasks, as it enables human-level accuracy and streamlines debugging of automated workflows. To address this, we propose a trace-based similarity metric $C(E_t, E_g)$ that measures the alignment between an agent's execution path E_t and a set of reference (golden) traces E_g, enabling systematic validation of task completion patterns and effective intervention during execution.

3.4.1 Definition of Consistency. In web automation, *consistency* refers to the similarity of execution patterns when performing the same task with identical or varying input parameters. We formalize this concept as follows:

Given a task T and two input parameter sets P_1 and P_2, let E_1 and E_2 denote the execution traces corresponding to $T(P_1)$ and $T(P_2)$ respectively. The *consistency score* C between the two executions is defined as:

$$C(E_1, E_2) = S(f(E_1), f(E_2)) \tag{1}$$

Here, f is a feature extraction function, and S is a similarity metric normalized to the interval $[0, 1]$. A score of 1 indicates perfect consistency.

We decompose each execution trace into its constituent steps. For an execution trace E consisting of n steps $[e_1, e_2, \ldots, e_n]$, each step e_i is represented by a feature vector $h(e_i)$, defined as:

$$h(e_i) = [g_i, a_i, \alpha_i] \tag{2}$$

where, $g_i \in G$ denotes Cybernaut's goal state at step i, $a_i \in A$ denotes the action executed at step i, and, $\alpha_i \in \mathbb{R}^d$ is a d-dimensional attribute vector representing the properties of the web element interacted with at step i. The complete feature representation of an execution trace E is then the ordered sequence of its step-level feature vectors: $f(E) = [h(e_1), h(e_2), \ldots, h(e_n)]$. This formalization provides a principled foundation for quantitatively comparing execution behaviors and evaluating the consistency of web automation agents.

3.4.2 Calculating Consistency using LLM. Measuring consistency between execution traces presents unique challenges due to the inherent variability in LLM-driven automation. Even when executing identical tasks, the resulting action sequences may differ superficially while still achieving the same functional outcome. For example, consider two executions of the same form-filling task: **Execution 1:** [click input field → type text], and, **Execution 2:** [directly type text into input field]. Although these sequences produce different feature vectors $f(E_i)$, they are functionally equivalent. Similar variations may arise in scrolling behavior, element selection strategies, and interaction sequences. A robust consistency measure must account for these acceptable variations while detecting true deviations.

A straightforward approach to handle such nuanced comparisons is to leverage LLMs for consistency evaluation. Given two execution traces, an LLM can be prompted to assess their similarity based on the semantics of the actions, rather than exact structural alignment: $C(E_1, E_2) = LLM(f(E_1), f(E_2)) \rightarrow [0, 1]$. This approach benefits from the LLM's natural language understanding capabilities, enabling flexible and semantic comparison without the need for manually engineered rules. It is particularly effective at recognizing variations that preserve task intent, such as reordering of steps or alternative interaction modalities. However, LLM-based evaluation is computationally expensive and may become infeasible at industrial scale, where millions of execution traces are analyzed. Moreover, it is susceptible to non-determinism — repeated evaluations on the same input pair can yield inconsistent scores. Achieving reliable results requires careful prompt engineering and strict control over evaluation conditions, which adds complexity and reduces reproducibility.

3.4.3 Calculating Consistency using Embedding Models. An alternative approach to LLM-based evaluation is the use of *embedding models*, which strike a balance between semantic understanding and computational efficiency. These models encode execution traces into dense vector representations, enabling rapid and scalable similarity comparisons. Given two execution traces E_1 and E_2, the consistency score is defined as:

$$C(E_1, E_2) = \text{Sim}(\text{Embed}(f(E_1)), \text{Embed}(f(E_2))) \tag{3}$$

where $\text{Embed}(\cdot)$ is an embedding function and $\text{Sim}(\cdot, \cdot)$ is a similarity metric such as cosine similarity.

These embedding-based methods offer a promising middle ground between the semantic richness of LLM-based evaluation and the efficiency requirements of real-world systems. They produce deterministic outputs, significantly reduce computational overhead, and support millisecond-level inference times—making them well-suited for high-throughput environments. Once trained, embedding models provide consistent and reproducible evaluation criteria across millions of comparisons. Given these advantages, we adopt the embedding-based approach as our final solution for consistency computation in the proposed workflow.

4 Performance Evaluation

4.1 Experimental Setup

SOP Generation from User Demonstration: We leveraged custom browser extension to capture web interaction sequences in JSON format. The recorded demonstrations along with task definitions were then processed by an LLM (Clause 3.7) to generate structured SOPs.

Embedding Model Data Collection and Preparation: To fine-tune the consistency embedding model, we constructed a labeled dataset by executing each task 10 times using Cybernaut. We then manually reviewed and paired the resulting execution traces, annotating each pair as either *similar* or *dissimilar*. The fine-tuning dataset includes 494 similar and 322 dissimilar pairs, derived from 80 executions across 8 distinct task types. Labeling was based on the principle of functional equivalence rather than exact action sequence matching. Non-critical variations - such as additional scrolls, redundant clicks, or differences in timing, were deliberately ignored. Instead, our annotations prioritized the "spirit of execution", aiming to capture whether the agent achieved the same outcome through a logically coherent and consistent behavior pattern.

Embedding Model Training using Siamese Network with Contrastive Loss: We employed a Siamese network architecture to fine-tune the consistency embedding model. Each execution trace pair was processed through identical network branches that produced fixed-length 768-dimensional embedding via mean pooling over step-level feature vectors. The network was trained using a contrastive loss function, which minimized the distance between embedding of similar traces while maximizing it for dissimilar ones. Fine-tuning was performed using *all-distilroberta-v1* over 816 labeled pairs for 3 epochs, using a learning rate of 5e-5 and a weight decay of 0.01. We used a binary classification evaluator for validation. This approach enabled the model to capture semantically meaningful patterns in execution behavior, forming a robust basis for consistency scoring.

Cybernaut Agent Setup: We extended the `Browser-Use` framework (version 0.1.40) to support task reproducibility and consistency monitoring. However, our solution can be extended to any web agent. Enhancements include (1) configuration support for saving interacted element signatures during onboarding, (2) customized tool invocations, (3) improved answer validation, and (4) robust URL handling. All LLM-driven operations inside cybernaut were powered by Claude 3.7, ensuring uniform behavior across planning and execution components.

4.2 Results

4.2.1 Task Completion Evaluation. In this section, we present the evaluation results of Cybernaut on our company's internal benchmark, highlighting the impact of demonstration-based SOP generation and critical element handling on task completion accuracy. For the public WebVoyager benchmark [6], we did not generate demonstrations, as most tasks lack clearly defined ground truth and reproducible step sequences. Nevertheless, for completeness, we evaluate Cybernaut (configured without SOPs but with critical element handling) on this benchmark and compare its performance against the Browser-Use baseline. Cybernaut achieves comparable accuracy (80.3% v/s 82.2% for SOTA) on the WebVoyager benchmark, despite the absence of demonstrations. For further details, please refer to section 4.3.

Table 1 summarizes the accuracy improvements on the internal benchmark dataset. The integration of SOP alone yields a 13.9% accuracy improvement (from 72% to 82.02%) over the state-of-the-art (SOTA) baseline `Browser-Use`. Further incorporating the critical element detection and handling module leads to an additional 9.3% gain over SOTA, resulting in a final accuracy of 88.68%.

Table 1: Cybernaut accuracy on internal benchmark

Model	Accuracy
Browser-Use	72.00%
Cybernaut with SOP	82.02%
Cybernaut with SOP + Critical element fix	**88.68%**

4.2.2 Task Consistency Evaluation. Table 2 presents the results of the consistency model evaluated on a manually labeled validation dataset. The out-of-the-box model performs poorly, requiring an impractically high prediction threshold of 99.8% to achieve 71.1% accuracy and 72.6% F1 score, due to its lack of prior exposure to structured web execution traces. In contrast, after fine-tuning, the Siamese model achieves superior performance with 84.7% accuracy and 87.3% F1 score at a more reasonable threshold of 81.1%, demonstrating its enhanced capability to differentiate between consistent and inconsistent execution patterns.

Table 2: Consistency model performance on validation set

Model Type	Prediction threshold	Accuracy	F1 Score
Out-of-the-box	99.8%	71.1%	72.6%
Fine-tuned	81.1%	84.7%	87.3%

Table 3: Execution consistency analysis: similarity score comparison

Execution Type	Average Similarity Score (%)
Consistent	90.97
Inconsistent	60.60

Note: Similarity scores computed using cosine similarity between execution traces. Consistent executions demonstrate significantly higher similarity scores, indicating reliable task reproduction.

During run-time, the consistency model will compare execution traces with reference traces to assess similarity. Analysis of cosine similarities between known executions shows consistent cases achieving 90.97% similarity versus 60.60% for inconsistent ones as shown in Table 3. This margin demonstrates the model's ability to differentiate equivalent executions, validating its use in system monitoring.

4.3 Performance on WebVoyager Public Benchmark Dataset

For the WebVoyager benchmark dataset (643 tasks across 15 websites) [6], we did not generate demonstrations, as most tasks lack a clearly defined ground truth and reproducible step sequences. In many cases, the clicked links vary based on external factors such as product updates (e.g., change in airpods version) and dynamic recommendation changes for the task (e.g., different headphones recommendation). Nevertheless, for completeness, we evaluate Cybernaut on this benchmark and compare it with the Browser-Use baseline. As shown in Table 4, Cybernaut achieves an accuracy of 80.3%, representing a marginal 2.37% drop compared to the SOTA baseline. A detailed analysis reveals that this discrepancy is not concentrated in any specific domain. In fact, several tasks are successfully completed by Cybernaut but not by Browser-Use, and vice versa. The variations are primarily attributable to non-determinism in LLM behavior, CAPTCHA blocks on external websites, and dynamic content differences.

Table 4: Cybernaut accuracy on Webvoyager benchmark

Model	Accuracy
Browser-Use	**82.20%**
Cybernaut	80.30%

5 Conclusion and Future Work

In this paper, we presented *Cybernaut*, an advanced framework for web automation AI agent designed to address critical challenges in enterprise environments. Empirical evaluation demonstrated an 88.68% task completion rate on internal benchmarks, representing a 23.2% improvement over baseline methods. Our consistency evaluation method, powered by a fine-tuned embedding model, effectively distinguishes between consistent and inconsistent execution patterns with 84.7% accuracy, enabling reliability at industrial scale. These results establish Cybernaut as a robust and and generalizable solution for highly accurate and consistent enterprise-grade web automation, and provide a foundation for future advancements.

Future work will explore *multi-step demonstration learning* to handle more complex workflows involving conditional execution traces, as well as the integration of visual information (e.g., page screenshots) alongside user recorded JSON to improve element recognition accuracy. Additionally, we plan to investigate graph-based approaches for modeling execution path structures and transitions, and to enhance our consistency evaluation framework using visual context to better capture UI state dynamics. Also, we will extend consistency metrics to run during execution and nudge model in right direction if model deviates from previous consistent paths.

References

[1] Amazon. 2025. Nova act. (2025). https://nova.amazon.com/act.

[2] Anthropic. 2024. Computer use (beta). Anthropic, (Oct. 2024). https://docs.anthropic.com/en/docs/agents-and-tools/computer-use.

[3] André Correia and Luís A. Alexandre. 2023. A survey of demonstration learning. *arXiv preprint arXiv:2303.11191*. https://arxiv.org/abs/2303.11191.

[4] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*. A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors. Vol. 36. Curran Associates, Inc., 28091–28114. https://proceedings.neurips.cc/paper_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf.

[5] Zhiqi Ge et al. Iris: breaking gui complexity with adaptive focus and self-refining. arXiv Preprint, (2025). https://arxiv.org/abs/2412.10342 arXiv: 2412.10342 [cs.CV].

[6] 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)*. Association for Computational Linguistics, Bangkok, Thailand, (Aug. 2024), 6864–6890. doi:10.18653/v1/2024.acl-long.371.

[7] Wenyi Hong et al. Cogagent: a visual language model for gui agents. arXiv Preprint, (2024). https://arxiv.org/abs/2312.08914 arXiv: 2312.08914 [cs.CV].

[8] Jing Yu Koh et al. Visualwebarena: evaluating multimodal agents on realistic visual web tasks. arXiv Preprint, (2024). https://arxiv.org/abs/2401.13649 arXiv: 2401.13649 [cs.LG].

[9] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences. arXiv Preprint, (2020). https://arxiv.org/abs/2005.03776 arXiv: 2005.03776 [cs.CL].

[10] Fanbin Lu, Zhisheng Zhong, Ziqin Wei, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Steve: a step verification pipeline for computer-use agent training. arXiv Preprint, (2025). https://arxiv.org/abs/2503.12532 arXiv: 2503.12532 [cs.CV].

[11] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent. arXiv Preprint, (2024). https://arxiv.org/abs/2408.00203 arXiv: 2408.00203 [cs.CV].

[12] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: real-world website navigation with multi-turn dialogue. arXiv Preprint, (2024). arXiv: 2402.05930 [cs.CL].

[13] Magnus Müller and Gregor Žunič. 2024. Browser use: enable ai to control your browser. (2024). https://github.com/browser-use/browser-use.

[14] Liangbo Ning et al. 2025. A survey of webagents: towards next-generation ai agents for web automation with large foundation models. *arXiv preprint arXiv:2503.23350*. https://arxiv.org/abs/2503.23350.

[15] OpenAI. 2025. Computer-using agent. OpenAI, (2025). https://openai.com/index/computer-using-agent/.

[16] Christopher Rawles et al. Androidworld: a dynamic benchmarking environment for autonomous agents. arXiv Preprint, (2025). https://arxiv.org/abs/2405.14573 arXiv: 2405.14573 [cs.AI].

[17] Yueqi Song, Frank Xu, Shuyan Zhou, and Graham Neubig. Beyond browsing: api-based web agents. arXiv Preprint, (2025). https://arxiv.org/abs/2410.16464 arXiv: 2410.16464 [cs.CL].

[18] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: autonomous multi-modal mobile device agent with visual perception. arXiv Preprint, (2024). https://arxiv.org/abs/2401.16158 arXiv: 2401.16158 [cs.CL].

[19] Hao Wen et al. Autodroid: llm-powered task automation in android. arXiv Preprint, (2024). https://arxiv.org/abs/2308.15272 arXiv: 2308.15272 [cs.AI].

[20] Tianqi Xu et al. Crab: cross-environment agent benchmark for multimodal language model agents. arXiv Preprint, (2024). https://arxiv.org/abs/2407.01511 arXiv: 2407.01511 [cs.AI].

[21] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: grounded mobile ui understanding with multimodal llms. arXiv Preprint, (2024). https://arxiv.org/abs/2404.05719 arXiv: 2404.05719 [cs.CV].

[22] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. arXiv Preprint, (2024). https://arxiv.org/abs/2401.01614 arXiv: 2401.01614 [cs.IR].

A User demonstration to step-by-step SOP instructions

To convert user demonstrations into step-by-step SOP instructions, we utilize an LLM that takes as input a high-level task definition and a browser-recorded execution trace in JSON format. As shown in Listing 1, users perform the task manually following the SOP, while their interactions are captured via Chrome's recorder. The LLM analyzes this interaction trace along with the task objective (e.g., identifying reusable laptop models within a specified age range) and, using a prompt template 2, generates a generalizable, step-by-step SOP 3. The prompt guides the model to infer user intent, abstract relevant actions, and exclude irrelevant or redundant interactions, resulting in a clear and reusable instruction set. This methodology enables precise documentation of operational workflows, facilitating reliable automation by downstream web agents.

```
{
    "title": "Recording IMDB",
    "steps": [
        {
            "type": "setViewport",
            "width": 1173,
            "height": 901,
            "deviceScaleFactor": 1,
            "isMobile": false,
            "hasTouch": false,
            "isLandscape": false
        },
        {
            "type": "navigate",
            "url": "https://www.imdb.com/",
            "assertedEvents": [
                {
                    "type": "navigation",
                    "url": "https://www.imdb.com/"
    ,
                    "title": ""
                }
            ]
        },
        {
            "type": "click",
            "target": "main",
            "selectors": [
                [
                    "aria/All",
                    "aria/[role=\"none\"]"
                ],
                [
                    "span.sc-bBjSGg svg"
                ],
                [
                    "xpath///*[@data-testid=\"
    category-selector-button\"]/svg"
                ],
                [
                    "pierce/span.sc-bBjSGg svg"
                ]
            ],
            "offsetY": 11,
            "offsetX": 2.1796875
        },
        {
            "type": "click",
            "target": "main",
            "selectors": [
                [
                    "#suggestion-search-container
    a > span.ipc-list-item__text"
                ],
                [
                    "xpath///*[@id=\"nav-search-
    form\"]/div[1]/div/div/div/ul/a/span[1]"
                ],
                [
                    "pierce/#suggestion-search-
    container a > span.ipc-list-item__text"
                ],
                [
                    "text/Advanced Search"
                ]
            ],
            "offsetY": 12.359375,
            "offsetX": 97.8671875,
            "assertedEvents": [
                {
                    "type": "navigation",
                    "url": "https://www.imdb.com/
    search/title/?ref_=nv_sr_menu_adv",
                    "title": ""
                }
            ]
        },
        {
            "type": "click",
            "target": "main",
            "selectors": [
                [
                    "aria/Expand all",
                    "aria/[role=\"generic\"]"
                ],
                [
                    "div.sc-ed40b8bf-0 > div span"
                ],
                [
                "xpath///*[@data-testid=\"adv-
search-expand-all\"]/span"
            ],
            [
                "pierce/div.sc-ed40b8bf-0 >
div span"
            ],
            [
                "text/Expand all"
            ]
        ],
        "offsetY": 11,
        "offsetX": 60.7578125
    },
    {
        "type": "click",
        "target": "main",
        "selectors": [
            [
                "aria/Enter release date above
[role=\"textbox\"]"
            ],
            [
                "[data-testid='
releaseYearMonth-start']"
            ],
            [
                "xpath///*[@data-testid=\"
releaseYearMonth-start\"]"
            ],
            [
                "pierce/[data-testid='
releaseYearMonth-start']"
            ]
        ],
        "offsetY": 35,
        "offsetX": 92.5
    },
    {
        "type": "change",
        "value": "2020-01",
        "selectors": [
            [
                "aria/Enter release date above
[role=\"textbox\"]"
            ],
            [
                "[data-testid='
releaseYearMonth-start']"
            ],
            [
                "xpath///*[@data-testid=\"
releaseYearMonth-start\"]"
            ],
            [
                "pierce/[data-testid='
releaseYearMonth-start']"
            ]
        ],
        "target": "main"
    },
    {
        "type": "click",
        "target": "main",
        "selectors": [
            [
                "aria/Enter release date below
[role=\"textbox\"]"
            ],
            [
                "[data-testid='
releaseYearMonth-end']"
            ],
            [
                "xpath///*[@data-testid=\"
releaseYearMonth-end\"]"
            ],
            [
                "pierce/[data-testid='
releaseYearMonth-end']"
            ]
        ],
        "offsetY": 27,
        "offsetX": 33.90625
    },
    {
        "type": "change",
        "value": "2020-12",
        "selectors": [
            [
                "aria/Enter release date below
[role=\"textbox\"]"
            ],
            [
                "[data-testid='
releaseYearMonth-end']"
            ],
            [
                "xpath///*[@data-testid=\"
releaseYearMonth-end\"]"
            ],
            [
                "pierce/[data-testid='
releaseYearMonth-end']"
            ]
        ],
        "target": "main"
    },
    {
        "type": "click",
        "target": "main",
        "selectors": [
            [
                "[data-testid='autosuggest-
input-test-id-languages']"
            ],
            [
                "xpath///*[@data-testid=\"
autosuggest-input-test-id-languages\"]"
            ],
            [
                "pierce/[data-testid='
autosuggest-input-test-id-languages']"
            ]
        ],
        "offsetY": 12,
        "offsetX": 116.5
    },
    {
        "type": "change",
        "value": "japa",
        "selectors": [
            [
                "[data-testid='autosuggest-input-test-id-languages']"
            ],
            [
                "xpath///*[@data-testid=\"autosuggest-input-test-id-languages\"]"
            ],
            [
                "pierce/[data-testid='autosuggest-input-test-id-languages']"
            ]
        ],
        "target": "main"
    },
    {
        "type": "click",
        "target": "main",
        "selectors": [
            [
                "aria/Japanese"
            ],
            [
                "[data-testid='checkbox-test-id-ja']"
            ],
            [
                "xpath///*[@data-testid=\"checkbox-test-id-ja\"]"
            ],
            [
                "pierce/[data-testid='checkbox-test-id-ja']"
            ]
        ],
        "offsetY": 26,
        "offsetX": 28.5
    },
    {
        "type": "click",
        "target": "main",
        "selectors": [
            [
                "aria/See results"
            ],
            [
                "[data-testid='adv-search-get-results']"
            ],
            [
                "xpath///*[@data-testid=\"adv-search-get-results\"]"
            ],
            [
                "pierce/[data-testid='adv-search-get-results']"
            ]
        ],
        "offsetY": 29,
        "offsetX": 60.5
    },
    {
        "type": "click",
        "target": "main",
        "selectors": [
            [
                "aria/Sort by"
            ],
            [
                "#adv-srch-sort-by"
            ],
            [
                "xpath///*[@id=\"adv-srch-sort-by\"]"
            ],
            [
                "pierce/#adv-srch-sort-by"
            ],
            [
                "text/POPULARITY"
            ]
        ],
        "offsetY": 17.2734375,
        "offsetX": 61.3515625
    },
    {
        "type": "change",
        "value": "USER_RATING_COUNT",
        "selectors": [
            [
                "aria/Sort by"
            ],
            [
                "#adv-srch-sort-by"
            ],
            [
                "xpath///*[@id=\"adv-srch-sort-by\"]"
            ],
            [
                "pierce/#adv-srch-sort-by"
            ],
            [
                "text/POPULARITY"
            ]
        ],
        "target": "main"
    },
    {
        "type": "click",
        "target": "main",
        "selectors": [
            [
                "aria/Ascending sort order",
                "aria/[role=\"none\"]"
            ],
                [
                    "[data-testid='test-sort-order
    '] > svg"
                ],
                [
                    "xpath///*[@data-testid=\"test
    -sort-order\"]/svg"
                ],
                [
                    "pierce/[data-testid='test-
    sort-order'] > svg"
                ]
            ],
            "offsetY": 15,
            "offsetX": 9.1796875
        }
    ]
}
```

Listing 1: A sample execution trace from Recorder (JSON)

B Prompt Template to generate SOP from user demonstration

```
prompts:
  generator: |
    <role>
    You are a professional operations manager whose
expertise is to document Standard Operating Procedure (SOP)
in a clear and precise manner.
    This SOP will be used as a step-by-step instruction for
an AI-powered browsing agent to complete similar tasks.
    </role>

    You are now given a demo of the operation procedure
performed by a human associate for the following task
described within <task_description> tags:
    <task_description>
    <INPUT_TASK_DESCRIPTION_EXAMPLE>
    </task_description>

    The demo peration procedure is recorded as a browser
replay in .json format within the <browser_replay_in_json>
tags:
    <demo>
    <browser_replay_in_json>
    <TEXT_REPLAY>
    </browser_replay_in_json>
    </demo>

    You are asked to compose an SOP which an AI-powered
browsing agent can follow and complete similar tasks within
the <task_for_sop> tags below:
    <task_for_sop>
    <INPUT_TASK_DESCRIPTION_GENERAL>
    </task_for_sop>

    Below is the requirement of what should be included in
the SOP that you are going to compose:
    <requirement>
    1. For the first step in your SOP:
      - Use both website name and the exact URL in your
instruction
    2. For the second step and onwards in your SOP:
      - Look holistically at the demo and identify the
intention and goal behind each browsing action and step
recorded.
      - Use the intention and purpose behind to guide your
composition of SOP.
      - If navigating to a specific website is a critical
action to achieve the goal, always include the web page name
 and the exact URL of this navigatio action.
      - For other actions, e.g., mouse clicks, keyboard
inputs, that are related to the task, include them as
illustration examples in your instrcution.
    3. Your SOP should only include the knowledge that can
be drawn from the demo replay within <demo> tags.
      - Do not come up with an SOP from your memory.
    4. Exclude steps within <demo> tags that are unrelated
to the task within <task_for_sop> tags. Examples of such
steps are:
      - Steps related to solving CAPTCHA.
      - Steps related to close pop up windows.
      - Mouse clicks on non-interactable elements such as
background, or plain text.
    </requirement>

    You should follow the formatting instructions below to
provide an SOP as your final answer:
    <format_instructions>
    1. Using <sop> tags to include all contents below.
    2. Restate the task description content within <
task_for_sop> tags above, now using <task> tags.
    3. <task_for_sop> may be a general version of the <demo>
 example. Please identify proper input parameters so that <
task_for_sop> can be faithfully represented.
      - Format the input parameters as .json within <
input_param> tags. E.g., {input_param_1: "Explanation"}.
      - If no input parameter needed, output an empty dict
within <input_param> tags. I.e., {}.
    4. Document your final SOP within <instructions-step-by-
step> tags.
      - Only provide the instructions within <instructions-
step-by-step> tags. No need to explain within <instructions-
step-by-step> tags.
      - Use proper referece to the input parameters
identified. E.g., using <INPUT_PARAMETER_1>
      - Using numbered points to organize your sop.
    </format_instructions>

  guardrail: |
    Additional guardrails that you are asked to follow:
    <guardrails>
    1. Only navigate the websites you are in. Do not open
any URL from your memory.
    </guardrails>

  output_format: |
    Below are the output formatting requirements:
    <output_format>
    1. Provide your final answer to the task within <
final_answer> tags.
```

```
      - If you cannot find the answer related to the <task>
from your browsing activities, say "No answer found". Do not
 make up one yourself based on your prior knowledge.
    2. Provide your thoughts within <thoughts> tags.
    </output_format>

  prompt_head: |
    You are a browsing agent and are asked to perform a web
browsing task described within <task> tags below.

    You are ask to act like a human associate who will
strictly follow the instructions within <instructions-step-
by-step> tags, using the appropriate input parameter values
provided within the <input_param> tags.
```

Listing 2: Prompt Template (YAML) for converting User Demonstration to SOP

C Sample SOP generated from the prompt

```

<sop>
<task>
Navigate to https://www.imdb.com/?ref_=nv_home and give me
top 20 movies and shows in Japanese language with highest
number of ratings in year 2020.
</task>

<input_param>
{
   "language": "Japanese",
  "year": "2020",
  "sort_by": "USER_RATING_COUNT"
}
</input_param>

<instructions-step-by-step>
1. Navigate to https://www.imdb.com/?ref_=nv_home
2. Click on the "All" dropdown menu at the top of the page (
usually located in the search bar area)
3. Click on "Advanced Search" from the dropdown menu
4. In the Advanced Search page, first click on Expand all to
 access all search filters and then click on "Expand Release
 date" section
5. Enter "<year>-01" and "<year>-12" in the "Enter release
date above" field
6. In the language search field, type "<language>" (e.g., "
Japanese")
7. Select the checkbox for "<language>" from the dropdown
results
8. Click on the "See results" button
9. On the results page, click on the "Sort by" dropdown menu
10. Select "<sort_by>" from the dropdown options (e.g., "
USER_RATING_COUNT")
11. If needed, click on the sort order button to change from
 ascending to descending order
12. The page will now display the top 20 movies from <
country> in <language> language with the highest number of
ratings in <year>
</instructions-step-by-step>
</sop>
```

Listing 3: User Demonstration translated to a sample robust SOP

Received 10 Jun 2025