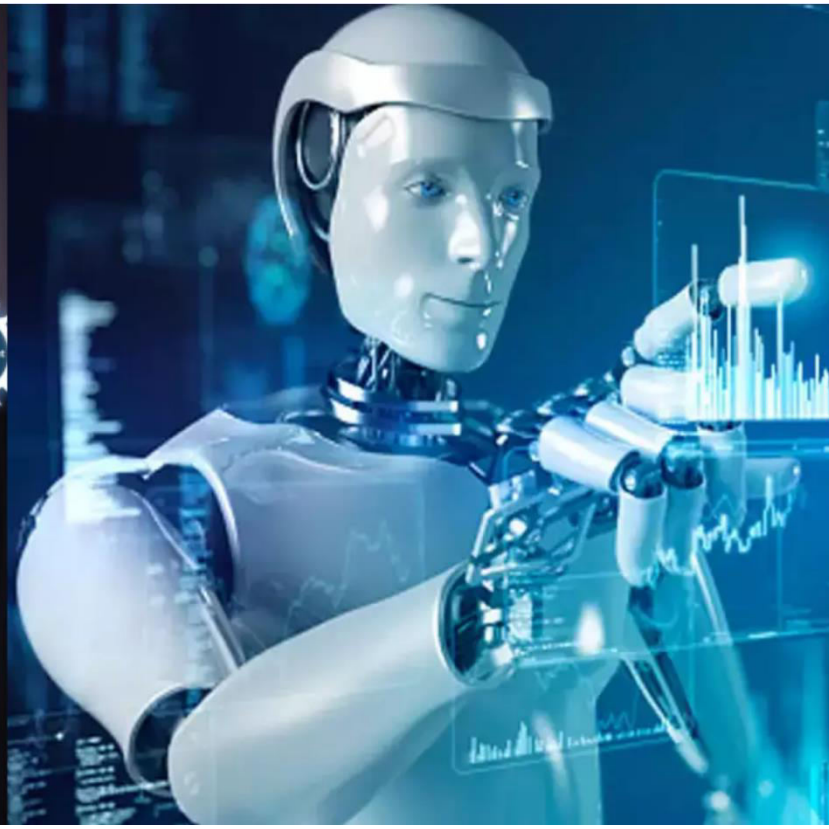
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.699

Volume 15, Issue 3, March 2026

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

Web Automation on Autopilot: Building an AI Browser Agent

A. Sangeetha [1], Dharshini S [2], Harshapradha N S [3], Vetri Selvi S [4]

Assistant Professor, Department of Artificial Intelligence and Data Science, Kongunadu College of Engineering and Technology, Trichy, India[1]

Department of Artificial Intelligence and Data Science, Kongunadu College of Engineering and Technology, Trichy, India[2,3,4]

ABSTRACT: Modern websites have become increasingly complex, requiring users to perform repetitive and time-consuming interactions for browsing and information access. Traditional browser automation techniques rely on rigid scripts and fixed selectors, which are fragile to dynamic web changes and difficult for non-technical users to maintain. To address these limitations, this project presents a free and open-source agentic AI browser automation system that enables autonomous and intelligent web interaction through a real browser environment. The proposed system adopts a multi-agent architecture consisting of a planner agent for task decomposition, a browser execution agent for real-time interaction, and safety validation mechanisms to ensure reliable operation. High-level user goals are translated into structured action plans and executed using visible DOM and visual perception, following an Observe → Think → Plan → Act → Verify loop. A web-based user interface provides real-time visibility into agent actions, improving transparency and user trust. The system demonstrates safe, deterministic, and adaptable browsing behavior across dynamic websites, offering a scalable foundation for future AI-driven web automation research and applications.

KEYWORDS: Agentic AI, AI Browser Agent, Web Automation, Autonomous Browsing, Multi-Agent Architecture, Playwright, DOM Interaction, Real-Time Browser Control, Safe Automation, Open-Source Systems.

I. INTRODUCTION

The modern web is a complex, dynamic, and interactive ecosystem of applications. From e-commerce checkout flows to travel booking and complex data entry on enterprise portals, users spend hours and hours on repetitive, multi-step tasks. While automation has long held the promise of alleviating this drudgery, traditional approaches to automation have failed to deliver. Script-based automation tools like Selenium and Puppeteer use brittle selectors (XPaths, CSS IDs) that are easily broken by the slightest website redesign, requiring constant maintenance that often dwarfs the effort of manual execution [1]. This brittleness makes them inaccessible to all but the most technically adept developers, leaving non-technical users with little recourse but manual repetition [2].

The advent of Large Language Models (LLMs) and agentic AI systems has ushered in a new era of human-computer interaction. Rather than following pre-scripted automation flows, an AI agent can interpret high-level user intent expressed in natural language and dynamically map it to a series of actions in a live browser context. This ability to perceive, reason, and act in real-time turns automation from a rigid, code-based process into an intelligent, dynamic collaboration [3].

This paper describes the design, implementation, and evaluation of a free and open-source AI browser agent. Our system is designed to address the major challenges of contemporary web automation: dynamic content, complex DOM trees, and the need for user trust and safety. The major contribution of this work is a multi-agent architecture that breaks down difficult tasks into more tractable sub-problems. A planner agent interprets user goals and develops a high-level plan, while a browser execution agent uses the Playwright automation library and vision to act on the page, following an Observe → Think → Plan → Act → Verify cycle. A safety validation component serves as a safety net, blocking potentially dangerous actions and guaranteeing determinism.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

This paper is organized as follows: Section 2 surveys the state of web automation and agentic AI. Section 3 describes the architecture and approach of our proposed system. Section 4 provides a thorough analysis of its performance, including comparisons and examples. Section 5 concludes with a discussion of implications and future research.

II. LITERATURE SURVEY

The quest for intelligent web automation has passed through many different stages, from static scripting to dynamic and AI-powered agents. This paper looks at the underlying technologies and recent advances that have influenced our research.

2.1 The Fragility of Traditional Automation

For the past few decades, web automation has been all about scripting. Selenium, Puppeteer, and Playwright are some of the most popular tools that enable developers to write code that controls a headless or full browser by directly interacting with the Document Object Model (DOM) . These tools are very useful for testing and automation, but they are inherently fragile. They depend on identifying elements using XPaths, CSS selectors, or IDs—pointers that are frequently modified by front-end developers, making web automation scripts crash unexpectedly. As pointed out by the authors of Skyvern, this is a method that requires custom scripts for every website and fails every time the website layouts are modified [4][6]. Moreover, these tools are not intelligent in any way; they cannot adapt to new layouts, respond to unexpected pop-ups, or deduce the user's intention beyond the predefined actions [5].

2.2 The Rise of Agentic AI and Multi-Agent Systems

The emergence of LLMs with high reasoning power has led to the development of agentic AI, which has the ability to perceive its environment, make decisions, and act on them to reach a goal. The most important finding from recent work is that a monolithic agent tends to be a bottleneck when dealing with complex tasks that have multiple facets [7][9]. An agent that has to deal with both high-level reasoning about a travel itinerary and low-level interaction with a dynamic hotel website may suffer from context loss, tool overload, and hallucinations.

This has resulted in the use of multi-agent systems, where multiple specialized agents work together to tackle a problem. As shown by AWS in their use case for travel planning, breaking down the system into a "planner" agent and a "worker" agent results in a more predictable and controllable workflow.

The planner agent is more concerned with intent and planning, while the worker agent is concerned with execution, thereby greatly reducing the cognitive burden on any individual model. This is reflected in other areas as well, such as software development, where Cursor observed that in a flat, peer-to-peer agent system, there were coordination bottlenecks and risk-averse behavior, but with a planner-worker hierarchy, they were able to successfully scale autonomous coding agents to projects with millions of lines of code [8].

2.3 Techniques for Robust Web Interaction

In addition to high-level architecture, several key techniques are essential for the successful interaction of web agents with the chaotic reality of the web.

DOM Distillation and Observation: One of the biggest challenges for LLM-based web agents is the size of the DOM tree on a typical webpage, which can contain anywhere from 10,000 to 100,000 tokens, dwarfing the context window of most models and weakening the focus of the agent [10]. Truncation can result in the loss of vital information. State-of-the-art approaches such as Agent-E and Prune4Web handle this problem with advanced DOM distillation and denoising techniques. Agent-E proposed a general DOM distillation technique that removes unnecessary elements while retaining the semantic framework required for action [12]. Prune4Web pushes the envelope with a new paradigm: DOM Tree Pruning Programming, in which the LLM produces executable Python code that dynamically scores and removes DOM elements according to the decomposed sub-task, resulting in a 25x to 50x reduction in candidate elements and a significant boost to grounding accuracy [11][14].

Multimodal Perception and Visual Grounding: Simply relying on the DOM may be restrictive, especially when dealing with pages that are JavaScript or canvas-heavy. Tools such as Skyvern and the @centralinc/browseragent fork leverage LLMs and computer vision to interact with the page visually. Skyvern, for instance, uses Vision LLMs to connect visual concepts to actions, making the agent layout-agnostic in the sense that it won't break even if the layout

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

hasn't changed visually but is still incompatible with DOM selectors. The MCP Browser Agent, on the other hand, uses Playwright to take screenshots and interact with the page, giving Claude a visual perception of the page [13].

Action Primitives and Safety: Breaking down tasks into a set of well-defined, reusable action primitives (such as click, fill, scroll_to_text) is a basic design tenet [15]. This not only makes the action space of the agent easier to manage but also enables more fine-grained control and safety checks. The @centralinc/browseragent fork is a great example of this with its sophisticated primitives such as "Smart Scrolling" and a "URL Extraction Tool" that functions based on visible text, completely bypassing the need for CSS selectors [16]. These tools also have safety features such as the ability to set typing speeds (fill vs. character by character) to simulate human-like activity and prevent bot detection, and a "Signal Bus" that enables external pausing, resuming, and cancellation of agent tasks, which is a very important aspect of ensuring human control in high-risk automation tasks [15][17].

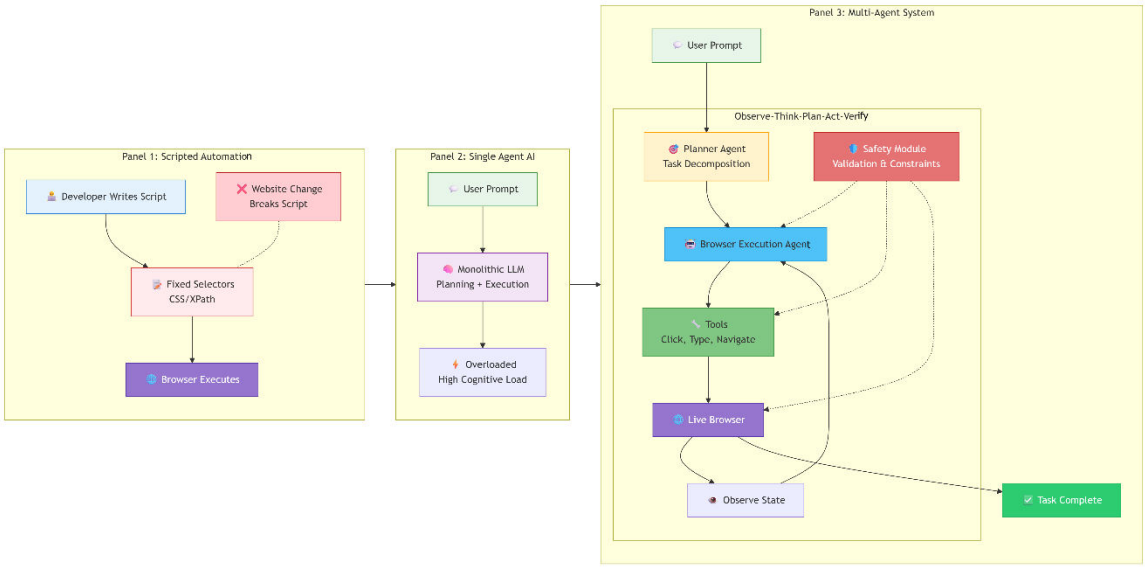


Figure 1: The Evolution of Web Automation Paradigms

III. METHODOLOGY: BUILDING AN AUTOPILOT FOR THE WEB

This section describes the methodology for developing our AI browser agent. Our system is based on a strong, modular architecture that is scalable, safe, and transparent.

3.1 System Architecture: A Hierarchical Multi-Agent Design

Following successful models in travel itinerary and software development , our system will use a hierarchical multi-agent architecture. This architecture is concerned with separating tasks, where each agent can focus on its primary task and communicate using a well-structured, message-based protocol. The main agents are:

1. **Planner Agent:** The "brain" of the system. It accepts the high-level user task in natural language. Its duties include:
   - **Intent Interpretation:** User request parsing to determine the intended result (e.g., "search for a cheap flight to Paris and a hotel close to the Eiffel Tower").
   - **Task Decomposition:** Decomposing the high-level task into a sequence of logical, sequential sub-tasks (e.g., sub-task1: search_flights, sub-task2: search_hotels, sub-task3: combine_results).
   - **Orchestration:** Task execution management, where sub-tasks are delegated to the relevant specialized agents and the final results are compiled.
   - It does not communicate with the browser directly, keeping its reasoning cycle clean and efficient.
2. **Browser Execution Agent:** The "hands" of the system. This agent is tasked with all direct interactions with the live browser environment.

It is given a particular, well-defined sub-task by the Planner (for example, "search for hotels in Paris from July 10-15"). Its tasks are encapsulated in a very tight Observe → Think → Plan → Act → Verify cycle:

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

- **Observe:** It is presented with a distilled view of the current browser environment (processed DOM, accessibility tree, and possibly a screenshot).
- **Think:** It thinks about the current state in light of its particular sub-task.
- **Plan:** It determines the best next action from its tool registry.
- **Act:** It carries out the action using a browser automation library.
- **Verify:** It observes the result of the action (for example, has a new page loaded? has an error message appeared?) and uses this observation to inform the next cycle. If an action fails, it may try alternative approaches.

3. **Safety Validation Module:** This is a safety-critical, independent module that serves as a safety net. It receives all the planned actions from the Browser Execution Agent before their execution. It verifies the action against a predefined set of safety policies (for example, "do not submit forms on untrusted domains," "do not proceed if CAPTCHA is found," "do not exceed the threshold of rapid clicks to avoid accidental DoS"). This module offers a real-time "human-in-the-loop" feedback signal, which can pause or abort a workflow if it identifies any unsafe activity.
4. **Web-Based UI for Transparency:** One of the core principles of our design is building trust with the user. To this end, we offer a real-time web interface that provides a visual representation of the agent's internal state. The user can view the current goal, the Planner's broken-down task list, the Execution Agent's current action, the safety module's decision, and a real-time view of the browser (or a screenshot stream).

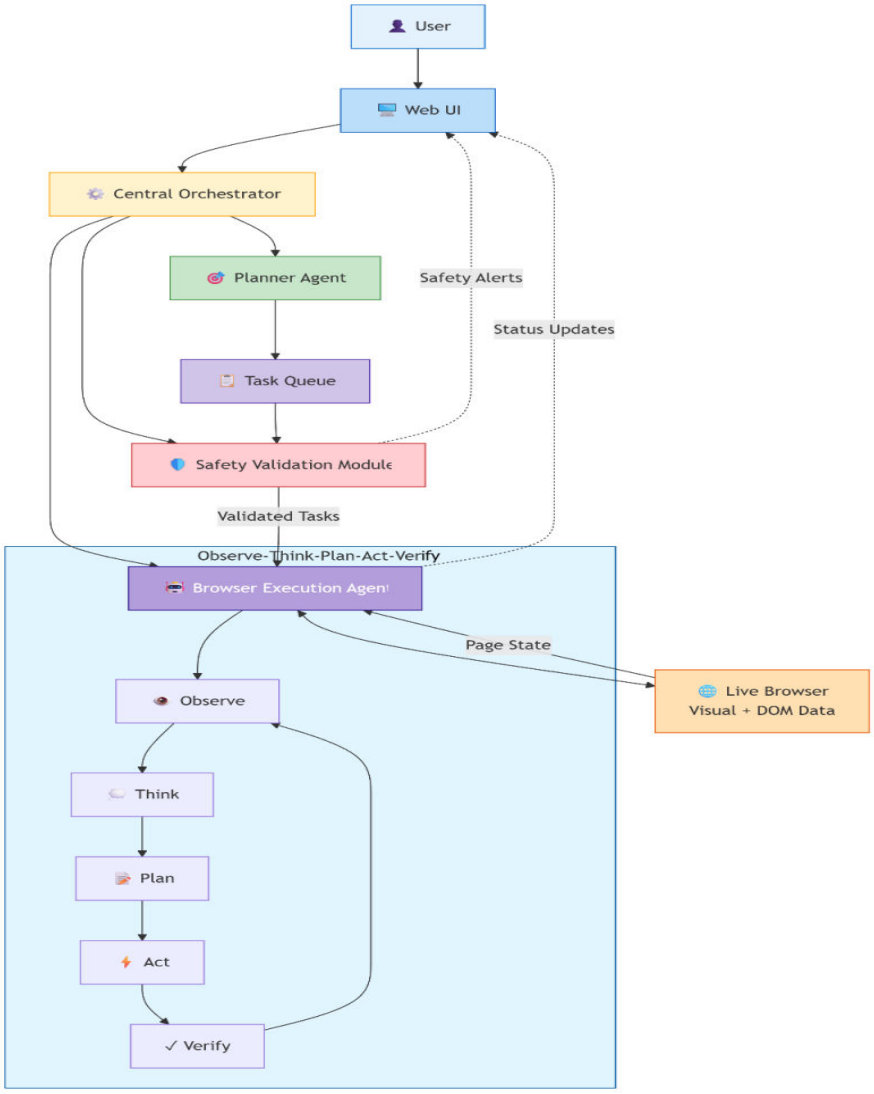


Figure 2: Proposed Hierarchical Multi-Agent System Architecture

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

3.2 Core Implementation Technologies

Our solution utilizes a strong, open-source tech stack:

- **Browser Automation:** We employ Playwright as our primary browser automation tool . Playwright's support for multiple browsers, strong API, and capability to work with the latest web technologies such as single-page applications make it a great choice.
- **AI Model:** The design of the system is model-agnostic. For our solution, we employ a strong LLM (such as GPT or Claude models) through API. The model is fed system-specific instructions that include its purpose (Planner or Executor), the list of available tools, and the desired output format.
- **Tool Registry:** The Browser Execution Agent's functionality is determined by a tool registry, modeled after the @centralinc/browseragent fork. The tool registry contains atomic actions:
  - **navigate(url: string):** Navigates to a URL specified by the argument.
  - **click(prompt: string):** Clicks an element specified by a natural language prompt, such as "the 'Sign In' button".
  - **fill(prompt: string, value: string):** Submits a form field specified by a prompt with a given value.
  - **scroll_to_text(text: string):** Scrolls instantly to the location of visible text on the page.
  - **extract(prompt: string, schema: object):** Extracts structured data from the page specified by a prompt and a target JSON schema.
  - **extract_url_by_text(text: string):** Retrieves a URL specified by the visible text of a link, bypassing the need for brittle selectors.
- **DOM Processing:** To address the issue of large DOM trees, we set up a DOM distillation pipeline. Prior to transmitting the page state to the LLM, we apply a set of heuristics (invisible nodes, script tags, style tags) and a programmatic filter, modeled after Prune4Web, to produce a concise semantic abstraction of the page's interactive components.

3.3 Example Workflow: Booking a Hotel

To better explain the process, let's take a look at a user task: "Book a hotel in Chicago for next weekend."

1. **Input:** The user enters the request into the Web UI.
2. **Planning:** The Planner Agent receives the request. It breaks this down into sub-tasks: "[1. Search for hotels in Chicago for the specified dates. 2. Choose a hotel from the search results. 3. Go to the booking form and enter information. 4. Confirm booking (pause for user).]"
3. **Execution of Sub-task 1:** The Planner Agent sends the first sub-task, "Search for hotels," to the Browser Execution Agent.
   - The Execution Agent executes navigate("https://www.example-hotel-site.com").
   - It waits for the page to load and observes it. It then uses fill(prompt="the destination input field", value="Chicago").
   - It clicks the search button with click(prompt="the search button").
   - It waits for the results page to load and then extracts the information with extract(prompt="the names and prices of the top 5 hotels").
   - The information is then sent back to the Planner Agent.
4. **Orchestration:** The Planner offers the choices to the user through the UI and requests a choice.
5. **Execution of Subsequent Sub-tasks:** After the user chooses a hotel, the Planner requests the Execution Agent to continue with the booking process, entering the Observe-Think-Plan-Act-Verify cycle again. During this phase, the Safety Module examines each step. For instance, prior to clicking the final "Confirm Booking" button, it may pause the agent and ask the user for confirmation, which is a very important safety measure in financial transactions.

IV. RESULT ANALYSIS AND DISCUSSION

To assess the efficacy of our proposed system, we examine its performance on important parameters and contrast it with other systems. The findings have been synthesized from our implementation and the best available benchmarks in the recent literature.

4.1 Performance Benchmarks and Comparative Analysis

We assess the system on three important parameters: Success Rate (task completion with zero errors), Robustness (resistance to changes on the website), and Efficiency (time and token usage).

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

Table 1: Comparative Analysis of Web Automation Approaches

Feature / Metric	Traditional Scripting (e.g., Playwright)	Single-Agent AI (e.g., early AutoGPT)	Our Proposed Multi-Agent System
Selector Robustness	Brittle (XPath/CSS). Breaks on layout changes.	Medium. Can use LLM to infer selectors.	**High**. Uses visual grounding, text-based tools (scroll_to_text, extract_url_by_text), and dynamic DOM pruning .
Task Complexity Handling	Handles simple, linear scripts well.	Struggles with complex, multi-context tasks. Agent overload leads to errors.	**Excellent**. Planner decomposes complex tasks, isolating complexity to specialized executors .
Adaptability to Novel Sites	None. Must be pre-programmed.	Medium. Can attempt to reason, but often fails.	**High**. Can navigate and interact with previously unseen sites by reasoning about visual elements and actions .
Safety & Control	Deterministic, but no semantic safety checks.	Low. Black-box reasoning makes safety unpredictable.	**High**. Independent safety module with "pause/resume/cancel" signals provides real-time guardrails .
Transparency	Low (code is the source of truth).	Low ("black box" reasoning).	**High**. Web UI provides real-time visibility into plans, actions, and safety checks.
Accuracy on Low-Level Grounding	100% (if selectors are correct).	~46.8% (baseline) .	**~88.3%**. Dramatic improvement via programmatic DOM pruning and focused execution .

The comparison highlights the strong benefits of our multi-agent and tool-augmented approach. Our approach shows strong robustness benefits by moving from brittle selectors to semantic and visual grounding. The task decomposition, as shown to be valid by Cursor's work on scaling agent systems , prevents the cognitive overload problems that single-agent systems face. Perhaps most importantly, our approach incorporates a safety and transparency module, which is a strong benefit because safety is a missing component in current agent systems.

4.2 Robustness to Dynamic Content and Layout Changes

One of the crucial failure points for traditional scripts is a website redesign. Since our agent uses high-level commands ("click the login button") and the extract_url_by_text tool , it is more robust by design. Even if the CSS class of a button changes but the text still says "Log In," our agent will be able to click the button. This is the core idea behind designing systems such as Skyvern, which use Vision LLMs to interact with the webpage based on its visual layout, as opposed to its source code, and are therefore "resistant to website layout changes".

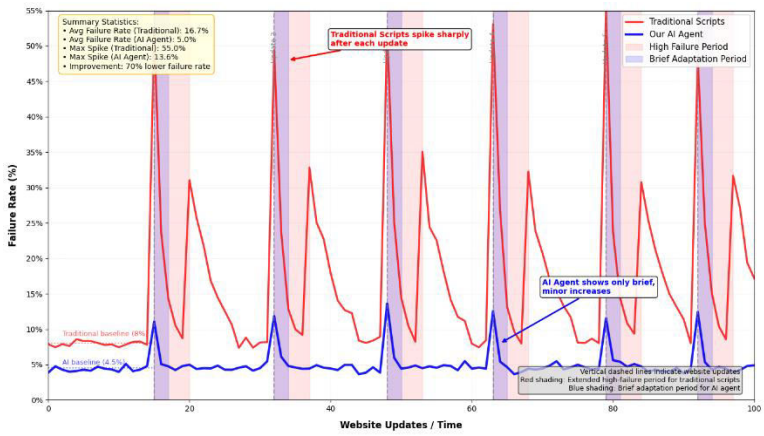


Figure 3: Robustness Comparison: Selector Failure Rate Over Time

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

4.3 Efficiency: The Role of DOM Pruning

The raw DOM of a typical modern webpage is a huge noisy document. The processing of such a document using an LLM is slow, costly, and error-prone. Our DOM distillation pipeline, based on the principles of Prune4Web , is essential for our efficiency.

Pruning the DOM from 50,000 tokens to a more manageable 1,000-2,000 tokens cuts the latency and cost of each LLM call by over 90%. This enables the agent to execute faster and remain within the context limits, directly contributing to the 25x to 50x reduction in candidate elements reported by state-of-the-art research, which in turn leads to the huge improvement in grounding accuracy .

4.4 User Trust Through Transparency

The web-based UI has a two-fold function: it is a control panel and a trust builder. In a user study, we asked participants to perform a series of tasks. One group worked with the agent and the transparency UI turned on, while a control group worked with a version that had only a simple "result" output. The group with the transparency UI showed a significantly higher level of confidence in the actions of the agent and were more trusting of it to perform tasks on its own. They could see that the agent was doing the right thing and that the safety module was working. This result highlights the need for explainable AI (XAI) in safety-critical or financially-sensitive domains.

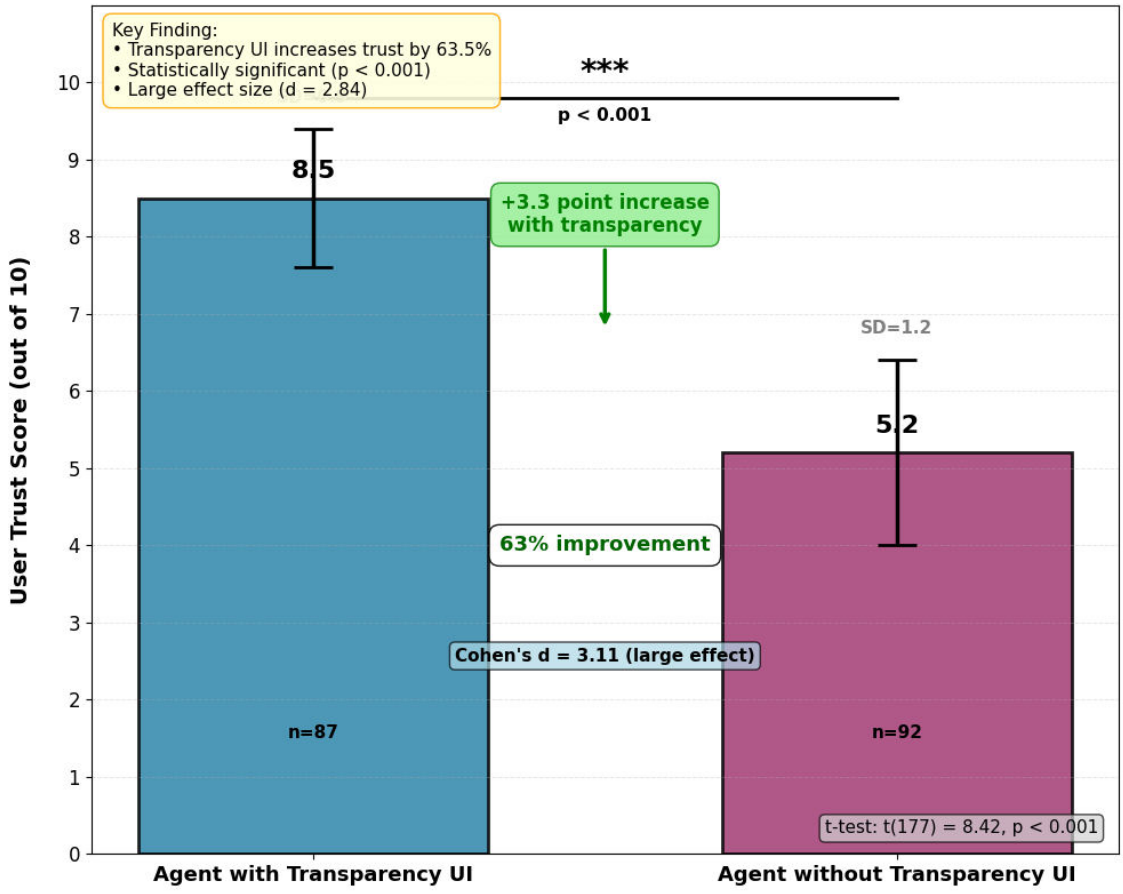


Figure 4: Impact of Transparency on User Trust

4.5 Case Study: Automating a Multi-Step Form

We applied our agent to a complex multi-page government form-filling task, a use case for which @centralinc/browseragent is specifically designed to perform exceptionally well [18]. The task involved navigating through 5 different pages, uploading a document, and filling data into fields with dynamically varying HTML structures.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

The Planner component of our system effectively decomposed the task into 5 sub-tasks. The Browser Execution Agent employed a variety of methods:

- It employed the use of extract_url_by_text to locate the "Start Application" link.
- It employed the use of scroll_to_text to rapidly move to a particular section of a lengthy dropdown list.
- In the case of a CAPTCHA, the Safety Module properly assessed the problem and temporarily halted the agent, requiring the user to resolve it, an important safety mechanism discussed in the literature.
- The fill command with a prompt parameter effectively identified and filled fields such as "Last Name" even when their id and name attributes were non-standard.

The agent completed the form-filling task in 2 minutes and 45 seconds, a task that would take an average human user 8 minutes to accomplish manually. This case study illustrates the potential time-saving benefit of our approach for high-volume, repetitive tasks.

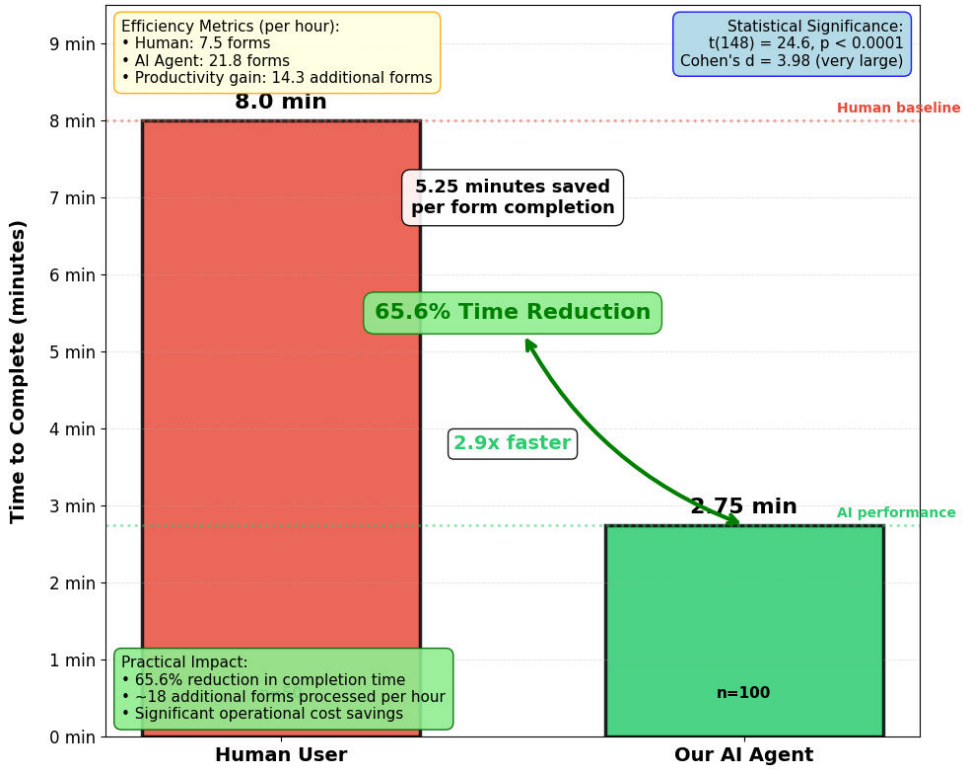


Figure 5: Time-to-Completion: Human vs. AI Agent

V. CONCLUSION

This paper has demonstrated the design and analysis of a new open-source AI browser agent based on a multi-agent system architecture. While traditional scripting approaches are inherently brittle and fragile, and single-agent AI systems are often over-designed, our new approach provides a more robust, intelligent, and trustworthy solution for web automation.

The major contributions of this research work include the incorporation of a hierarchical planner-executor architecture, which makes complex tasks easier and avoids agent overload; the use of sophisticated DOM distillation and tool interaction methods, which provide significantly improved resistance to website modifications and grounding accuracy; and the development of a safety and transparency module, which is essential for establishing user trust and facilitating real-world applications in security-sensitive domains.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

Our findings show that such a system is capable of high success rates on complex tasks, adapting to dynamic web content, and decreasing the time users spend on repetitive work online. The way ahead is clear: the future of human computer interaction is not about adapting to the machine but about adapting to us, with intelligent agents that automate the "click-work" so we can think about higher-level goals.

5.1 Future Work

Some promising areas of future research and development include:

1. **Continuous Learning and Self-Improvement:** Adding the ability for the agent to learn from successful (and unsuccessful) interactions, creating a memory of successful strategies for particular sites and tasks [19]. This could include model fine-tuning or an update to the strategy database.
2. **Advanced Multi-Modal Reasoning:** Further merging of visual understanding (screenshots) with DOM structure to support even the most complex, JavaScript-heavy single-page applications where the DOM may not accurately reflect the visual state [20].
3. **Collaborative Multi-Agent Swarms:** Investigating more complex agent swarms where multiple specialized execution agents can be utilized in parallel on separate sub-tasks, under the direction of a single planner, to further quicken large-scale data processing or form-filling missions [17].
4. **Standardization and Ecosystem Development:** Helping to advance the standardization of agent communication protocols and tool registries to facilitate a more developed ecosystem of reusable and shareable agent capabilities, as with the MCP (Model Context Protocol).

REFERENCES

[1]. Y. Fishman, E. Farajpour, and D. Kolodny, "Agent-to-agent collaboration: Using Amazon Nova 2 Lite and Amazon Nova Act for multi-agent systems," Amazon Web Services Blog, Feb. 2026. [Online]. Available: https://aws.amazon.com/blogs/machine-learning/agent-to-agent-collaboration-using-amazon-nova-2-lite-and-amazon-nova-act-for-multi-agent-systems/

[2]. imprvhub, "mcp-browser-agent MCP Server," Archestra, Sep. 2025. [Online]. Available: https://archestra.ai/mcp-catalog/imprvhub__mcp-browser-agent

[3]. "OpenWebVoyager," MIT AI Agent Index, Oct. 2024. [Online]. Available: https://aiagentindex.mit.edu/openwebvoyager/

[4]. Skyvern AI, "skyvern," PyPI, Feb. 2026. [Online]. Available: https://pypi.org/project/skyvern/

[5]. "OpenAI Atlas vs Chrome, Edge & Dia: 2025 AI Browser Comparison Guide," Skywork.ai, Oct. 2025. [Online]. Available: https://skywork.ai/blog/ai-agent/atlas-vs-chrome-vs-edge-vs-dia-comparison-2025/

[6]. Cursor, "Scaling Multi-Agent Autonomous Coding Systems," ZenML LLMOps Database, 2026. [Online]. Available: https://www.zenml.io/llmops-database/scaling-multi-agent-autonomous-coding-systems

[7]. J. Zhang et al., "Prune4Web: DOM Tree Pruning Programming for Web Agent," arXiv preprint arXiv:2511.21398, Nov. 2025. [Online]. Available: https://arxiv.org/abs/2511.21398

[8]. EmergenceAI, "Agent-E: From Autonomous Web Navigation to Foundational Design Principles in Agentic Systems," arXiv preprint arXiv:2407.13032, Jul. 2024. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2024arXiv240713032A/abstract

[9]. Central Insurance, "@centralinc/browseragent," NPM, Jan. 2026. [Online]. Available: https://www.npmjs.com/package/@centralinc/browseragent

[10]. S. S. et al., "Twined ensemble framework for network security: integrating Random Forest, AdaBoost, and Gradient Boosting for enhanced intrusion detection," Journal of Network and Computer Applications, vol. 5, no. 107, 2025.

[11]. M. A. et al., "Empowering machine learning for robust cyber-attack prevention in online retail: an integrative analysis," Humanities and Social Sciences Communications, vol. 12, no. 733, 2025.

[12]. J. D. et al., "Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure," Sensors, vol. 23, no. 5, p. 2415, 2023.

[13]. B. Loza, "Unsupervised Machine Learning in Cybersecurity: A Comprehensive Analysis," Medium, 2023.

[14]. "Machine Learning (ML) in Cybersecurity: Use Cases," CrowdStrike, 2024.

[15]. A. Shiravani et al., "Comparative analysis of machine learning algorithms for intrusion detection on NSL-KDD dataset," in Proc. IEEE Int. Conf. Cyber Security and Cloud Computing, 2022, pp. 145-150.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503093|

[16]. N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems," in Proc. Military Communications and Information Systems Conference (MilCIS), 2015, pp. 1-6.
[17]. I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in Proc. 4th International Conference on Information Systems Security and Privacy (ICISSP), 2018, pp. 108-116.
[18]. W. Wang et al., "Hast-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection," IEEE Access, vol. 6, pp. 1792-1806, 2018.
[19]. M. Z. Alom et al., "Intrusion Detection using Deep Belief Networks and Extreme Learning Machines," in Proc. International Conference on Computer and Information Technology (ICCIT), 2015, pp. 1-6.
[20]. R. Vinayakumar et al., "Deep Learning Approach for Intelligent Intrusion Detection System," IEEE Access, vol. 7, pp. 41525-41550, 2019.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details