

A Large Language Models Based Autonomous Browser Agent

Byagari Ashritha¹, Macha Mahalaxmi², Gugulothu Anjali³

^{1,2,3} Dept. of CSE, G. Narayanamma Institute of Technology & Science (for women)

byagariashritha@gmail.com¹, mahalaxmimacha14@gmail.com², gugulothanjali6@gmail.com³

Abstract— Doing repetitive web-based tasks like searching for information, filling out online forms, and getting data by hand takes a lot of time and is easy to make mistakes. Rule-based automation systems that are based on rules often break when the structure of a website changes or when they have to deal with dynamic web environments. Due to the quick progress of Large Language Models (LLMs), smart systems can now understand instructions in natural language, reason, and make decisions on their own. This project introduces an LLM-Based Autonomous Browser Agent engineered to automate web-based tasks, including navigation, form completion, and data extraction, in a dynamic and adaptive manner. The suggested system combines a Large Language Model with a browser automation framework like Playwright to make an intelligent reasoning-action loop. The agent reads the user's instructions, makes a plan for a series of actions, carries them out in the browser, sees what happens, and then changes its plan on the fly to get the desired result. The system uses smart task decomposition, adaptive decision-making, and strong error-handling to make it more reliable and faster. These features help the agent handle complicated web interactions quickly, cut down on unnecessary actions, and make sure tasks are done correctly. The proposed solution is more flexible, scalable, and robust than traditional automation tools. The system that was developed can be used in real life for helping students, collecting data, automated testing, and automating administrative tasks on the web. This project shows how LLM-powered autonomous agents could change how people interact with the web by making automation smarter, more flexible, and easier to use.

Index Terms— Large Language Models, Autonomous Web Agents, Browser Automation, ReAct (Reasoning and Acting), Playwright, Natural Language Understanding, Multilingual Processing, Speech Recognition, Intelligent Automation, Human-Computer Interaction.

I. INTRODUCTION

The internet has grown so quickly that it has changed how people get services like education, jobs, communication, and information retrieval. People are using web browsers more and more to do things like look for scholarships, apply for jobs, fill out online forms, and use web apps. But these tasks often need the same manual effort over and over again, constant user attention, and moving around on several web

pages, which makes them take a long time and be easy to mess up.

Selenium WebDriver and Playwright are two examples of traditional browser automation tools that let you control web browsers through code. However, they only work with pre-written scripts and fixed workflows. These systems aren't flexible and often break when the structure of a website changes

or when they come across dynamic content. Also, they need programming skills, which makes them hard for people who aren't tech-savvy to use.

Recent developments in Artificial Intelligence, especially in Large Language Models (LLMs) like GPT-4 and Gemini, have made it possible for computers to understand, reason, and make decisions in natural language. These models can understand complicated instructions and give answers that are based on the situation, which makes them good for making smart automation systems.

By combining LLMs with browser automation frameworks, you can make autonomous agents that can understand user commands, plan actions that take more than one step, and interact with real-world web interfaces. The ReAct approach makes this even better by letting the system keep reasoning and acting in a loop, which lets it change to fit new web environments.

We suggest an LLM-Based Autonomous Browser Agent that fills the gap between traditional rule-based automation and smart task execution in this work. The system takes natural language commands through text or voice input, uses an LLM to process them, and then carries out the actions in a real browser environment. It can handle interactions in multiple languages, fill out forms automatically using saved user profiles, and recover from errors to ensure smooth operation.

The proposed system's goal is to make it easier to interact with the web by giving users a single, smart, and easy-to-use platform that can do all of their web tasks on its own. This method shows how AI-driven automation can boost productivity, cut down on manual work, and make the user experience better in many different applications.

II. LITERATURE REVIEW

Because internet technologies are growing so quickly, more and more people are using web platforms to do things like apply for jobs, look for scholarships, and find information.

But interacting with many websites by hand takes a lot of time and isn't very efficient, which is why automated solutions are needed. In the past, people used rule-based browser automation tools like Selenium WebDriver and Playwright, which let you interact with web elements by clicking and filling out forms [10]. These systems work well, but they rely on prewritten scripts and can't adapt to changing web environments. Improvements in Natural Language Processing (NLP) made it possible for systems to understand questions from users in plain language. Jurafsky and Martin's early work brought attention to ways to understand language [7]. But these kinds of systems could only get information and couldn't do things on the web in the real world. The advent of Large Language Models (LLMs) markedly enhanced automation capabilities. Brown et al. showed that LLMs can learn from only a few examples [1], and Vaswani et al. came up with transformer architectures [9]. Wei et al.'s ReAct framework made it possible for models to combine reasoning and action to do tasks in more than one step [2]. Tree of Thoughts also made structured reasoning better in complicated situations [3]. Recent research has investigated the integration of LLMs with web environments. WebShop showed that language agents can work with web interfaces [4]. These methods show how powerful LLM-driven agents can be, but they don't always work well with real-world browser automation systems. Speech interaction makes things even easier to get to. Rabiner and Juang's research laid the groundwork for speech recognition systems [6], which made it possible to talk to smart systems. Even with these improvements, current systems still have trouble dealing with dynamic web content, input in multiple languages, and making decisions in real time. A single system that combines LLM reasoning with real browser automation is needed. This project builds on these advances to create an LLM-based autonomous browser agent that can understand natural language commands and do web tasks quickly.

III. SYSTEM REQUIREMENTS AND DESIGN

CONSIDERATIONS

To work well and reliably, the proposed system needs both hardware and software parts. To support browser automation and real-time processing, a system must have at least an Intel i5 processor, 8 GB of RAM, and a stable internet connection. You need a microphone for voice input to work. The system is built with Python 3.10 or higher and uses frameworks like Streamlit for the user interface and browser-use for combining LLM-based reasoning with browser control. Playwright takes care of the browser automation, which lets you use modern web apps.

The system design uses a modular and layered structure to make sure it can grow, be fixed, and be changed easily. There are many layers, such as the presentation layer, translation layer, task routing layer, agent layer, and browser control layer. The system can process user input, convert it to a standard format, figure out what the right task is, and carry out actions through specialized agents thanks to the different layers. This modular design makes it easier to find and fix bugs and add new features or agents.

During development, a number of important design factors were taken into account. The system is set up to accept natural language input, so users can interact without needing to know anything about technology. It has multilingual support so that people can enter information in different languages, which makes it easier to use. The system also has voice input and output features so that you can use it without having to touch anything. Strong error handling systems are in place to deal with problems like content that isn't available or navigation errors, making sure that tasks are done correctly.

Security and usability are also important aspects of the system. User authentication and profile management are implemented to securely store personal data such as name, email, and resume, which are used for automatic form filling. Sensitive information such as passwords is protected using hashing techniques. The user interface is designed to be simple and intuitive, allowing users to easily provide commands and view results.

Overall, the system is designed to provide a unified, intelligent, and user-friendly platform that combines AI-based reasoning with real browser automation, enabling efficient execution of complex web tasks.

IV. PROPOSED SYSTEM

The suggested system is an LLM-based Autonomous Browser Agent that can use natural language instructions to automate tasks on the web. The system combines Large Language Model (LLM) reasoning with real-time browser automation to let you do smart things like search for information, fill out forms, and move around websites. The system is more flexible and adaptable than traditional rule-based automation tools because it combines AI with browser control.



Figure 4.1 Architecture of the system

The system's architecture is made up of layers for presentation, translation, routing, agents, and browser control. Streamlit is used to build the presentation layer, which is an interactive user interface that lets users enter tasks by typing or speaking. The translation layer takes input in multiple languages and changes it into English so that tasks can be done the same way every time. The routing layer looks at the processed input and sends it to the right specialized agent based on the type of task.

The agent layer has a number of domain-specific agents, like YouTube, internship, scholarship, LinkedIn, and general browsing agents. Each agent is made to do a certain type of task using structured prompts and pre-planned ways to do things. These agents use the browser-use framework, which is a mix of LLM reasoning and browser automation. The Playwright-powered browser control layer lets you interact with real web apps by clicking on things, filling out forms, and uploading files.

The ReAct method is used by the system to work. It uses a reasoning-action loop. The agent looks at the current state of the webpage, thinks about what to do next, does it, and then updates its understanding based on what happened. The system can change dynamically to fit different web environments because this process keeps going until the job is done.

The system also has user authentication and profile management features, which keep track of personal information so that forms can be filled out automatically, saving users time and effort. The system also lets you speak and hear, which makes it easier for people to use and interact with. To make sure that tasks are done correctly, error handling mechanisms are built in to deal with problems like content that isn't available or navigation that doesn't work.

The suggested system is an intelligent, scalable, and user-friendly platform that lets people do complex web tasks on their own using natural language commands. It fills the gap between traditional automation and AI-driven solutions.

V. IMPLEMENTATION

The proposed system is built on Python as the main programming language and uses a number of frameworks and libraries to make smart browser automation possible. Streamlit is used to make the user interface, which lets users enter tasks by typing or speaking. The application works as a local web-based system, which makes it easy to access without having to go through a lot of trouble to set it up.

The browser-use framework, which combines Large Language Model (LLM) reasoning with browser automation, powers the main features. The system is built on a task-based architecture, which means that each user command is processed and sent to a specific agent. These agents are implemented as asynchronous functions using Python's asyncio library, which makes it easy to run many steps in a task at once.

Playwright takes care of the browser automation, which lets the system interact with real web pages. It lets you do things like open websites, click on things, fill out forms, upload files, and get information. The system can follow a reasoning-action loop because it works with browsers. This lets it make decisions on the fly while doing tasks.

The deep-translator library is used by the system to turn user input from any language into English, which makes sure that processing is always the same. The SpeechRecognition library is used to implement voice input. It captures audio from the microphone and uses Google's speech recognition service to turn it into text. The pyttsx3 library handles voice output, which lets you use text-to-speech even when you're not connected to the Internet.

There is also a module for user authentication and profile management in the system. This is where user information like name, email address, phone number, and resume are safely stored. This information is used to automatically fill out forms for job and internship applications, which saves time. To keep passwords safe, hashing techniques are used to protect them.

To make the system more reliable, it has built-in error handling and execution control mechanisms. To keep the agents from getting stuck in infinite loops, they have step limits and time limits. The system also has basic ways to fix errors, like trying again when content isn't available or finding other options.

The implementation brings together AI-based reasoning, browser automation, and user interaction into a single system that can do complicated web tasks quickly and easily, with little help from the user.

VI. RESULTS

We tested the LLM-Based Autonomous Browser Agent on a number of real-world web automation tasks, and it worked quickly and reliably. The system was able to do things like general web searches, play YouTube videos, send WhatsApp messages, apply for scholarships, apply for internships, and follow voice commands. The system could understand natural language commands and carry out multi-step actions on its own thanks to the combination of Large Language Models and browser automation.



Figure 6.1 Performance of Task Execution

The results show that the ReAct (Reasoning + Acting) framework lets the system handle dynamic web interactions well. In YouTube automation, the agent was able to find and play videos based on what the user said. The system was able to move around the platform, choose the right content, and start playing it back without any help from a person, which showed that it could do the job correctly.

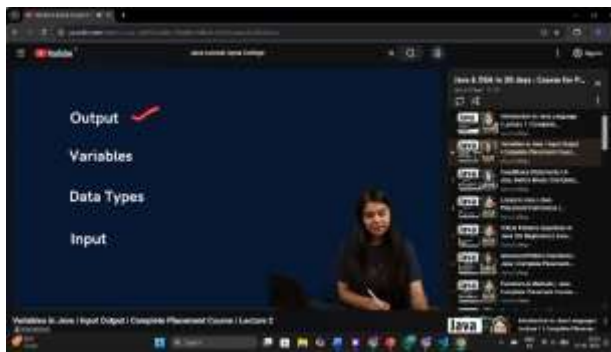


Figure 6.2 YouTube Automation

The system also worked well for communication tasks like sending messages on WhatsApp. The agent was able to open WhatsApp Web, wait for QR-based authentication, find the right contact or group, and send messages correctly. This shows that the system can work with web apps that run in real time.



Figure 6.3 WhatsApp Automation

The system showed that it could automate tasks like applying for scholarships and internships very well. The agent could find the right platforms, fill out the required form fields with information from the user's profile, and automatically upload resumes. The system took care of login requirements for internship applications and successfully completed the process up to the review stage, making sure that users were in control before the final submission.

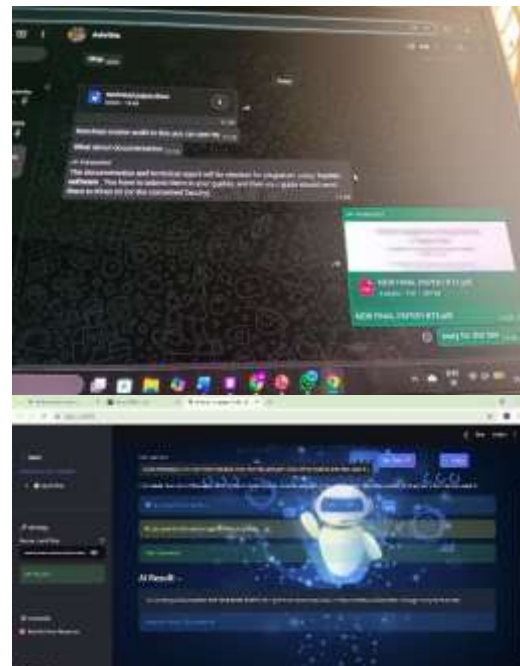
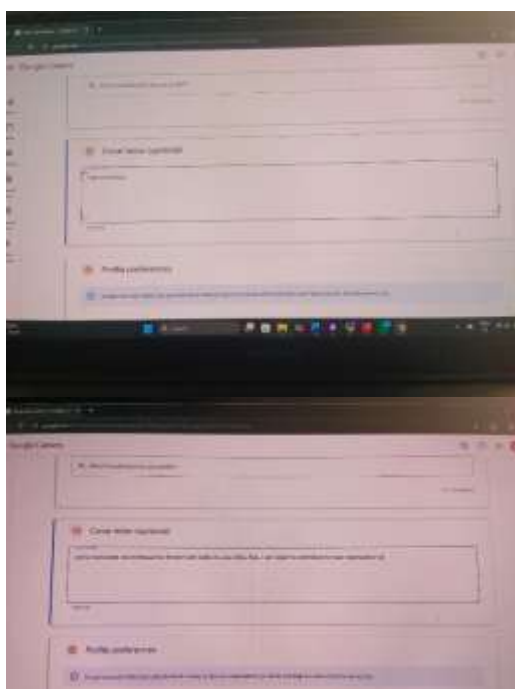


Figure 6.4 Flow of Automated Applications

The system also showed off advanced features like voice-based interaction. Speech recognition successfully picked up voice commands, turned them into text, and sent them to the task router. The system did the requested task with very little delay, which shows how voice integration can make things easier for users.



Figure 6.5 Task Execution by Voice

The Streamlit-based user interface made it easy and fun for users to interact with the program. Task history, execution time display, profile-based auto-fill, and voice response are some of the features that made the program easier to use and more efficient. The asynchronous execution model made sure that the system stayed responsive while tasks were being carried out.



Figure 6.6 The system's user interface

The modular design of the system also shows that it can grow easily. Different agents work on their own and can be added to support more platforms without changing the main system. This flexibility lets the system change to meet new needs and handle more complicated automation situations.

VI COMPARATIVE ANALYSIS WITH EXISTING SYSTEMS

Selenium and Playwright are two examples of traditional browser automation systems that use predefined scripts and element selectors to interact with web pages. These systems work well in controlled environments, but they are very sensitive to changes in website structure and need to be maintained all the time. Also, they don't understand natural language, can reason, or are easy to use, which makes them hard for people who aren't tech-savvy to use.

Recent improvements in Large Language Models (LLMs), on the other hand, have made it possible for systems like AutoGPT and WebGPT to do task reasoning and decomposition. But these systems mostly work in text-based settings and can't directly control real web browsers. Because of this, they can't do complicated tasks in the real world, like filling out forms, uploading files, or dealing with authentication workflows on web apps that change.

Voice assistants like Alexa and Siri let users talk to them and type to them, which makes them easier to use. But they can only use predefined skills and built-in services. They can't navigate random websites on their own or carry out multi-step tasks like applying for a job or extracting data.

Mind2Web is a more advanced approach because it lets LLM-based agents interact with real web environments. It shows good reasoning skills, but it needs a lot of work on large annotated datasets and doesn't have useful features like voice interaction, support for multiple languages, or automation based on user profiles.

The suggested system gets around these problems by using the ReAct framework to combine LLM-based reasoning with real browser control through Playwright. It lets people do complicated, multi-step web tasks using natural language commands without needing to know how to code. The system also has voice input and output, multilingual translation, profile-based auto-fill, resume upload, and login-resilient execution, which makes it more useful and easy to use in the real world.

In general, the proposed system is a single, smart solution that fills the gap between old automation tools and new AI-driven agents. It makes tasks easier to adapt, use, and automate from start to finish.

Table 6.1: Feature Comparison of Existing Systems and Proposed System

Feature	Selenium	AutoGPT	Alexa/Siri	Proposed
Natural Language	No	Yes	Yes	Yes
Voice Interaction	No	No	Yes	Yes
Browser Control	Yes	No	No	Yes
LLM Reasoning	No	Yes	Partial	Yes
Auto Form Fill	No	No	No	Yes
Error Recovery	No	Partial	No	Yes

VII. FUTURE ENHANCEMENTS

The suggested LLM-based autonomous browser agent shows that it can do a lot of web-based tasks automatically by following natural language instructions. However, there are a number of changes that can be made to make it work better, be more scalable, and be easier to use. One big change is that advanced planning and memory systems are now part of the program. The system now works on a task-by-task basis and doesn't have long-term memory. If the agent had persistent memory, it could learn from past interactions, remember what the user liked, and make future tasks run more smoothly. Another important improvement is the ability to run tasks at the same time. The current version only lets you process one task at a time. Allowing multi-agent or parallel execution would let the system do more than one thing at a time, which would make it more efficient and usable in real time. You can also add to the system by replacing the keyword-based router with an LLM-based intent classification module. This would make routing more accurate and let you interpret tasks in a more flexible and context-aware way. Another important area is making systems more stable in complicated web environments. In the future, we could work on better ways to handle CAPTCHAs, more secure ways to log in, and dynamic UI changes using better perception techniques like visual grounding and DOM understanding. Adding mobile and cloud deployment would make it much easier to access. If the system were set up as a cloud-based service or mobile app, users could access the agent from anywhere without having to set it up on their own. Integration with external APIs and services is one way to make things even better. This would let the agent do more than just automate browsers; it could also schedule calendars, automate emails, and manage documents. Finally, it's important to make sure that execution speed and cost-effectiveness are as high as possible. You can use caching, selective reasoning, and lightweight models to cut down on latency and API usage costs while keeping performance the same.

VIII. CONCLUSION

This paper introduced an LLM-based Autonomous Browser Agent intended to automate intricate web-based tasks through natural language directives. The system combines Large Language Models with real browser control using the ReAct (Reasoning and Acting) framework. This lets it intelligently, adaptively, and completely complete tasks in dynamic web environments.

The proposed system shows that it can do a lot of different things with little help from the user, like web browsing, multimedia control, messaging, job and internship applications, and data extraction. Adding support for multiple languages, voice input and output, and profile-based auto-fill makes the app much easier to use and more accessible. The system also does a good job of handling real-world problems like needing to log in and not being able to access content by using pause-and-resume execution and error recovery methods.

The proposed solution offers a cohesive platform that integrates reasoning, adaptability, and usability, eliminating the need for programming expertise or established workflows, in contrast to conventional automation tools and current AI-based systems. This makes it great for people who aren't tech-savvy and for use in the real world.

The system does add some latency because it uses LLM-based reasoning and relies on external APIs, but it is much more flexible and intelligent than other methods. The project shows how LLM-driven autonomous agents could change the way people use the web. This could lead to more advanced, user-focused automation systems in the future.

IX. REFERENCES

- [1] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, Jirong Wen (2024) A Survey on Large Language Model Based Autonomous Agents. *Frontiers of Computer Science*. <https://link.springer.com/article/10.1007/s11704-024-40231-1>
- [2] Yao, S., et al., "ReAct: Synergizing Reasoning and Acting in Language Models." *arXiv Research Paper*. <https://arxiv.org/abs/2210.03629>
- [3] Tom B. Brown, Benjamin Mann, Nick Ryder et al. (2020) Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems (NeurIPS)* 33:1877-1901. <https://arxiv.org/abs/2005.14165>
- [4] Shunyu Yao, Dian Yu, Jeffrey Zhao et al. (2023) Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *arXiv preprint arXiv:2305.10601*. <https://arxiv.org/abs/2305.10601>
- [5] Shunyu Yao, Howard Chen, John Yang, Karthik Narasimhan (2022) WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. *arXiv preprint arXiv:2207.01206*. <https://arxiv.org/abs/2207.01206>
- [6] Microsoft (2022) Playwright: Fast and Reliable End-to-End Testing for Modern Web Apps. *Software Documentation*. <https://playwright.dev/python/docs/intro>
- [7] Oriol Vinyals, Igor Babuschkin et al. (2022) WebShop: Towards Scalable RealWorld Web Interaction with Grounded Language Agents. *arXiv preprint arXiv:2207.01206*.

- [8] Daniel Jurafsky, James H. Martin (2009) *Speech and Language Processing*, 2nd Edition. Pearson.

- [9] Ashish Vaswani, Noam Shazeer et al. (2017) Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*

- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova (2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACLHLT*.

- [11] Lawrence R. Rabiner, Biing-Hwang Juang (1993) *Fundamentals of Speech Recognition*. Prentice Hall.