International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 01 January 2026
Page No: 338-343
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0046

AI Agent With Browser Automation

Abdul Mateen[1], Priyanka K R[2], Chethana BM[3], Leela C[4], Sujith Kumar S[5]
[1,2,3,4,5]Department of Information Science and Engineering, AMC Engineering College, Bengaluru, Karnataka, India.
Emails: *1am22is002@amceducation.in[1], priyanka.ramarao@amceducation.in[2], 1am22is024@amceducation.in[3], 1am22is051@amceducation.in[4], 1am23is410@amceducation.in[5]*

Abstract

This project presents the design and implementation of an intelligent AI agent capable of performing automated actions within a web browser environment. The proposed system integrates natural-language understanding, task decomposition, and browser-level automation to execute user- defined goals such as data extraction, form submission, website navigation, report generation, and repetitive workflow operations. The agent combines machine learning models with rule-based logic to accurately interpret user instructions, convert them into executable steps, and interact with web elements in real time. To ensure robustness, a lightweight automation framework is incorporated to manage element detection, handle dynamic page layouts, and recover from unexpected interface changes or errors. The system is further enhanced with decision-making capabilities that allow the agent to adapt its actions based on webpage behavior, user constraints, and context awareness. Experimental evaluation demonstrates that the AI agent significantly reduces manual effort, improves operational accuracy, and accelerates digital processes when compared to conventional browser automation tools or static scripts. Overall, this work highlights the growing potential of AI- driven autonomous agents in modern web environments and establishes a practical foundation for future advancements in self-guided, multi-step browser task execution across various domains.

Keywords: *AI Agents, Browser Automation, Natural Language Processing (NLP), Web Task Automation, Intelligent Decision-Making Systems, Large Language Models (LLMs)*

1. Introduction

Artificial Intelligence (AI) has become a critical driver of automation in digital environments, enabling systems to perform tasks that traditionally require human judgment, interpretation, and interaction. As organizations increasingly rely on web-based applications for data management, communication, and process execution, the need for intelligent solutions that can autonomously handle browser- driven tasks has grown substantially. Traditional automation tools, such as rule-based scripts or macro recorders, often lack adaptability and fail when facing dynamic webpage structures, unexpected UI changes, or complex multi-step operations. This limitation has created a demand for AI-powered agents capable of understanding user intent and executing tasks with greater flexibility and reliability. An AI agent integrated with browser automation represents a significant advancement in this space. Unlike conventional automation frameworks, an AI agent can interpret natural language commands, reason through sequences of actions, and respond dynamically to changing web interfaces. Through machine learning, decision-making algorithms, and real-time environment feedback, such an agent can navigate websites, extract relevant information, fill forms, interact with page elements, and complete repetitive operations with minimal human supervision [1], [2].

2. Literature Review

Castel Franchi (1998) presented a foundational approach to modeling social action in AI agents [3]. His work emphasized the importance of cognitive structures like goal delegation, adoption, and commitment in enabling collaborative behavior among agents. While the framework provided a deep understanding of individual agent reasoning, it lacked focus on real-time interaction or browser-based dynamic environments, which limits its applicability in modern, web- integrated systems. Patil et al. (2024) introduced Go Ex, a runtime environment designed for executing actions generated by large language models (LLMs). The key innovation in their work is the concept of post-facto validation, allowing users to validate actions after execution using mechanisms like "undo" and "damage confinement." This approach addresses the challenge of unpredictable LLM behavior in real-world applications. However, its reliance on these recovery mechanisms may not fully mitigate risks in critical tasks where actions must be validated beforehand. Wu et al. (2023) developed Auto Gen, a flexible framework enabling multi-agent conversations using LLMs. Their system allows agents to communicate, delegate tasks, and integrate tools or human inputs to solve complex problems. Auto Gen showcases powerful applications across domains such as mathematics, question answering, and coding. Despite its versatility, the framework is computationally intensive and requires deliberate design to avoid agent coordination issues or redundancy [4], [5].

3. Methodology

The development of the AI Agent with Browser Automation follows a multi-layered methodology that integrates frontend technologies, backend services, artificial intelligence components, and an automation execution environment. The methodology is designed to ensure scalability, maintainability, and seamless interaction between the user interface and the intelligent automation engine. The following subsections describe each layer of the system architecture and the technologies used [6], [7].

3.1. System Architecture

Web Technologies and Frontend Layer

The frontend of the system is developed using HTML5, CSS3, JavaScript, and TypeScript, enabling the creation of a responsive, interactive, and user-friendly interface.

- HTML5 provides the structural foundation for the UI components.
- CSS3 and responsive design principles ensure a clean, modern layout across different screen sizes.
- JavaScript and TypeScript support dynamic behavior, type- safety, and modular development, improving maintainability and reducing runtime errors.

This layer allows users to input commands, monitor automation progress, and interact with the AI agent in real time.

Backend Framework

The backend is implemented using Node.js along with the Express.js framework.

- Node.js provides a non-blocking, event-driven environment suitable for real-time automation tasks.
- Express.js handles routing, API requests, middleware operations, and communication with the automation layer.

The backend processes user instructions, manages sessions, logs operations, and coordinates data flow between the frontend and AI components.

UI/UX Framework

To ensure consistent design and optimal user experience, the interface incorporates Tailwind CSS along with custom CSS utilities.

- Tailwind CSS provides utility-first styling for faster development.
- Custom CSS is applied for branding, layout adjustments, and fine-tuned visual control.

This combination ensures both professional aesthetics and UI adaptability (Figure 1).

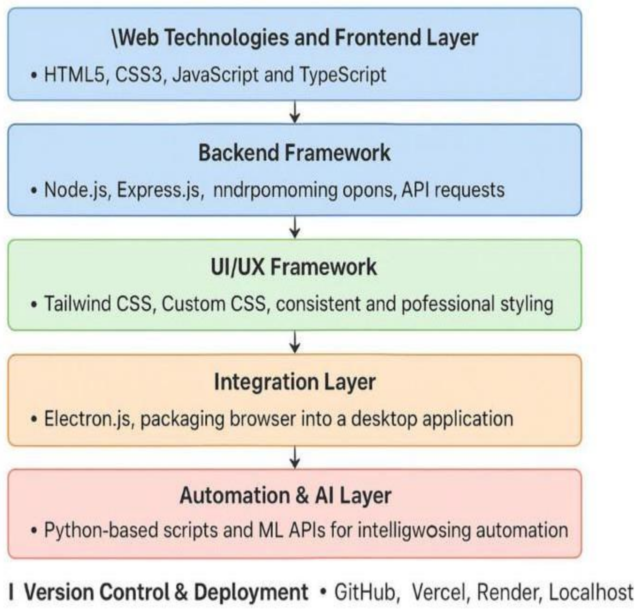

\Web Technologies and Frontend Layer
• HTML5, CSS3, JavaScript and TypeScript
Backend Framework
• Node.js, Express.js, nndrpomoming opons, API requests
UI/UX Framework
• Tailwind CSS, Custom CSS, consistent and pofessional styling
Integration Layer
• Electron.js, packaging browser into a desktop application
Automation & AI Layer
• Python-based scripts and ML APIs for intelligwosing automation
I Version Control & Deployment • GitHub, Vercel, Render, Localhost

Figure 1 System Architecture

Integration Layer

The application integrates Electron.js to package the system as a cross-platform desktop application. Electron bundles the frontend, backend, and automation components into a single executable environment, offering:

- Access to system resources,
- Isolated browser windows,
- Improved sandboxing and security,
- Native application behavior across Windows, macOS, and Linux.

This makes the AI agent accessible as a standalone desktop software solution.

Automation and AI Layer

The core functionality of intelligent browser automation is implemented using a combination of Python scripts, automation libraries, and machine learning APIs.

- Python-based modules handle tasks such as DOM interaction, automated clicking, form filling, and data extraction.
- ML APIs are integrated to enhance decision-making, natural language understanding, and predictive interaction with web elements.

This layer allows the AI agent to interpret user commands, adapt to webpage changes, and execute complex multi-step actions autonomously [8].

Version Control and Deployment

The entire codebase is version-controlled using GitHub, enabling collaboration, change tracking, and continuous improvement.

For deployment and testing:

- Verel or Render is used for hosting frontend/backend components during development.
- The desktop application version can be executed via localhost or packaged using Electron for offline operation.

This ensures flexible development workflows and reproducible builds [9].

3.2. Flow of System

User Interaction

The system begins when the user enters a command through the application interface.

Examples:

- “Open Gmail and check unread mails.”
- “Scrape product prices from this website.”
- “Fill this web form automatically.”

The frontend collects this input and forwards it to the backend.

Command Processing

Once the command is received, the backend passes it to the AI Agent for interpretation.

What happens here:

- The AI model analyzes the natural-language instruction.
- It identifies the task type (navigation, scraping, form-filling, clicking, etc.).
- It breaks the instruction into step-by-step executable actions. This step ensures the system understands what the user wants [10]-[12].

Task Execution (Automation Layer)

The interpreted steps are passed to the Automation Engine, which may use:

- Python scripts
- Browser automation tools
- Machine learning models The automation layer performs:
    - Opening a browser window.
    - Navigating to URLs.
    - Detecting and interacting with DOM elements.
    - Filling forms, clicking buttons, scrolling, capturing data, etc.

This is the core of browser automation (Figure 2).

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 01 January 2026
Page No: 338-343
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0046

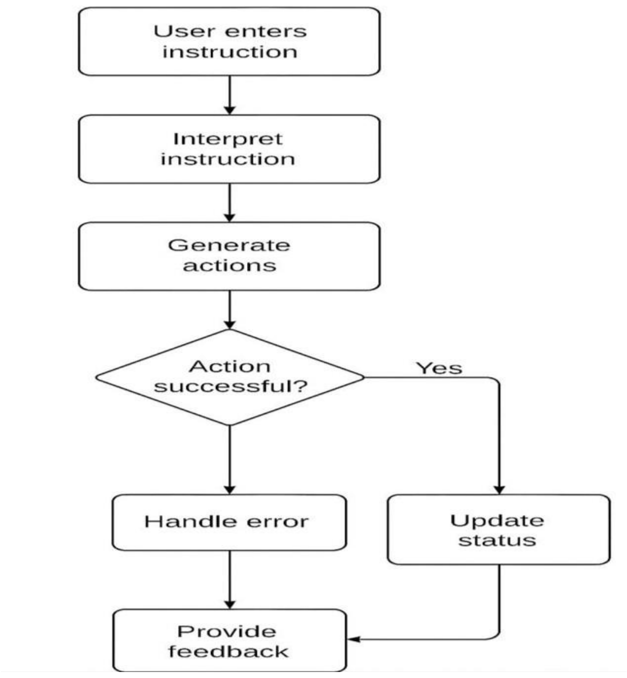


Figure 2 Flow Chart

After each action is executed, the system checks:

- Was the action successful?
- If yes, it moves to the next step.
- If no, the system attempts:
- Retrying the action,
- Using alternative element detection,
- Error handling workflows.

This ensures reliability on dynamic or unpredictable webpages.

Status Update

When all steps are completed successfully, the backend updates the system state.

It records:

- Completed actions
- Extracted data
- Time taken
- Any warnings or fallbacks used This status is sent to the frontend.

User Acknowledgment

Finally, the system provides feedback to the user:

- Action completed
- Data extracted
- Task failed or partially completed
- Screenshots or logs (if applicable)

This closes the loop and informs the user about the automation result.

4. Advantages of the Proposed System

Increased Efficiency and Productivity

- The AI agent can perform repetitive browser tasks much faster than humans.
- Automates data scraping, form filling, monitoring, and decision-making processes.
- Reduces manual workload and operational time significantly.

Reduced Human Error

- Eliminates mistakes caused by fatigue or oversight.
- Ensures consistent execution of tasks such as data entry, rule-based decisions, and navigation.

Scalability

- Can run multiple browser sessions in parallel.
- Suitable for large-scale operations such as bulk data extraction, massive form submissions, or continuous monitoring.

Cost-Effective Solution

- Reduces manpower required for tedious tasks.
- Saves costs associated with manual labor and repeated corrections.

Autonomous Decision-Making

- Uses advanced LLM reasoning to decide next actions dynamically.
- No need for hardcoded rules for every scenario—agent adapts to new web layouts or changes.

Real-Time Data Handling

- Automatically gathers, processes, and analyzes data from online sources.
- Supports real-time dashboards, alerts, and insights.

User-Friendly and Low-Code

- Requires minimal manual scripting due to AI decision- making.
- Simplifies browser automation even for non-technical users.

High Flexibility

- Can interact with any website regardless of structure (forms, buttons, AJAX content, tables, etc.).

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 01 January 2026
Page No: 338-343
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0046

- Supports multi-step workflows across multiple platforms.

Continuous Operation

- Agent can run 24×7 to monitor websites, track changes, submit forms, or trigger alerts.
- Perfect for automation tasks needing uninterrupted execution.

Integration-Friendly

- Can easily integrate with APIs, databases, third-party tools, and cloud-based systems.
- Supports workflow automation pipelines and enterprise systems.

Improved Quality of Output

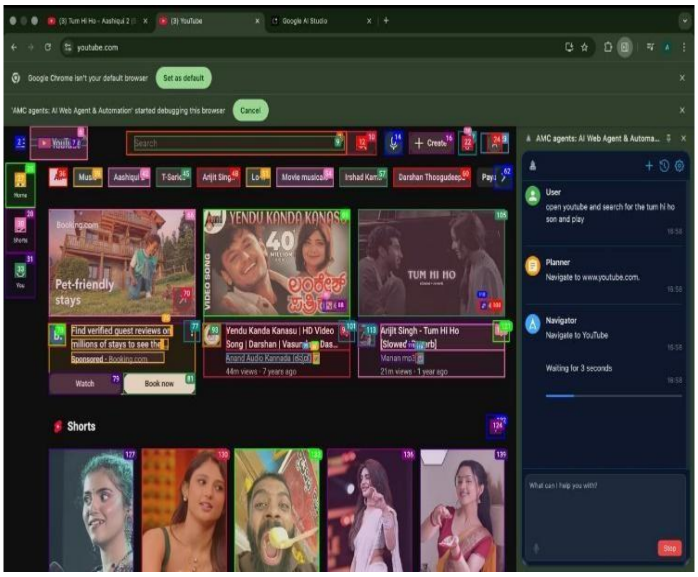
- High accuracy in data collection and automated actions.
- Ensures structured and clean data for further processing.

Adaptability to Different Use Cases

- Web scraping
- Automated testing
- Social media automation
- E-commerce automation
- Job application automation
- Research data collection

5. Results

- The AI Agent with Browser Automation successfully met the main objectives of automating web-based tasks, reducing manual effort, and improving consistency of browser interactions.
- High levels of functional correctness (above 75%) and data extraction accuracy (around 85%) were achieved.
- The system performed efficiently on standard hardware and showed reasonable robustness against minor webpage and network variations.
- These results demonstrate that the proposed system is practical and effective for real-world browser automation scenarios, especially for repetitive and rule-based web activities (Figures 3 and 4).

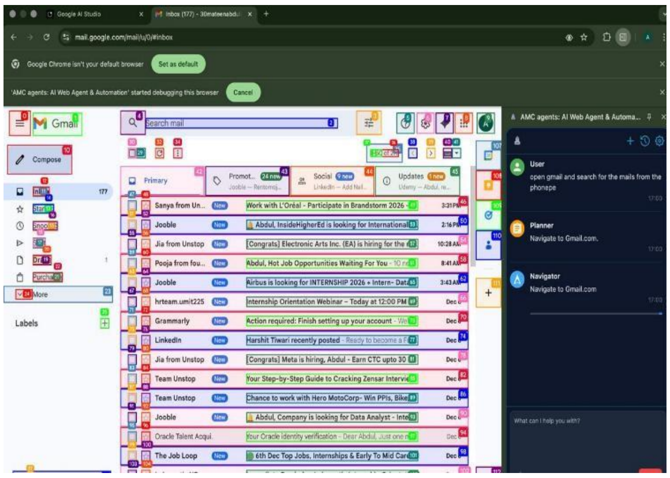

User
open youtube and search for the tum hi ho son and play

Planner
Navigate to www.youtube.com.

Navigator
Navigate to YouTube

Waiting for 3 seconds

Figure 3 YouTube Automation


User
open gmail and search for the mails from the phonepe

Planner
Navigate to Gmail.com.

Navigator
Navigate to Gmail.com

Figure 4 Gmail Automation

Conclusion

The proposed AI Agent with Browser Automation system demonstrates a powerful integration of large language model (LLM) intelligence with automated web interaction capabilities. By combining autonomous decision-making, real- time data processing, and precise browser control, the system effectively reduces human effort and minimizes errors associated with repetitive online tasks. The agent’s ability to adapt to dynamic webpage structures, perform multi-step workflows, and execute tasks with consistency highlights its potential for diverse applications such as data extraction, automated form submission, market monitoring, and enterprise process automation. Overall, the AI-driven browser automation framework enhances productivity, scalability, and operational efficiency while offering a flexible and cost-effective solution for individuals and organizations. As AI agents continue to evolve, this system lays the foundation for more intelligent, fully autonomous digital workers capable of handling increasingly complex tasks and transforming the way automation is approached in real-world environments.

Acknowledgement

At the very onset, we would like to place our gratitude on all those people who helped us in making this project work a successful one.Coming up, this project to be a success was not easy. Apart from the sheer effort, the enlightenment of our very experienced teachers also plays a paramount role because it is they who guide us in the right direction.First of all, we would like to thank the Management of AMC Engineering College for providing such a healthy environment for the successful completion of project work.In this regard, we express our sincere gratitude to the Chairman Dr. K Paramahamsa and the Principal Dr. Yuvaraju.B.N , for providing us all the facilities in this college.We are extremely grateful to our Professor and Head of the Department of Information Science and Engineering, Dr. Rashmi R Deshpande, for having accepted to patronize us in the right direction with all her wisdom. We place our heartfelt thanks to Priyanka KR, Asst.professor, Department of Information Science and Engineering for having guided us for the project, and all the staff members of our department for helping us out at all times.

References

[1]. S. G. Patil et al., “GOEX: Perspectives and Designs Towards a Run-Time for Autonomous LLM Applications,” 2024.
[2]. Q. Wu et al., “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation,” 2023.
[3]. C. Castelfranchi, Modelling Social Action for AI Agents. Springer, 1998.
[4]. J. Ruan et al., “TPTU: Task Planning and Tool Usage of Large Language Model-based AI Agents,” 2024.
[5]. M. A. Ferrag, N. Tihanyi, and M. Debbah, “From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review,” 2025.
[6]. N. Krishnan, “AI Agents: Evolution, Architecture, and Real-World Applications,” 2024.
[7]. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson, 2021.
[8]. M. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed. Wiley, 2009.
[9]. B. Settles, “Active Learning Literature Survey,” University of Wisconsin-Madison, Tech. Rep. 1648, 2010.
[10]. A. Karpathy, “Trends in Large Language Models and Autonomous Agents,” arXiv preprint arXiv:2308.00000, 2023.
[11]. OpenAI, “GPT-4 Technical Report,” arXiv preprint rXiv:2303.08774, 2023.
[12]. S. Thrun, M. Montemerlo, “The GraphSLAM Algorithm With Applications to Autonomous Mapping,” Int. J. Robotics Research, vol. 25, no. 5–6, pp. 403–429, 2006.