Agents on a LEASH: A Case Study in Micro-Managing Web Agent Behavior

Matthew Michelson, Adrien Bibal, Steven Minton

InferLink Corporation
2361 Rosecrans Avenue, Suite 348
El Segundo, California 90245
mmichelson@inferlink.com, abibal@inferlink.com, sminton@inferlink.com

Abstract

AI web agents that perform tasks on behalf of users will transform business through massive productivity gains via automation of specific tasks using a web browser. However, current commercial agents and agentic frameworks offer few guarantees that erroneous behavior will be limited. In this paper, we present LEASH, a framework that borrows from planning and focuses on defining micro-tasks for an agent that consists of both sensing and actions. By pre-sensing before an action and post-sensing after, the agent can drastically increase the likelihood of accurate behavior and mitigate risk. We present a real use-case where a LEASH agent performs autonomous pre-screening tasks of articles submitted to the Journal of AI Research (JAIR). The LEASH agent performs the tasks accurately, at scale, and without harm. We compare this to a commercially available agent which may perform well, at times, but which also exhibited grossly uncontrolled behavior.

Supplementary Material —
https://zenodo.org/records/19008844

Introduction

In the modern era of AI and agents, there is a race to put tools into production environments to assist humans in knowledge work tasks and increase business productivity. This era of agents differs from traditional software. Traditionally software behavior is programmatically defined and therefore limited in its range of action. In contrast, agents act independently on the behalf of the user to perform tasks without requiring the user to explicitly define how to do them. One of such agents are web agents, which autonomously manipulate a web browser to fulfill a task (see, e.g., WebVoyager from He et al. (2024)). As a concrete example of the difference between the functioning of traditional software and AI web agents, consider the interaction with a form on a website. A traditional software program might POST directly to the form, using HTTP calls that are constructed specifically for that application. An AI web agent, on the other hand, would actually type into the input boxes and then click the login button, in that sequence (much like a person). The common workflow with such web agents is to define a high level goal for the AI and let the agent "fill in the rest." As a running example throughout this paper, consider the "screening" process applied to new submissions at the Journal of AI Research (JAIR). Like many other academic journals, there are some baseline criteria that must be met *before* a paper is sent out for peer review. At JAIR, this includes confirming that the PDF file submitted by the authors is in the proper format, verifying that the submitter has properly entered all the author's names into the JAIR submission site system, and checking several other basic requirements.

This pre-screening is currently done by hand, which is an onerous task. This is neither a good use of time for journal personnel nor is it interesting. Further, the task, while similar each time, is diverse enough that it would be challenging to solve with traditional software. For instance, ensuring that the paper meets the JAIR formatting standards would be very difficult to do without AI, because it's fundamentally a stylistic check: does the paper layout look correct, are the margins and fonts matching, etc. The similar-yet-diverse nature of the problem actually makes it ripe for an AI agent.

In most common commercial agentic AI frameworks, performing these pre-review checks would involve crafting a high-level prompt that instructs the system to navigate the JAIR website for new submissions and evaluate them against the pre-screening criteria. However, this high-level approach has several important limitations.

First, and most importantly, by defining the tasks at such a high level, there is ample opportunity for the agent to actually do less than expected, or to incorrectly interpret what was requested, or even to execute unanticipated actions (which can be dangerous). The more room is left for the AI to make choices about actions, the more potential room there is for mistakes. And these mistakes could lead to behavior with significant negative consequences. We call this the "Don't Press the Red Button" problem. Intuitively, there are certain consequences to an agent's mistake that can be serious (e.g., pushing that button). More formally, "Don't Press the Red Button" is defined as having an agent engage in high-impact, but unintended actions, often with unsafe side effects (e.g., unauthorized API calls, irreversible state changes, etc.). For instance, imagine an AI web agent begins to click random buttons on the Webpage or invent URLs to hit or, in fact, make API calls (we have examples of just such behavior in our Results below). In the case of JAIR,

we never want the agent to "accidentally" reject a paper by clicking the "Reject" button.

One way to conceptualize this issue is that, fundamentally, agentic systems are given a technical specification for a task (in the form of a prompt) and the AI's goal is to turn that specification into an implementation (the steps the agent will take). In general, the more narrowly we define the specification, the less room for improvisation in the implementation (and therefore fewer potential mistakes).

While the "Don't Press the Red Button" issue is significant, there is another issue associated with AI agents that work at such a high level. Specifically, they don't offer granular control for logging and error handling, the way traditional software systems do. Enterprise software often has a requirement to understand if and when errors occurred, with enough logging to aid in determining root causes (and potential fixes). One way to address this is to specify the task in terms of a series of micro-specifications, rather than one large one, so that the agent can report its performance (or errors) for each step associated with each specification.

Finally, as software developers, the ability to estimate errors and program for them is a key concept in good software engineering. We write tests and checks precisely to mitigate various failure states we believe can happen. This is especially challenging when working with (or developing) agents, as the space of failures is much larger. For instance, in a traditional software, ensuring that a value returned by a function is a number is a clear programming task that one can develop for (e.g., with data integrity checks and tests). With AI agents, predicting if the agent will navigate itself to a strange part of a website and begin to enter data into forms is much harder to predict and prepare for in code. And again, the space of potential errors to consider increases the higher the level of the specification of the task.

To address these issues, in this paper we present the LEASH framework (**L**ogical **E**nforcement and **A**lignment of **S**afety **H**arnesses). LEASH is based upon techniques from planning and robotics and is defined by the following key principles:

- Micro tasks are more easily defined and managed than high level tasks.
- Pre-Sensing allows the agent to confirm what it will do, ahead of an action (e.g., is there even a Login button on this page?)
- Post-Sensing checks verify that the action was performed correctly (e.g., the value for the username is indeed in the Username box on the form)
- Logging and graceful error handling should be done as in traditional software (we can wrap micro-tasks in appropriate logging and error handling components)

A key component of LEASH is the nature of the micro tasks. While one could conceivable specify all microtasks with traditional programming, we specifically focus on micro-tasks that are agentic in nature themselves (e.g., a micro-task would be "Click the Login button" in contract to a regularly programmed step that would identify form elements using XPath and submit them with a POST API call). This is a design decision and there are alternatives. Consider the step of logging into JAIR. Of course, this could be done with traditional methods by performing a POST HTTP call with the appropriate form parameters filled in. In LEASH, we perform this task agentically. As described below, one microtask identifies the Username field and fills it in (e.g., by typing), another does so for the Password field, and yet another task then clicks the Login button, all autonomously. Each of these microtasks involves pre- and post-sensing to ensure correctness of the action and the state of the agent. While in some cases this involves more overhead than programming, such as the case with a simple POST, the benefit to treating these steps with an AI web agent is that even if the website format changes (for instance, renaming the form elements), the web agent still works, while traditional programming will not be robust enough to handle such changes. On the flip side, a challenge of LEASH is that the agent may make mistakes that simple code would not (for instance, it will fail to identify the Login button to click, on rare occasions). In general, the benefits of agents far outweigh traditional software for many tasks. There are simply certain tasks that would be extremely difficult to write in traditional code, such as determining if the PDF format matches a stylistic template.

LEASH is formulated as a program in Backus-Naur Form (BNF) (Backus 1959; Backus et al. 1960). That is, we have a Goal, State, Microtasks (each of which has a pre-sensing, action and post-sensing), as well as potential conditionals (such as IF or NOT) and loops. With these primitives, we define LEASH with the following BNF syntax:

$$
\begin{aligned}
\langle \text{LEASH} \rangle &::= \langle \text{goal} \rangle\ \langle \text{state} \rangle\ \langle \text{sequence} \rangle \\
\langle \text{sequence} \rangle &::= \langle \text{statement} \rangle \mid \langle \text{statement} \rangle\ \langle \text{sequence} \rangle \\
\langle \text{statement} \rangle &::= \langle \text{microtask} \rangle \mid \langle \text{conditional} \rangle \\
&\quad \mid \langle \text{loop} \rangle \mid \langle \text{state-update} \rangle \\
\langle \text{microtask} \rangle &::= \langle \text{pre-sensing} \rangle\ \langle \text{action} \rangle\ \langle \text{post-sensing} \rangle \\
\langle \text{conditional} \rangle &::= \textbf{IF}\ \langle \text{condition} \rangle\ \textbf{THEN}\ \langle \text{sequence} \rangle \\
&\quad [\textbf{ELSE}\ \langle \text{sequence} \rangle] \\
\langle \text{loop} \rangle &::= \textbf{FOR EACH}\ \langle \text{item} \rangle\ \textbf{IN}\ \langle \text{collection} \rangle \\
&\quad \textbf{DO}\ \langle \text{sequence} \rangle \\
\langle \text{item} \rangle &::= \langle \text{state-var} \rangle \mid \langle \text{dom-node} \rangle \mid \langle \text{record} \rangle \\
\langle \text{collection} \rangle &::= \langle \text{item} \rangle \mid \langle \text{item} \rangle\ \langle \text{collection} \rangle \\
\langle \text{condition} \rangle &::= \langle \text{eval} \rangle \mid \textbf{NOT}\ \langle \text{condition} \rangle \\
&\quad \mid \langle \text{condition} \rangle\ \textbf{AND}\ \langle \text{condition} \rangle \\
\langle \text{state-update} \rangle &::= \langle \text{state-var} \rangle \leftarrow \langle \text{value} \rangle
\end{aligned}
\tag{1}
$$

To make this concrete, as described above, consider the specifics of the JAIR pre-screening task. As described previously, JAIR pre-screening involves a series of checks to ensure that incoming submissions meet the minimum requirements prior to being sent out for peer review. This involves logging into the Web app for JAIR's OJS (Open Journal Systems) deployment, identifying the incoming papers, and making judgments regarding the basic requirements, including the following:

1. The submitted paper must be a PDF

2. The PDF must conform to the journal's stylistic standard (template)
3. All of the authors listed in the PDF must also be listed as contributors in the journal's submission system (and in the same order)
4. The authors who submitted the paper must have appropriately submitted answers to all of the pre-screening questions that are presented during the submission processes (e.g., why is this work important, what other papers are similar, have you submitted this work elsewhere)

More concretely, examples of how agents for the assessment of papers in JAIR can be defined as LEASH programs are shown in Figures 1 and 2.

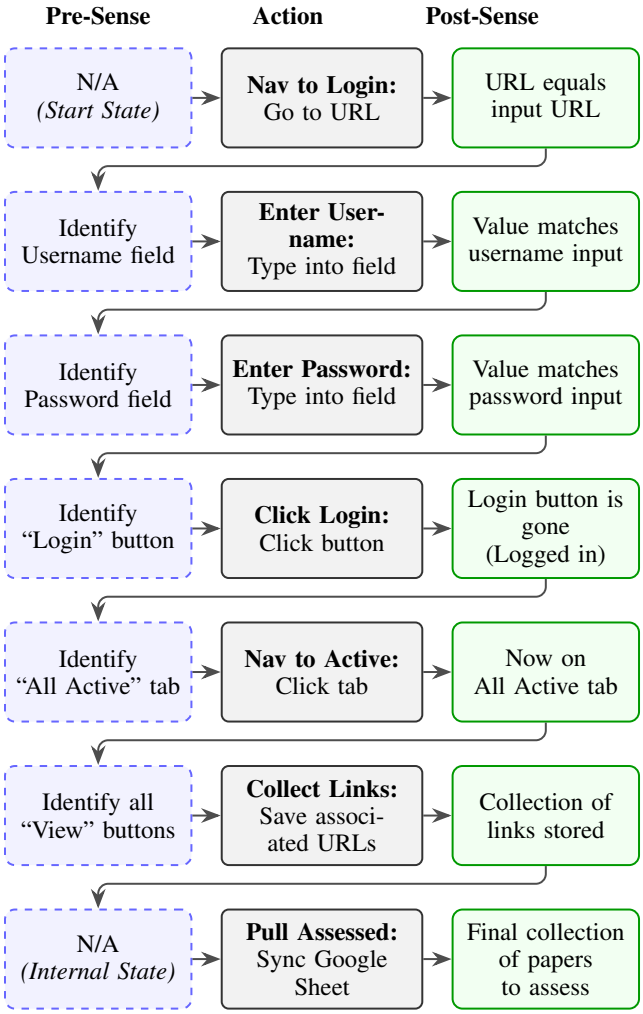

Figure 1: Example of JAIR sub-agent defined as a LEASH program for pulling papers. The flowchart highlights the strict "sense-act-sense" execution loop, ensuring state verification before and after every action.

The ideas underlying LEASH are not new, though applying them to AI agents is. LEASH builds upon the history of sense-act-sense (Albus 2002) (or sense-plan-act-verify (Rojas et al. 2016)) that has become commonplace in robotics (see Related Work). Across various approaches (such as shielding (Alshiekh et al. 2018)), the goal is to prevent a robot from performing an unsafe action, which is achieved through careful sensing both ahead of and after a given action (with differing approaches, such as learning to avoid bad actions in a policy or halting them from occurring at runtime).

As with robotics, the use of the pre-sensing and post-sensing, combined with breaking the task into microtasks is what puts the safety rails into the framework. To make this concrete for Web agents, consider a Pre-Sensing check when filling in a username as part of logging into a website. If the agent is supposed to supply a username into the Username box on a form, it must first identify that a Username box exists on the page. This pre-sensing validation serves to ensure that the agent is on the right page in the first place, the form exists, and it has a Username input field. The action then fills that input box with the correct username. Once that action is performed, the Post-sensing check ensures that the value written into the Username input box matches what was expected (e.g., identify the Username box, pull out the value there, and match it to the expected value).

In this way, LEASH can capture a number of errors. If the agent is on the wrong page, the Pre-Sensing check will fail as there is no username. If the action step fails to enter a username (e.g., due to misidentification of the input box) or fails to enter the correct value, the Post-Sensing check will identify this error. Finally, if the value is entered in the wrong box, say the Password, Post-Sensing would identify that the Username box is still blank, flagging an error. By defining the micro-task in this way, we can also add significantly deeper logging (based upon the Pre-Sensing, action, and Post-Sensing) as well as graceful error handling (e.g., for any of the error cases identified above). While significantly more overhead than an agent tasked with higher-level commands only, our LEASH framework keeps the agent on much tighter rails (as we will show), which helps mitigate the potential for catastrophic errors. Note that while the sensing process itself is prone to errors, we argue that the smaller and the more precise micro-tasks are defined, and therefore the more sensing checks are performed, the less likely it is that an error is impactful, or goes uncaptured.

Experiments

To test this approach we built a LEASH agent that performs the JAIR pre-review criteria checks, on the live JAIR site. The agent first logs into the site to get any new submissions. Then, for each new submission, it performs the 4 checks outlined above. It renders its judgment to a Google Sheet, along with its reasoning, where a person then confirms the agent's assessment and takes the appropriate next step (e.g., summarily reject and email the authors what to change, or move the paper to get reviewed).

The LEASH agent for JAIR is defined in Appendix A, with pre-action sensing, actions, and post-sensing actions for each step (micro-task). Again, note that the sensing and actions are defined as agentic micro-tasks, rather than relying on brittle programming elements such as XPath, which are

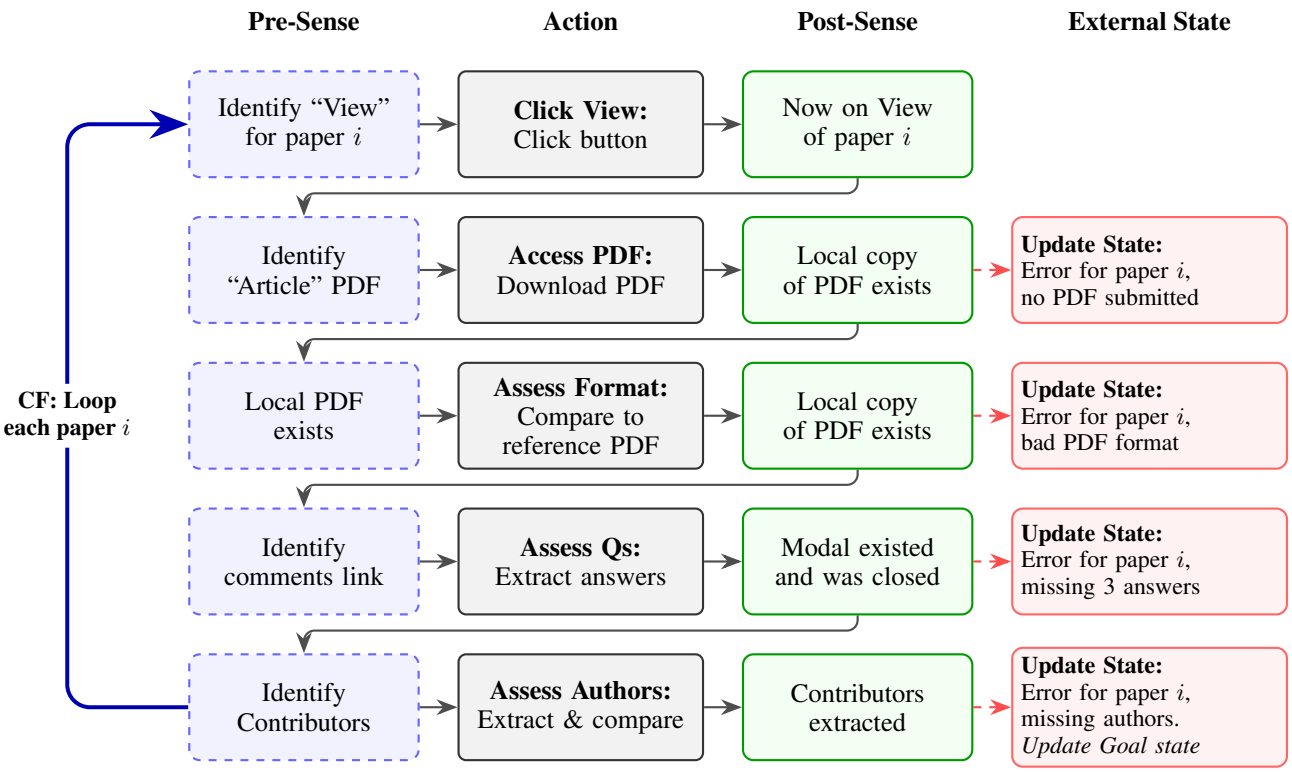

Figure 2: Example of JAIR sub-agent assessing submitted papers. The flowchart incorporates a control flow (CF) loop to process each paper i, generating external state updates (red) when errors are detected in the post-sensing phases.

subject to change by site designers and would therefore require reprogramming. In other words, agentic micro-tasks are robust.

LEASH uses Stagehand for AI-based website usage. Stagehand converts the HTML of a page into a representation of labels, roles, and text, which they call an Accessibility Tree. Each element of this Tree is linked to its XPath elements. Stagehand then reasons about the Accessibility Tree, rather than the raw HTML (e.g., asking a language model, "which item is the Username field") and translates that response back to the linked XPath elements. In this way, it can reason about the form of the website but know which actual HTML elements to interact with.

The agent has been running, live, since October 19th, 2025. Between then and November 3rd, the system rendered 91 decisions, which human reviewers then assessed for this publication.[1] The experimental code is written in Python and leverages Google's Gemini (2.5 Flash and Pro) for language model calls, along with Stagehand, as mentioned, for autonomous website navigation and form interaction.

We compared the LEASH approach to a well used, commercial agent system for autonomous web tasks: Perplexity's Comet. However, we were hesitant to allow Comet to run autonomously against the actual JAIR submission site (given that it must login with elevated credentials), and therefore we created a locally-hosted carbon-copy of JAIR to ensure against the "Don't Press the Red Button" problem. We note that we would have also tested with OpenAI's ChatGPT Atlas browser, but that browser was unable to access the localhost domain where our carbon copy JAIR was hosted. To be clear, as it's task-specific, this comparison is meant to be illustrative of LEASH, rather than a general purpose performance benchmark.

The local hosted version of JAIR contained 5 sample papers, each of which represented a failure case (or passing case) of the criteria (e.g., paper ready for review, no PDF, poorly formatted PDF, etc.). For Comet, we ran the browser, with its Web agent enabled, across 3 independent runs (removing cookies across each run), by first pointing it to the login page of the locally-hosted copy of JAIR and providing it with the prompt in Appendix B, after which we recorded the interaction and the results. Of course, we ran the LEASH agent against the locally hosted JAIR as well.

Results

Precision Results

A key component of our agent is the ability for it to perform the main task, namely assessing the four criteria that determine if a submission can be put into the pile for review (or be summarily rejected due to structural issues, such as not being a PDF submission). In Table 1, we present the precision results, based upon manually verifying the LEASH agent's

[1] As of our final submission (February 25th), the system rendered 757 decisions while being continuously improved.

decision for all 91 submissions. It's important to note that a paper may have multiple criteria failures at once, so below we present the results overall and also break it down specifically for each criterion (which is why the total number of errors at the overall paper level is less than the sum of all errors across all criteria). In all cases, we present precision as the number of correct assessments over the total number of assessments ("Total Count.")

In all these cases, the LEASH agent correctly identifies rejection cases with sufficient accuracy to be useful. In two cases in particular: (i) determining that a paper passed all criteria and is ready for review and (ii) determining that the submitted materials failed to include a PDF file, the agent performed with near perfect accuracy. We note that determining if a paper is not a PDF is a simple check, but still one that requires the agent to successfully log into website, identify and navigate to the submission's "detail page" (e.g., the submission page with all of the information about a paper, including the submitted file) and assess the file type.

When we investigated the AI's errors, we found that while it was incorrect, it did have logic to its faults. For the cases where the Authors were incorrect, the main mistake occurred when the agent was unable to parse the PDF (more on that below) and therefore could not determine a suitable author list from the paper itself (to compare to the author list on the submission site). This is the same common error when the agent incorrectly determined that the submitted PDF did not conform to JAIR's formatting. Concretely, the agent often had this type of PDF misparsing error when the paper was a resubmission and the first few pages were an explanation of changes made in the paper, for instance, instead of the expected first page of a submission. We note, for processing purposes associated with cost and time, our LEASH agent only looks at the first page of the PDF to determine the PDF style and the authors list. We considered this constraint reasonable, given that the formatting of JAIR requires that the author list is on the first page (if it is not, then the PDF is not well formatted).

More interesting were the errors when the agent assessed if a submission answered the 3 required pre-questions with a yes-or-no. Despite good-faith effort to write a prompt that explicitly asks the agent not to judge the quality of the answers, this is precisely what it did for many of those mistakes. Here is one example (we always require the agent to justify its decisions):

> The author's comments address the importance of the work (question 1) and the previous submission history (question 3). However, for question 2, while the author lists three previously published JAIR papers, they do not provide the required explanation of how their work differs from them.

Here, the authors answered the question, but the agent didn't believe they provided enough information. The agent is being too judgmental. It is supposed to determine that 3 answers exist, not that they are sufficient. Yet, these are reasonable(ish) errors, in contrast to an agent at the higher level, which can go quite off the rails (see Behavior results below).

The precision results for locally hosted JAIR were meant to be trivial, by comparison. The LEASH agent performed with 100% accuracy in its 3 runs on the local JAIR copy. As for Comet, in only 1 of its 3 runs did it perform with 100% accuracy on the 5 test documents. In one of those failure cases, the agent never completed the task (see below). In the other case, it performed with 60% accuracy, as it consistently failed to identify if the PDFs differed in format from what JAIR requires (due to its inability to load the PDFs correctly in that run). This inconsistency in Comet's success rate makes it hard to compare it to LEASH. Comet's average results could be reported, but this would then not be a direct comparison.

Behavior

Precision and timing results justify the use of AI web agents: the agent performs well in its task, within a reasonable amount of time, therefore helping the JAIR review team alleviate some of the burden from some of the more tedious tasks. However, precision and time savings mean nothing if the agent cannot be trusted (Ribeiro, Singh, and Guestrin 2016). Fundamentally, trust is earned by the agent performing on rails, and gracefully failing when it recognizes when it is not. This is the core tenant, above all, for the LEASH framework, alluded to multiple times above as "Don't Press the Red Button."

We contrast LEASH with the behavior in Comet. As mentioned above, in one of the runs Comet failed to perform the task at all. Clearly, without the guidance of micromanagement, Comet presses the red button. In the most egregious run, (the 2nd time we ran it) the Comet agent failed to perform the task as it got sidetracked trying to access a specific PDF. Once it went down this path it began to perform potentially dangerous actions, consistently. We captured the behavior in a video [2]. During this run, Comet invents and navigates to errant URLs 6 times, resulting in 404 errors (in the video around 0:32, 4:36, 4:48, 5:07, 5:19 and 5:54). It also attempts to hit a random API endpoint as well (around 5:51 in the video). In other words, there are seven occurrences in the run where Comet attempts potentially dangerous behavior. Repeatedly hitting non-existent URLs (or API endpoints) can result in a Denial of Service (generating traffic resulting in server errors). More worrisome, the agent could have interacted with a working URL or API endpoint. Given its privileged access as a reviewer it could have done harm to the journal (say by deleting something or rejecting a paper). The version of Comet that we used was 141.0.7390.55.

Discussion

The goal of the LEASH framework is to strike a balance between the ability of agentic software to perform robust, generalized tasks (such as assessing a JAIR submission's pre-submission criteria) with the control of traditional software. To that end, it's worth discussing two aspects of LEASH agent development and behavior - specifically (1) the challenges when developing agents and (2) the types of errors a LEASH agent will make versus other approaches where the agent acts on a high-level goal.

[2] https://www.youtube.com/watch?v=ZnkmrHR3jo8

	Total Number of Papers	Number of Occurrences where AI Assessment is Incorrect	Precision
Overall	91	9	90.11%
Paper Ready for Review (Passed All Criteria)	33	1	96.97%
Authors Incorrect	17	2	88.24%
Not a PDF Submission	19	0	100.00%
Issue with PDF format	26	4	84.62%
Issue with answering 3 initial questions	16	3	81.25%

Table 1: Precision results of LEASH on the JAIR assessment use cases.

In terms of software development, there is a fundamental difference when developing traditional software versus agents, particularly when considering the "error surface." That is, the distribution of errors is significantly more varied and unexpected with AI web agents versus traditional software that runs "on rails" by definition. Agentic software can behave in unexpected ways and with unexpected frequency, making it much more challenging to test the software (or develop tests in the first place). We've found that it simply takes more iterations and real-runs to expose the agent (and ourselves) to more and more possibilities of behavior. The micro-task approach employed by the LEASH approach is meant to minimize the potential error distribution versus an agent tasked with a higher level description of a task and then set loose. Further, the Pre-Sensing, action and Post-Sensing approach allows for LEASH to gracefully exit at any of those points when unexpected behavior is detected.

The second point of discussion involves the types of errors, in particular contrasting those we generally see with a LEASH approach versus an agent tasked at a higher level. The most common error for our JAIR LEASH agent is that it fails to identify the part of the HTML page where the action will take place. For instance, it periodically fails to identify an input area (such as Username or the Login button). The next most common error is a failure in navigation: the agent doesn't navigate to the correct page, such as the page where the PDF of the submission is stored. In all of these cases, the agent successfully realizes it's in an erroneous state or made an error in identification, and it gracefully exits, waiting for the next run. We contrast this with the errors that Comet makes. As shown in the linked video, Comet makes a series of pretty significant errors including trying to access illegal URLs on the carbon-copy JAIR (resulting in 404 errors), making inappropriate API calls, and getting stuck in navigation loops. In particular, the 404 error pages and the mysterious API calls are the most distressing, as these could very well have been the Comet browser "pressing the red button." It's not safe, on the server side, to have an agent that will invent web pages or submit to APIs, especially if it's given privileged access to see all of the details of a potential submission.

LEASH's structured sensing mechanisms, specifically the pre-sensing validation steps, inherently reduce the likelihood of "pressing the red button," or committing unsafe side effects while pursuing a primary objective. However, empirically quantifying this safety advantage remains a significant challenge for two primary reasons. First, the volume of micro-interactions required for modern web navigation creates a verification bottleneck. A single task may decompose into hundreds of interactions; verifying that none of these constituent actions triggered an unintended harmful state requires a level of granular monitoring that is hard to set up. Proving the absence of a negative event is therefore more demanding than proving the presence of a positive success.

Second, the binary "red button" metaphor oversimplifies the spectrum of real-world web risks. In practice, dangerous actions are rarely distinct, monochromatic hazards but rather exist on a gradient of severity. Distinguishing between a benign unnecessary click (e.g., expanding a harmless dropdown) and a critical failure (e.g., submitting a form with incorrect data) is subjective and context-dependent. This ambiguity complicates the establishment of a ground-truth metric: it is unclear whether all unrequested actions should be penalized equally or weighted by potential impact. Consequently, while LEASH's architecture provides a mechanism to gate actions via sensing, developing a standardized benchmark to rigorously evaluate "safety from side effects" remains an open problem for future research.

Future Work

The control of autonomous agents is both a nascent field and one that can build upon the significant body of research in planning, verification, and human-robot interaction. As web agents become increasingly capable, the challenge shifts from enabling basic functionality to ensuring that agents operate reliably, predictably, and safely within complex environments. The directions outlined below reflect opportunities to strengthen the LEASH framework along these dimensions.

A persistent challenge in LLM-based systems is prompt optimization. As observed in our Results section, LEASH's agent exhibited overly strict judgment, sometimes concluding that authors had not adequately answered form questions when, upon manual review, they arguably had. Calibrating the agent to align with human expectations constitutes a prompt optimization problem operating within an effectively unbounded search space, so one cannot definitively claim that no prompt exists that would resolve this issue. The only tractable approach is to invest a reasonable amount of good-faith effort and report outcomes relative to that effort. A natural direction for future work is to dedicate focused attention to systematically optimizing all prompts, both within

LEASH and Perplexity Comet, thereby enabling a comparison of these systems operating at their respective best configurations.

A second avenue for improvement involves modifying how LEASH interacts with web pages. Rather than controlling the browser directly through cursor movements and click actions, the agent could instead generate executable code that performs the desired browser interactions. This approach would combine the ability of web agents to interpret page structure and determine appropriate actions with the precision and controllability afforded by code. Additionally, code-based interaction opens the possibility of writing automated tests to verify correct behavior, offering stronger guarantees than are achievable through visual inspection of dynamic web content alone.

Additionally, future work could explore adaptive recovery mechanisms when sensing operations fail. In the current LEASH architecture, micro-tasks guide the web agent's execution, while pre-sensing verifies task feasibility and post-sensing confirms successful completion. When either sensing step fails, the system currently exits gracefully to avoid proceeding on an incorrect basis. However, a more sophisticated approach would leverage failure information to automatically refine the task or sensing specification. For example, if a micro-task instructs the agent to enter a username during registration, and the post-sensing detects an error message indicating that certain characters are disallowed, the system could automatically update the micro-task description to encode these constraints. Such self-correcting behavior would improve robustness and reduce the need for manual intervention when encountering unexpected interface requirements.

A fourth direction concerns the incorporation of explicit constraints into the LEASH framework. Currently, micro-tasks provide only positive instructions - specifications of what the agent should do - without corresponding prohibitions on undesirable actions. Augmenting the framework with constraint specifications would allow task designers to articulate not only the intended behavior but also the boundaries the agent must respect during execution. Under this extended model, pre-sensing would verify that the micro-task is achievable without violating the stated constraints, while post-sensing would confirm both successful task completion and constraint adherence. This formulation offers a principled mechanism for operationalizing the "Don't Press the Red Button" principle: rather than relying on implicit assumptions about safe behavior, constraints would explicitly define what constitutes a red button in each operational context. Such declarative safety boundaries would make the system's behavioral guarantees more transparent and verifiable.

Related Work

The rapid advancement of large language models has catalyzed a new generation of autonomous web agents capable of navigating and interacting with real-world websites. Among seminal work using LLMs is WebVoyager, which introduced a multi-modal agent that combines screenshot analysis with text extraction to complete end-to-end web tasks, achieving a 59.1% success rate on a benchmark spanning 15 popular websites (He et al. 2024). Subsequent systems have pushed performance further: Agent-E employs a hierarchical architecture separating high-level planning from low-level navigation, reaching 73.2% on WebVoyager's benchmark (Abuelsaad et al. 2024), while the Browser-Use framework reports 89.1% success on the same benchmark through changes made to WebVoyager code base (Browser Use Team 2024). Commercial deployments have followed, with Anthropic's Claude for Chrome (Anthropic 2025), OpenAI's Operator (OpenAI 2025b), OpenAI's ChatGPT Atlas (OpenAI 2025a), and Perplexity's Comet browser (Perplexity 2025), offering users autonomous web assistance. Many modern web agents employ iterative reasoning-action loops, an approach exemplified by the ReAct paradigm (Yao et al. 2023).

Despite these advances, controlling the behavior of autonomous web agents remains a significant challenge. As demonstrated by recent production deployments, agents operating without structured guidance can exhibit unpredictable and potentially harmful behavior: inventing URLs, triggering server errors, or attempting unauthorized API calls (see also the security analysis of Vardanyan (2025)). The ST-WebAgentBench benchmark explicitly targets this gap, evaluating agents not only on task completion but on safety and trustworthiness criteria (Levy et al. 2025). More fundamentally, the explainable AI (XAI) literature, such as Ribeiro, Singh, and Guestrin (2016)'s work, established that users cannot rely on accuracy metrics alone - they require interpretable explanations of model behavior to make informed decisions about when to trust predictions. This principle extends naturally to autonomous agents: trust requires not only successful task completion but predictable, bounded behavior that humans can verify and understand.

The challenge of ensuring safe, constrained behavior in autonomous systems has a rich history in planning and robotics. Classical AI planning provides formal foundations for specifying tasks, reasoning about action preconditions and effects, and computing action sequences that achieve goals while respecting constraints (Ghallab, Nau, and Traverso 2016). Plan-constraint and preference extensions in PDDL3 enabled planners to avoid trajectories violating safety or ethical considerations (Gerevini and Long 2006), while temporal logic on finite traces such as LTLf/LDLf demonstrated how rich constraints can be compiled into planning and synthesis workflows (De Giacomo, Vardi et al. 2013). These pre-execution approaches illustrate how behavioral restrictions can be enforced ahead of time to guarantee safe plans. Complementing such pre-sensing control, a substantial body of work addresses supervision during execution. Temporal network frameworks allow agents to dispatch plans while checking consistency and responding to deviations or unexpected events (Morris and Muscettola 2000).

More recently, researchers have moved beyond post-hoc monitoring to active runtime enforcement. In reinforcement learning, shielding has emerged as a key technique: a reactive system monitors the learner's proposed actions and suppresses or substitutes those deemed unsafe (Alshiekh et al. 2018). Online variants compute safety decisions at runtime

rather than offline, increasing scalability to complex domains (Könighofer et al. 2023), and extensions to multi-agent systems address decentralized shielding in partially observable settings (Melcer, Amato, and Tripakis 2024). Robotics research has converged on closed-loop safety architectures embodying a "sense → decide → act → verify" paradigm that unifies classical control with modern runtime verification. Early introspection frameworks enabled robots to analyze sensory streams and verify that actions achieved expected outcomes (see, e.g., Rojas et al. (2016)), while verification-in-the-loop approaches formalized this feedback cycle for control policies with reach-avoid constraints (Wang, Papachristodoulou, and Margellos 2025). Safety-critical architectures employing layered tactile sensing physically embody pre-action safety checks and post-action force verification, and learning-based mechanisms such as shields for safe reinforcement learning enforce both pre-action checks and post-action monitoring to maintain formal safety properties throughout execution (Könighofer et al. 2025).

What remains missing is a synthesis of these two research traditions: the perceptual intelligence and flexibility of LLM-powered web agents combined with the principled control guarantees of planning and robotics frameworks. Current web agents operate with minimal structural constraints, relying on the LLM's implicit reasoning to avoid harmful actions - an approach that, as our results and others demonstrate, proves insufficient for safety-critical deployments. Conversely, classical planning and shielding techniques assume well-defined action spaces and state representations that do not readily accommodate the unstructured, visually complex environments of real-world websites. The LEASH framework presented in this paper addresses precisely this gap, introducing a layered control architecture that applies pre-sensing verification before micro-task execution and post-sensing confirmation afterward, enabling web agents to operate with the behavioral guarantees previously available only in more constrained robotic and planning domains.

Conclusion

This paper introduced LEASH, a framework for AI web agents that prioritizes controllability and graceful failure over unconstrained autonomy. Through a structured decomposition of tasks into micro-tasks with pre-sensing and post-sensing mechanisms, LEASH ensures that agents operate within well-defined boundaries and halt execution when those boundaries are violated, a principle we term "Don't Press the Red Button."

Our evaluation on the JAIR submission review task demonstrates that LEASH achieves strong precision (90.11% overall) while maintaining predictable behavior. Where errors occurred, they followed interpretable patterns: PDF parsing failures for non-standard submissions and overly strict judgment when evaluating author responses to required questions. Critically, these errors were bounded and comprehensible rather than erratic.

The contrast with Perplexity Comet underscores the value of the LEASH approach. While Comet achieved comparable precision in successful runs, its behavior proved inconsistent and, in one instance, potentially dangerous: inventing URLs, triggering 404 errors, and attempting to access API endpoints without guidance. Such unpredictable actions, particularly from an agent with privileged access, represent unacceptable risks in production environments. Trust in AI agents cannot be established through average-case performance alone; it requires guarantees about worst-case behavior.

We believe the LEASH framework offers a path toward deploying AI web agents in sensitive contexts where reliability matters more than flexibility. Our approach should generalize beyond the example JAIR use-case, to other enterprise workflows where providing privileged access to an agent could lead to harmful mistakes (e.g., compliance checks, audits, etc.). Future work will focus on prompt optimization to further improve precision, code-based browser control for enhanced verifiability, and adaptive recovery mechanisms that allow the system to learn from sensing failures without compromising safety. As AI web agents become more capable, frameworks that constrain their autonomy while preserving their utility will be essential for their responsible adoption.

References

Abuelsaad, T.; Akkil, D.; Dey, P.; Jagmohan, A.; Vempaty, A.; and Kokku, R. 2024. Agent-E: From Autonomous Web Navigation to Foundational Design Principles in Agentic Systems. In *NeurIPS Workshop on Open-World Agents*.

Albus, J. S. 2002. Outline for a Theory of Intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3): 473–509.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe Reinforcement Learning via Shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2669–2678.

Anthropic. 2025. Getting Started with Claude for Chrome. https://support.claude.com/en/articles/12012173-getting-started-with-claude-for-chrome.

Backus, J. W. 1959. The Syntax and the Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. In *Proceedings of the International Conference on Information Processing*, 125–132.

Backus, J. W.; Bauer, F. L.; Green, J.; Katz, C.; McCarthy, J.; Perlis, A. J.; Rutishauser, H.; Samelson, K.; Vauquois, B.; Wegstein, J. H.; et al. 1960. Report on the Algorithmic Language ALGOL 60. *Communications of the ACM*, 3(5): 299–311.

Browser Use Team. 2024. Browser Use = State of the Art Web Agent. https://browser-use.com/posts/sota-technical-report.

De Giacomo, G.; Vardi, M. Y.; et al. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proceedings of the International Joint Conferences on Artificial Intelligenc*, volume 13, 854–860.

Gerevini, A.; and Long, D. 2006. Preferences and Soft Constraints in PDDL3. In *ICAPS Workshop on Planning with Preferences and Soft Constraints*.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

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

Könighofer, B.; Bloem, R.; Jansen, N.; Junges, S.; and Pranger, S. 2025. Shields for Safe Reinforcement Learning. *Communications of the ACM*, 68(11): 80–90.

Könighofer, B.; Rudolf, J.; Palmisano, A.; Tappler, M.; and Bloem, R. 2023. Online Shielding for Reinforcement Learning. *Innovations in Systems and Software Engineering*, 19(4): 379–394.

Levy, I.; Marreed, S.; Oved, A.; Yaeli, A.; Shlomov, S.; et al. 2025. ST-WebAgentBench: A Benchmark for Evaluating Safety and Trustworthiness in Web Agents. In *ICML Workshop on Computer Use Agents*.

Melcer, D.; Amato, C.; and Tripakis, S. 2024. Shield Decentralization for Safe Reinforcement Learning in General Partially Observable Multi-Agent Environments. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2384–2386.

Morris, P. H.; and Muscettola, N. 2000. Execution of Temporal Plans with Uncertainty. In *Proceedings of the National Conference on Artificial Intelligence (AAAI) and Conference on Innovative Applications of Artificial Intelligence (IAAI)*, 491–496.

OpenAI. 2025a. Introducing ChatGPT Atlas. https://openai.com/index/introducing-chatgpt-atlas.

OpenAI. 2025b. Introducing Operator. https://openai.com/index/introducing-operator.

Perplexity. 2025. The Browser that Works for You. https://www.perplexity.ai/comet.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why Should I Trust You?" Explaining the Predictions of any Classifier. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144.

Rojas, J.; Huang, Z.; Luo, S.; Kuang, Y. D. W.; Zhu, D.; and Harada, K. 2016. Robot Introspection via Wrench-based Action Grammars. *arXiv:1609.04947*.

Vardanyan, A. 2025. Building Browser Agents: Architecture, Security, and Practical Solutions. *arXiv:2511.19477*.

Wang, H.; Papachristodoulou, A.; and Margellos, K. 2025. Distributed Safe Control Design and Probabilistic Safety Verification for Multi-Agent Systems. *Automatica*, 179: 112393.

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