Devi Suhitha Chundru

Browser Automation with AI Agents: Automating Repetitive Web Tasks Using WebUI

Project Overview

As part of my exploration into **AI** and browser automation, I built a personal **AI agent** capable of performing repetitive web-based tasks such as filling out forms, searching online, and navigating websites all automatically. This project demonstrates how AI can simplify everyday digital workflows by mimicking human interactions in the browser.

Inspired by the concept of **AI-powered agents,** I implemented my own solution using **WebUI** and integrated it with browser automation capabilities. This agent can simulate keystrokes, mouse clicks, and intelligently follow prompts to complete assigned web tasks like logging into a site or entering search queries.

Project Goal

→ The primary goal of this project is to create a self-operating browser agent that:

- Understands and executes web-based tasks.
- Navigates through websites like Google or LinkedIn.
- Automates routine processes like form filling and information retrieval.
- Reduces manual effort and boosts productivity through intelligent prompt engineering.

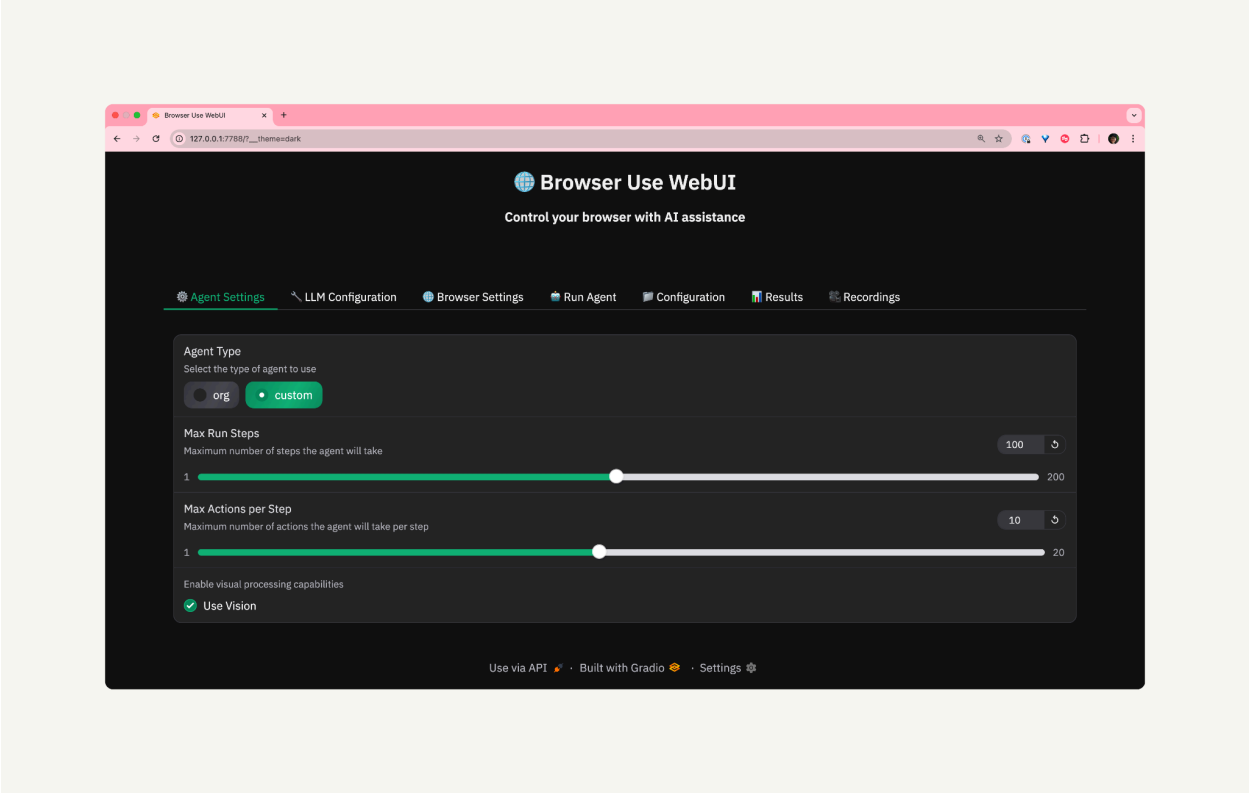
Tools and Technologies Used

Python	Core programming language for automation
Pip	Package manager for Python dependencies
Git	Version control system for project tracking
WebUI (Browser-UI)	Automation framework for controlling browser
Google Chrome	Target browser for automation
Google AI Studio API	For powering smart AI interactions
Terminal / CMD	Running scripts and installations

Devi Suhitha Chundru

Step 1: Set Up My Development Environment

In this project, I used **Browser Use's WebUI**. **Browser Use** is a developer tool that connects AI models with my browser, while **WebUI** adds a visual interface - so I could work with Browser Use with **clicks instead of code.**



A sneak peek of WebUI, the visual interface for Browser Use

To get WebUI up and running on my system, I needed to install a few development tools:

Python - Python is the programming language that powers WebUI and allows it to run on my computer. I used it to execute the WebUI automation scripts and handle the core logic.

Pip & UV - Pip and UV act like app stores for Python. I used them to download and install the necessary Python packages. These are pre-built code libraries that WebUI depends on to function properly.

Git - I used Git to download the latest version of WebUI's source code from its official online repository. It ensured I had the most up-to-date version of the tool for my project.

In this step, I performed the following installations to prepare my system for running WebUI:

- I installed Python, the programming language required to run WebUI on my local machine.
- I set up Pip and UV, which are Python package managers. These tools helped me install all the necessary dependencies required by WebUI.
- I installed Git, a version control tool, which I used to download the WebUI source code directly from its online repository.

My System Check - Pre-Installed Tools

Before proceeding with setup, I verified which development tools were already installed on my system using Windows PowerShell (Admin).

On Windows:
I opened PowerShell as Administrator
(Right-click on Start > Windows PowerShell (Admin))

Then I ran these commands:

```
python --version    # Check Python
pip --version       # Check Pip
uv --version        # Check UV
git --version       # Check Git
```

- If PowerShell displayed a version number, the tool was already installed.
- If it showed "**not recognized**", I marked that tool for installation.

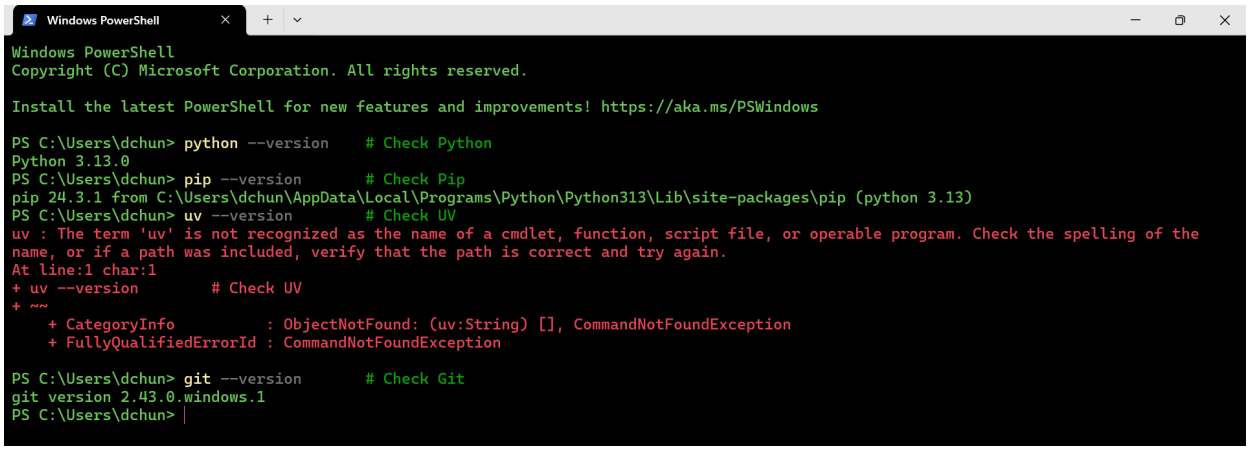
This initial check helped me skip unnecessary installations and focus only on the tools I didn't already have.

These are the results:

Tool	Command Used	Result
Python	python --version	✅ Python 3.13.0 was already installed.
Pip	pip --version	✅ Pip 24.3.1 was available and correctly linked to Python 3.13.
UV	uv --version	❌ PowerShell returned a "command not recognized" error – not installed.

Git git --version ✅ Git version 2.43.0.windows.1 was already installed.

From this check, I confirmed that Python, Pip, and Git were already available, while UV still needed to be installed.



Why are we installing UV?

Note: Please make sure Pip is installed first, before installing UV.

UV will be our main package manager because:

- It downloads and installs Python packages much faster than Pip.
- It handles dependency conflicts better.
- It makes installing WebUI's requirements more efficient.

Think of it like we're using a better, faster app store (UV) than the default, basic app store (Pip)!

Here are the instructions for installing UV:

Windows → In PowerShell,

run: pip install uv

```
PS C:\Users\dchun> pip install uv
Collecting uv
  Downloading uv-0.7.12-py3-none-win_amd64.whl.metadata (12 kB)
Downloading uv-0.7.12-py3-none-win_amd64.whl (18.8 MB)
   ---------------------------------------- 18.8/18.8 MB 4.1 MB/s eta 0:00:00
Installing collected packages: uv
Successfully installed uv-0.7.12

[notice] A new release of pip is available: 24.3.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\dchun>
```

Verify UV installation:

```
PS C:\Users\dchun> uv --version
uv 0.7.12 (dc3fd4647 2025-06-06)
```

Final Check - Let's Make Sure You Have Everything

Before moving forward, I did a final verification to make sure all required tools were correctly installed and accessible from the command line.

I ran the following commands in PowerShell (Admin):

- python --version # Confirmed Python installation
- pip --version # Confirmed Pip is working
- uv --version # Checked if UV is installed
- git --version # Verified Git version

```
PS C:\Users\dchun> python --version # Check Python
Python 3.13.0
PS C:\Users\dchun> pip --version # Check Pip
pip 25.1.1 from C:\Users\dchun\AppData\Local\Programs\Python\Python313\Lib\site-packages\pip (python 3.13)
PS C:\Users\dchun> uv --version # Check UV
uv 0.7.12 (dc3fd4647 2025-06-06)
PS C:\Users\dchun> git --version # Check Git
git version 2.43.0.windows.1
```

To upgrade your tools to the latest version, run the following commands:

- python -m pip install --upgrade pip # Upgrade Pip to the latest version
- pip install --upgrade uv # Upgrade UV to the latest version

Devi Suhitha Chundru

Step 2: Install and Run WebUI

Now that I had my development environment set up, I moved on to downloading and running WebUI.

Why did I need to download WebUI's code manually?

WebUI isn't a traditional application that can be installed through an app store or installer. It's an open-source tool hosted on GitHub, which means I had to clone its source code and run it locally using Python.

What I Did in This Step

- Cloned WebUI's source code from GitHub using git clone.
- Set up a Python virtual environment to manage dependencies cleanly.
- Ran WebUI using the provided command-line scripts.

With the setup complete, I was ready to launch the WebUI interface and begin automating my browser using AI 🤖.

Downloading WebUI's Code

To begin working with WebUI, I needed to download its source code from GitHub. Since WebUI is an open-source project, the codebase is hosted publicly and must be cloned to my local machine before use.

The steps to downloading WebUI's code depend on my operating system:

Steps I Followed

1. Navigated to the Documents folder
I opened PowerShell and ran:

```
cd ~/Documents
```

This command moves me into the Documents directory, where I wanted to store the WebUI code.

2. Cloned the WebUI repository

I executed:

```
git clone https://github.com/browser-use/web-ui.git
```

Cloning the repository downloaded a full copy of WebUI's source code into my Documents folder.

```
PS C:\Users\dchun> cd ~/Documents
PS C:\Users\dchun\Documents> git clone https://github.com/browser-use/web-ui.git
Cloning into 'web-ui'...
remote: Enumerating objects: 2692, done.
remote: Total 2692 (delta 0), reused 0 (delta 0), pack-reused 2692 (from 2)Receiving objects:  99% (2666/2692), 12.95 MiB | 25.88 MiB


Resolving deltas: 100% (1638/1638), done.
```

Why am I cloning the repository?

Cloning a repository means creating a local copy of the entire codebase from a remote source, in this case, GitHub. This allows me to work on the project locally, make changes, and track my modifications without affecting the original repository.

Recap: What is WebUI?

WebUI is an open source interface for Browser Use that helps us create browser agents powered by AI models. Think of Browser Use as the engine of a car, and WebUI as the steering wheel and dashboard that helps us drive the car.

Browser Use connects AI models with browsers, and WebUI translates Browser Use's code into a graphical portal, so creating AI browser agents is easier and more visual! Think of the AI browser agents as smart assistants that can:

- Navigate websites and click buttons
- Fill out forms and search fields
- Extract information from web pages
- Make decisions about what to do next

Simply tell the automation what to do in plain English, and the AI model will help figure out how to complete the task!

3. Entered the cloned project directory

I ran:

cd web-ui

The **web-ui** folder was created by Git during the cloning process. It contains all the files necessary to run WebUI.

Why am I navigating into the web-ui directory?

This command changes my current directory to the newly created web-ui folder, where all the project files are located.

```
PS C:\Users\dchun\Documents> cd web-ui
PS C:\Users\dchun\Documents\web-ui>
```

4. Verified my location inside the project
I used the command:

pwd

This command confirms my current working directory. I used it to ensure I was inside the web-ui folder before running further setup commands.

```
PS C:\Users\dchun\Documents\web-ui> pwd

Path
----
C:\Users\dchun\Documents\web-ui
```

Set Up the Python Virtual Environment

I set up a Python virtual environment to keep my project dependencies isolated and organized. This ensured that any packages I installed would not interfere with system-wide Python settings or other projects.

What is a Python virtual environment?

- A virtual environment is a self-contained workspace that includes:
- A dedicated Python interpreter (e.g., Python 3.11)
- A package manager like pip or uv
- All project-specific dependencies

The main benefit is that the environment is temporary and fully isolated. Once I deleted the environment folder, all installed packages were removed automatically keeping my system clean and avoiding version conflicts.

To begin, I created a Python virtual environment using the **uv** tool with Python 3.11. This allowed me to keep the project dependencies isolated from my global Python setup.

For Windows:

I created the virtual environment:

uv venv --python 3.11

This command created a new virtual environment in the .venv directory using Python 3.11.

Activated the virtual environment:

Depending on my operating system, I used one of the following commands to activate the environment.

On Windows:

.venv\Scripts\activate

```
PS C:\Users\dchun\Documents\web-ui> .venv\Scripts\activate
.venv\Scripts\activate : File C:\Users\dchun\Documents\web-ui\.venv\Scripts\activate.ps1 cannot be loaded because running scripts
is disabled on this system. For more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .venv\Scripts\activate
+ ~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

It means that **PowerShell's execution policy is set to block running scripts**, including the activate.ps1 file for Python virtual environments.

I ran this command in **PowerShell as Administrator**:

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

Then tried activating my virtual environment again:

```
PS C:\Users\dchun> .venv\Scripts\activate
(dchun) PS C:\Users\dchun>
```

Install Required Packages

After setting up and activating the environment, I installed all the necessary packages listed in the **requirements.txt** file by running:

running:

uv pip install -r requirements.txt

This command ensured that all dependencies needed for the WebUI project were downloaded into the virtual environment.

What are packages?

Packages are pre-written code that adds specific functionality to my project. The requirements.txt file lists all the packages that WebUI needs to run.

What are the packages in requirements.txt?

Here are the key packages we're installing:

- **playwright**: For controlling web browsers programmatically.
- **google-generativeai**: Google's AI API for generating text and making decisions
- **python-dotenv**: For managing environment variables and configuration
- **FastAPI**: A modern web framework for building APIs
- **uvicorn**: A lightning-fast web server for running our application
- **jinja2**: A template engine for generating dynamic HTML

Will these stay on my computer forever?

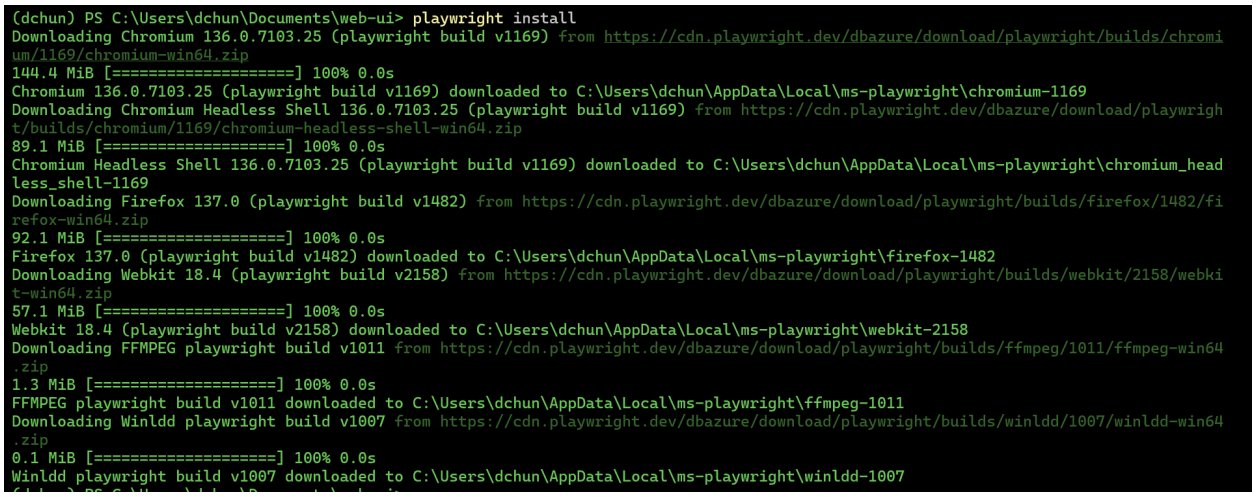
No! Because I'm using a Python environment, these packages are only installed in that isolated environment. They won't affect the rest of my computer, and I can safely delete everything by removing the project folder when I am done. This is one of the main benefits of using virtual environments - they keep my projects neat and isolated.

Once inside the **web-ui** folder, install the required packages:

```
(dchun) PS C:\Users\dchun> cd C:\Users\dchun\Documents\web-ui
(dchun) PS C:\Users\dchun\Documents\web-ui> uv pip install -r requirements.txt
Using Python 3.11.13 environment at: C:\Users\dchun\.venv
Resolved 164 packages in 1m 29s
      Built ibm-cos-sdk-core==2.13.5
      Built ibm-cos-sdk-s3transfer==2.13.5
      Built pyperclip==1.9.0
      Built ibm-cos-sdk==2.13.5
Prepared 164 packages in 1m 09s
Installed 164 packages in 9.07s
 + aiofiles==24.1.0
 + aiohappyeyeballs==2.6.1
 + aiohttp==3.12.9
 + aiosignal==1.3.2
 + annotated-types==0.7.0
 + anthropic==0.52.2
 + anyio==4.9.0
 + attrs==25.3.0
 + babel==2.17.0
 + backoff==2.2.1
 + beautifulsoup4==4.13.4
 + boto3==1.38.32
 + botocore==1.38.32
 + browser-use==0.1.48
 + cachetools==5.5.2
 + certifi==2025.4.26
 + charset-normalizer==3.4.2
 + click==8.2.1
 + colorama==0.4.6
 + courlan==1.3.2
 + dataclasses-json==0.6.7
 + dateparser==1.2.1
 + defusedxml==0.7.1
 + distro==1.9.0
 + faiss-cpu==1.11.0
```

Next, I installed Playwright:

playwright install



What is Playwright?

Playwright is a tool that lets us control web browsers programmatically. It's a browser automation framework that can:

- Open and control Chrome, Firefox, and Safari browsers
- Click buttons, fill forms, and navigate websites
- Take screenshots and record videos

What's the difference between Playwright and WebUI?

WebUI uses Playwright, but it also adds:

- A friendly web interface where we can type instructions in plain English
- AI integrations to help us understand and execute our instructions

Check That Playwright Was Installed

After installing the required packages, I verified that **Playwright** was installed correctly by running:

playwright --version

Devi Suhitha Chundru

Set Up Environment File

To configure the application, I created a personal environment file by copying the example template:

cp .env.example .env

What is an .env file?

An environment file (.env) stores configuration settings for an application. In my case, it stores settings and tells WebUI how to run e.g. which AI model it should use.

Why are we making a copy?

We copy from .env.example to create our own .env file because:

- Each developer might need different settings (like different API keys or paths).
- The example file (that comes with the GitHub repository) contains a template of all the settings we need.
- We don't want to modify the example file yet, but I'll need one to run WebUI.
- It's a common security practice to keep sensitive information (like API keys) in your own .env file that isn't shared with others.

```
(dchun) PS C:\Users\dchun\Documents\web-ui> cp .env.example .env
(dchun) PS C:\Users\dchun\Documents\web-ui>
```

Run My Agent in WebUI

Finally, I brought WebUI to life by running the following command in the terminal:

python webui.py --ip 127.0.0.1 --port 7788

What does this command do?

This command starts my WebUI server:

- python3 webui.py runs the Python script named webui.py, which contains the code to run WebUI on my computer.
- 127.0.0.1 is a special IP address that means "this computer" - it's like my computer's home address (also known as localhost).
- port 7788 is like a specific door number where WebUI will be accessible.
- Using this local address means WebUI is private and secure - only my computer can access it.

```
(dchun) PS C:\Users\dchun\Documents\web-ui> python webui.py --ip 127.0.0.1 --port 7788
* Running on local URL:  http://127.0.0.1:7788

To create a public link, set `share=True` in `launch()`.
```

Verified WebUI Was Running

Once I ran the command to start the server, I saw the following message in my terminal:

"Running on local URL: http://127.0.0.1:7788"

What's running?

At this point, WebUI was up and running as a **local web server**. It was like having my own private website available only on **my computer**.

Viewed in the Browser

I opened my web browser and went to:

http://127.0.0.1:7788

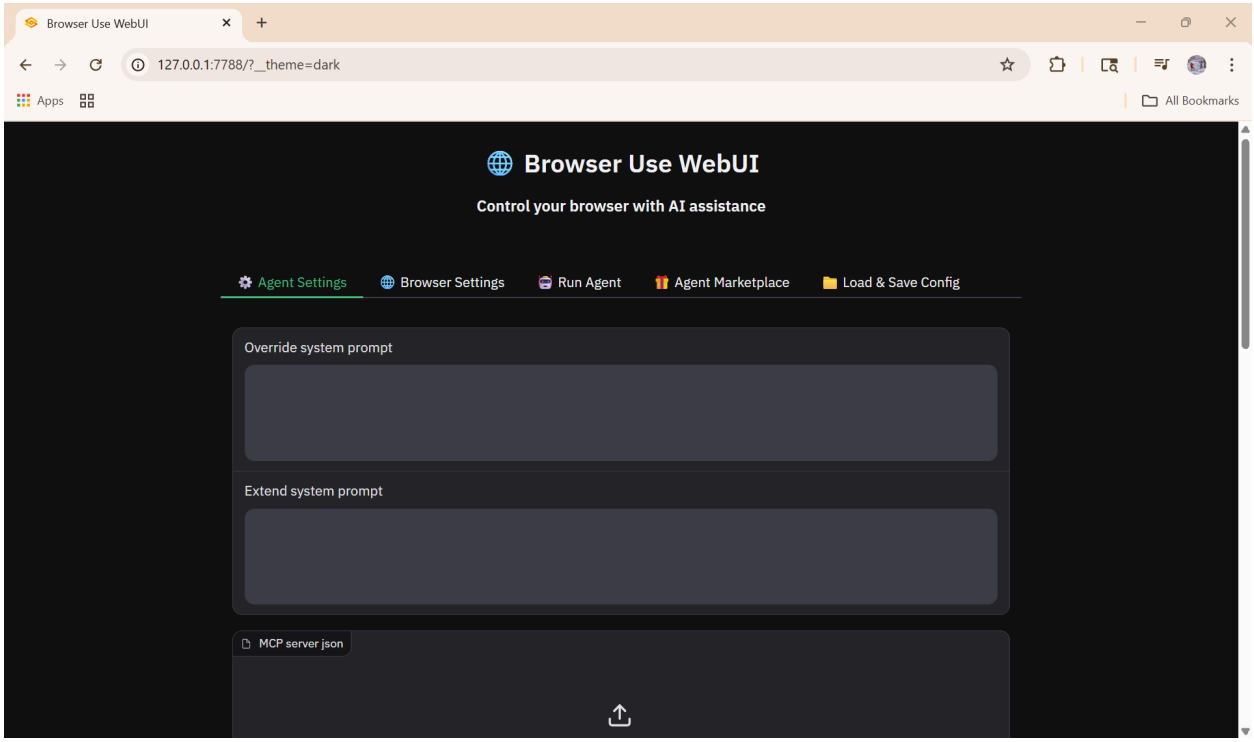
And just like that, the WebUI interface is loaded!

To try out the dark theme, I also visited:

http://127.0.0.1:7788/?__theme=dark

Woohoo! Welcome to WebUI - I made it! 🙌

The interface was up and running, and I was ready to start interacting with my AI agent through the browser.



What's the difference between Browser Use and WebUI?

Think of Browser Use as the engine of a car, and WebUI as the steering wheel and dashboard that helps you drive the car.

Browser Use connects AI models with browsers, and WebUI translates Browser Use's code into a graphical portal, so creating AI browser agents is easier and more visual!

Simply tell the automation what to do in plain English, and the AI model will help figure out how to complete the task!

Devi Suhitha Chundru

Started Using WebUI

I started using WebUI by running the application and trying a simple prompt to test it out.

What I Did in This Step

In this step, I:

- **Ran WebUI** and, as expected, ran into an error because no API key was configured yet.
- **Obtained an API key** from **Google AI Studio** by signing in, navigating to the API Keys section, and generating a new key.
- **Stored the API key securely** by adding it to my **.env** file, which kept it private and separate from my source code.

These steps allowed me to configure WebUI correctly so it could communicate with Google's AI services.

Quick Tour of WebUI

As a recap, WebUI is how I can create an AI-powered browser agent. Here's a quick rundown of the tabs I see.

⚙️ **Agent Settings** - Controls how my agent behaves when browsing websites e.g. a maximum number of clicks it can make.

🔧 **LLM Configuration** - Connects my agent to an AI model e.g. OpenAI, Gemini, DeepSeek.

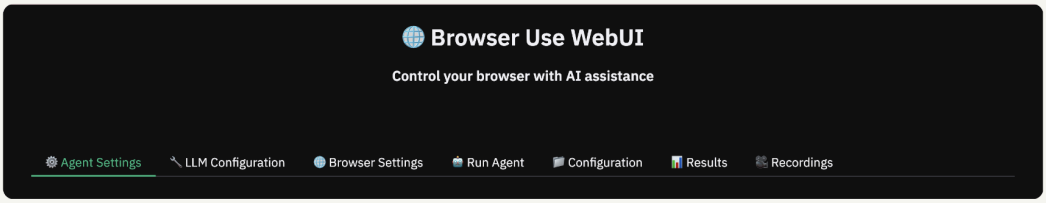
🌐 **Browser Settings** - Selects which browser I want to use.

🤖 **Run Agent** - Defines tasks and starts my agent.

📁 **Configuration** - Automates my WebUI settings by letting you upload a template configuration file.

📊 **Results** - Shows logs of what my agent did, once it's completed a task.

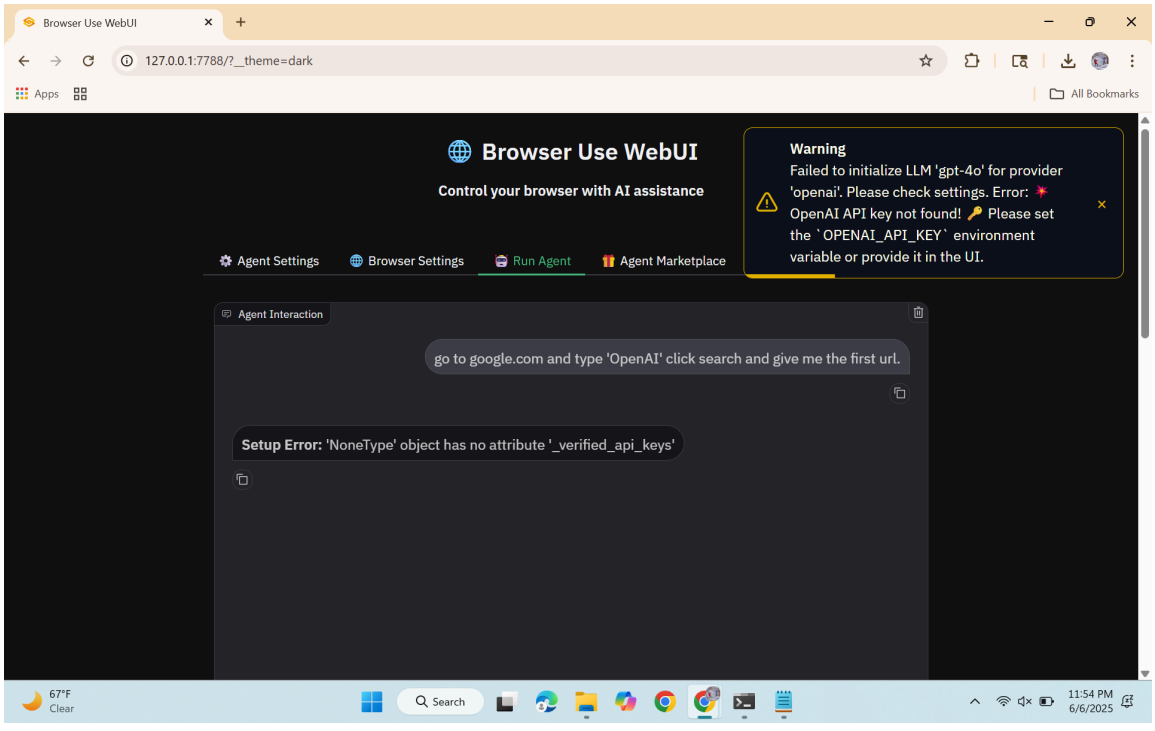
🎥 **Recordings** - Saves videos of my agent's actions.



Run WebUI (and see an error)

Let's see what happens when I tried to use WebUI straight away:

- I went to the **Run Agent** tab.
- I kept the default prompt in the Task Description: **go to google.com and type 'OpenAI' click search and give me the first url.**
- I clicked the **Run Agent** button.
- I saw an error.



Devi Suhitha Chundru

Oops! I have seen this error!!

💥 OpenAI API key not found!

🔑 Please set the OPENAI_API_KEY environment variable or provide it in the UI.

Why did I get this error?

WebUI needs to connect to an AI model to understand and execute my instructions. But, I haven't set up the connection to an AI model yet!

It's like I'm trying to make a phone call without a phone plan - I need to set up the service (in this case, the API key) before I can use it.

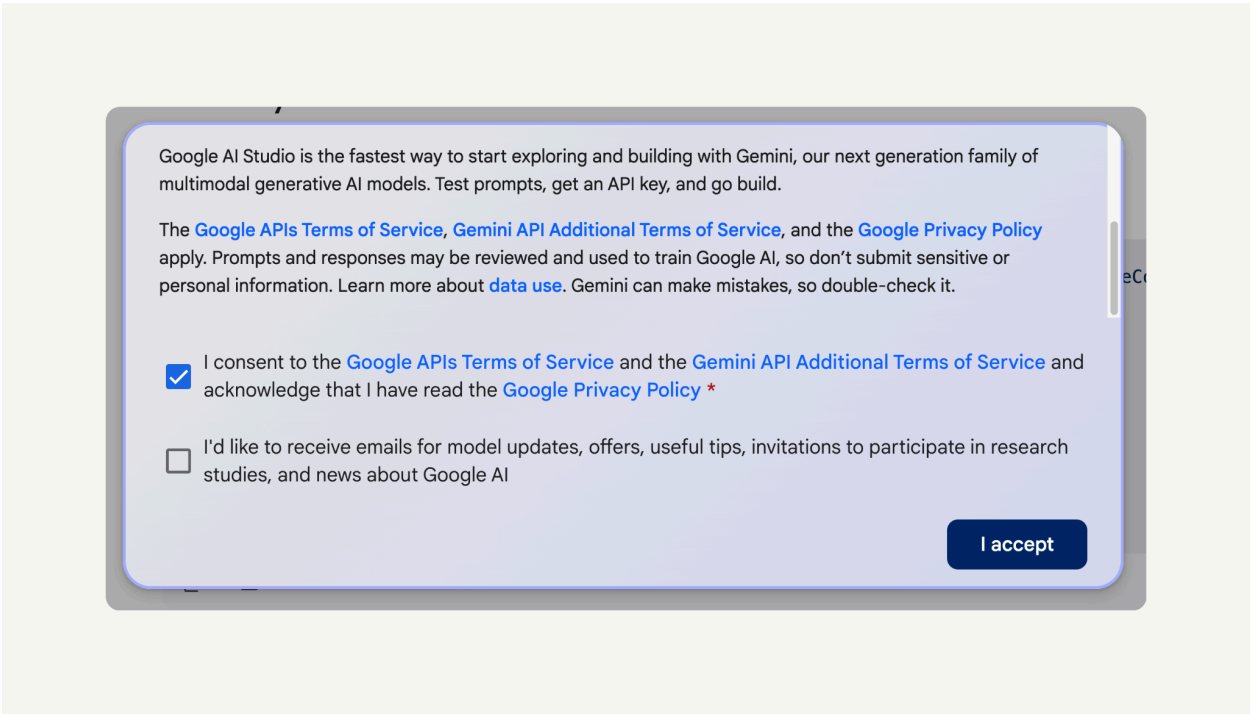
Let's fix this by getting an API key...

Get API Key

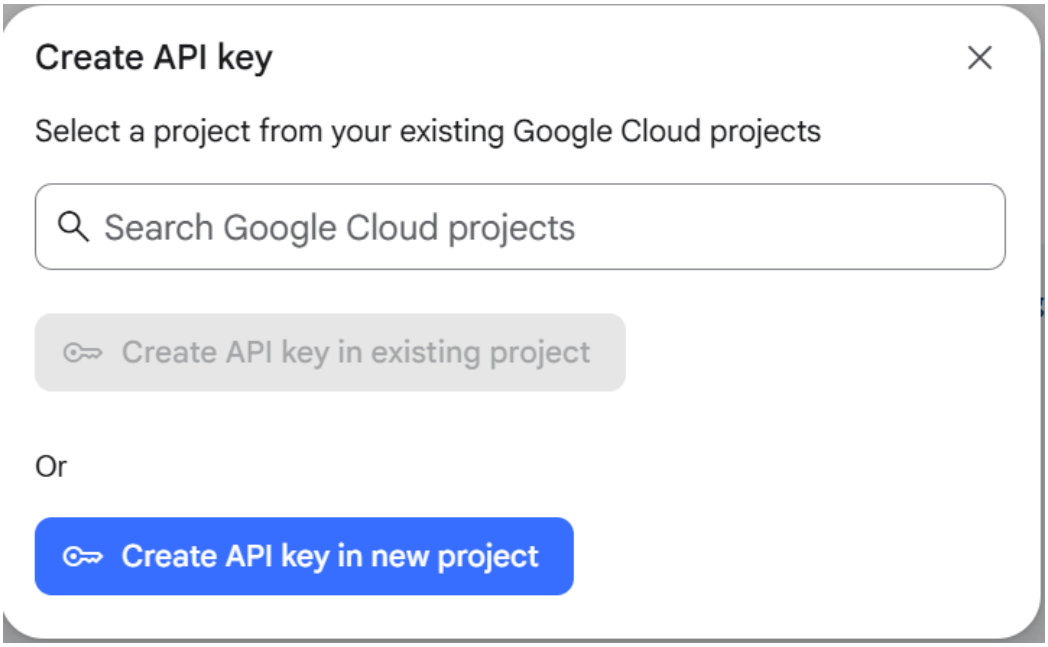
Instead of OpenAI, I used **Google AI Studio** as my browser agent's AI model. This is because Google AI Studio has a generous free tier. I could make 1,500 requests per day, which means I could ask my agent to do something 1,500 times a day.

- Go to https://aistudio.google.com/app/apikey.
- Log in with my Google account.
- Selected **Create API Key**.

If we've never used Google AI Studio before, we'll need to consent to the terms of service. Check the first box and select **I accept**.



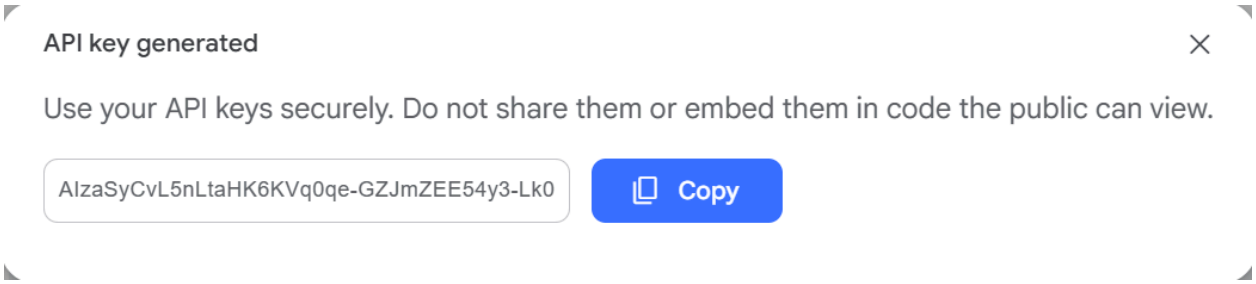
AI Studio asked me to select a project. What should I do?



Everything we create in AI Studio needs to be in a project for billing and organization.

- If there's an existing project in our organization or account, we can select it instead of creating a new one from scratch.
- It's safe to select an existing project in the dropdown - we won't be exposing other resources in the project. It's just that any charges will go towards that project's budget.
- Since what we're doing is free, our Google project won't be charged for anything.

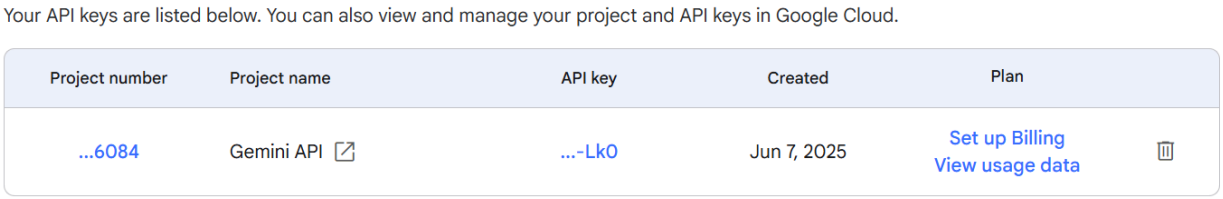
Selected **Copy** next to the generated API key.This is like a **password**, so don't share it with anyone!



Why do we need an API key?

An API key is like an access pass to the AI models in Google AI Studio. We'll need to pass it to WebUI, so that WebUI can use it to connect to an AI model.

Once WebUI is all connected, the AI model will be the brain that plans out how our agent will complete the task we've given in the Task Description. The AI model will also be the one that makes decisions about what to do next e.g. click a button, type in a form, navigate to a URL, complete the task and return the result.

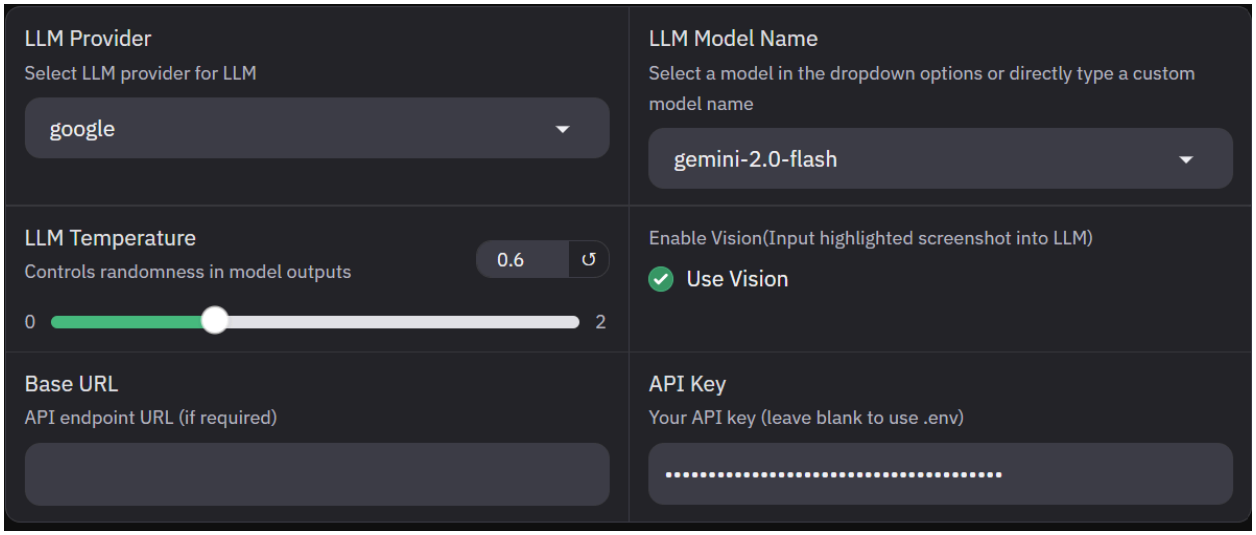


Devi Suhitha Chundru

Configured LLM Settings

Now that I had my API key, let's connect WebUI to the AI model:

- Headed back to the **WebUI** browser tab.
- Selected the **LLM Configuration** tab.



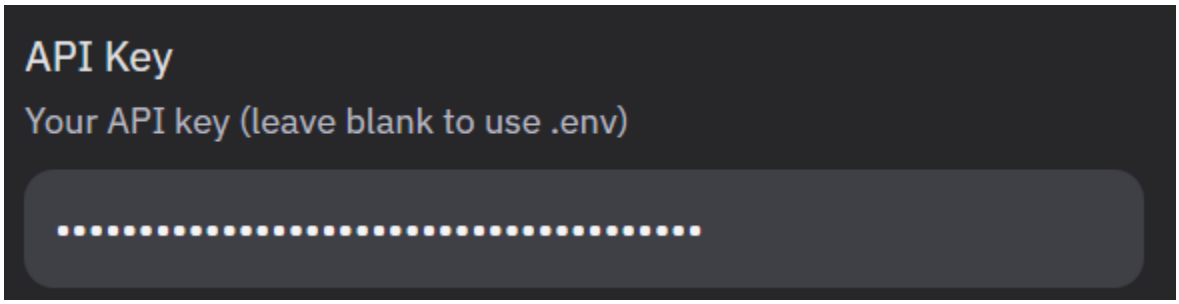
Under **LLM Provider**, I selected **Gemini**. I noted that **Google AI Studio** is the provider, but **Gemini** is the actual AI model I was using.

I kept the default **Model Name** that appeared. This is the specific version of the Gemini AI model I was using, and the default selection was free.

LLM Provider: google → ✅ This **is correct**. Google is the provider; Gemini is the model.

LLM Model Name: gemini-2.0-flash → ✅ I'm using Gemini!

- Pasted the API key I've copied into the **API Key** field.



Devi Suhitha Chundru

Step 4: Ran My First Agent Task

Let's watch my AI agent finish its first task.

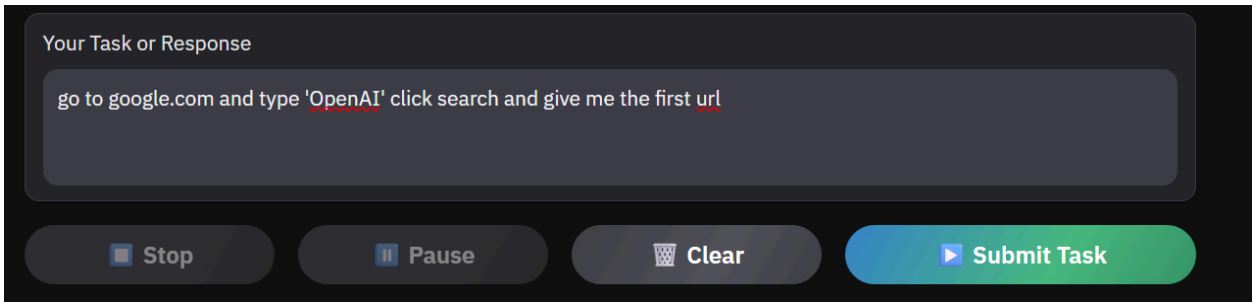
In this step, I:

- Run the agent and observe its actions.

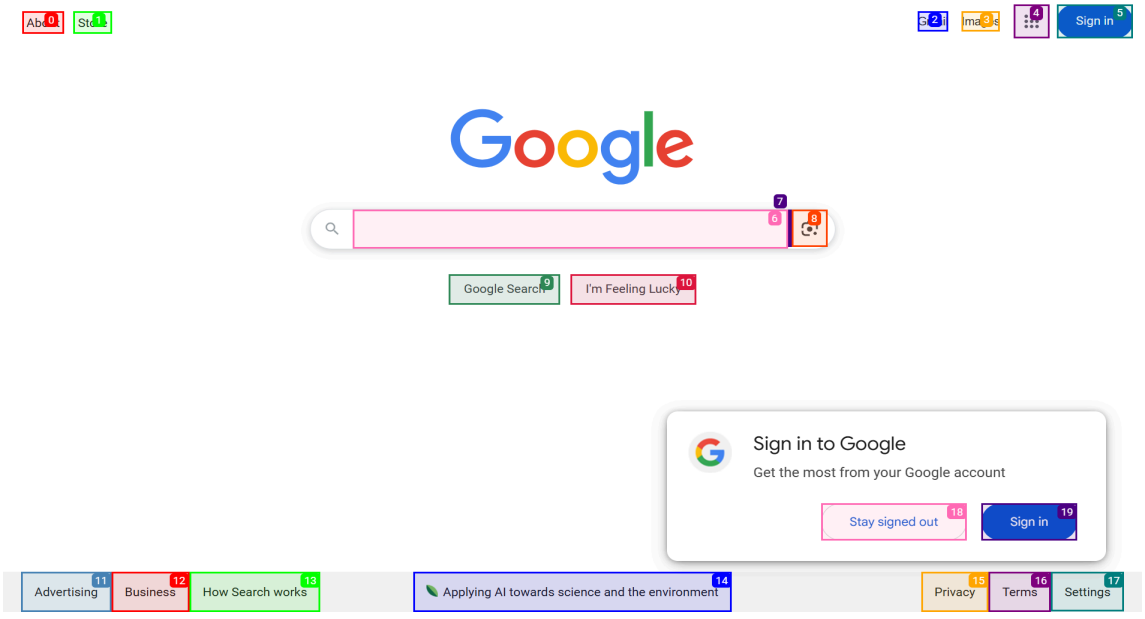
Ran WebUI Again

- Go back to the **Run Agent** tab.
- Kept the default prompt in the **Task Description:**

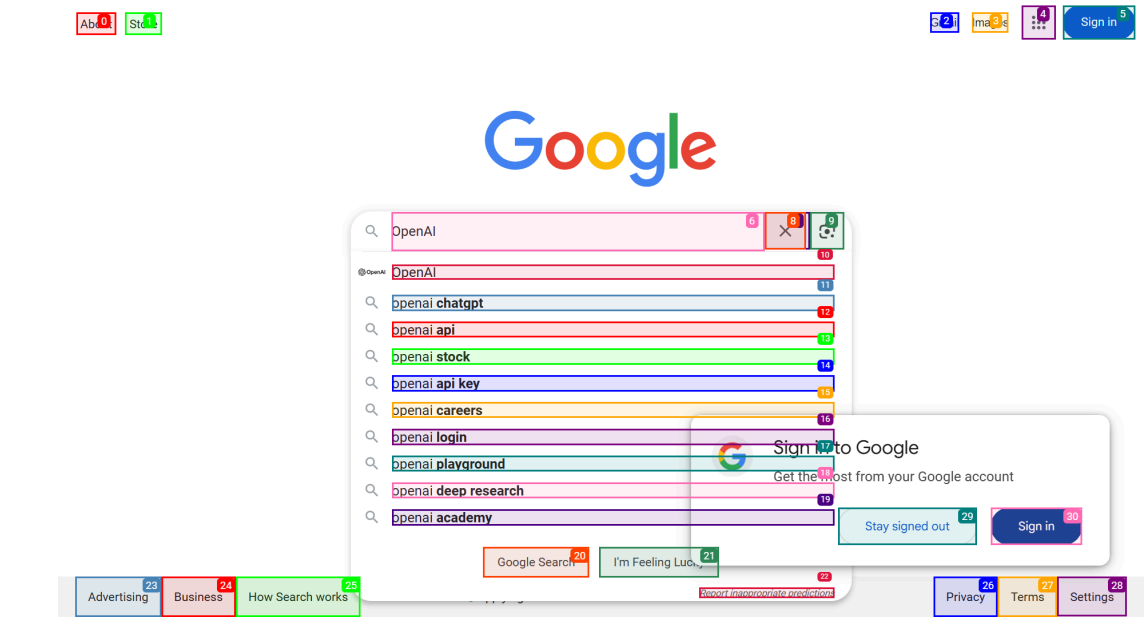
go to google.com and type 'OpenAI' click search and give me the first url

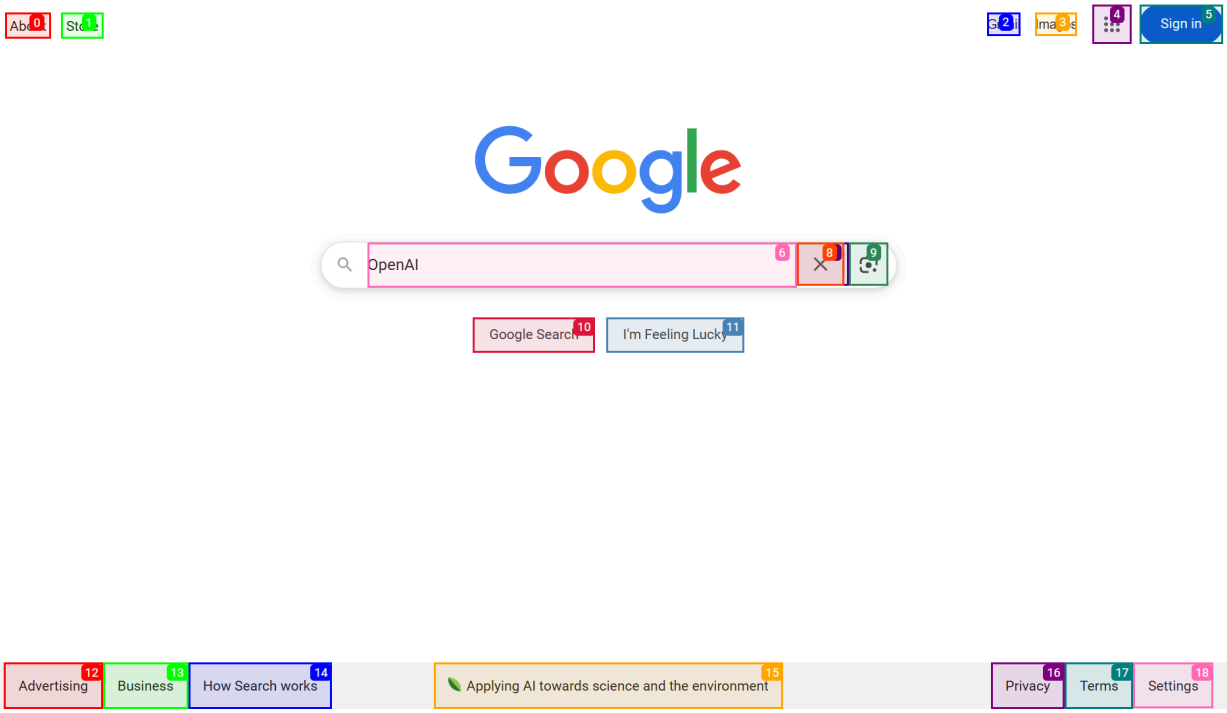


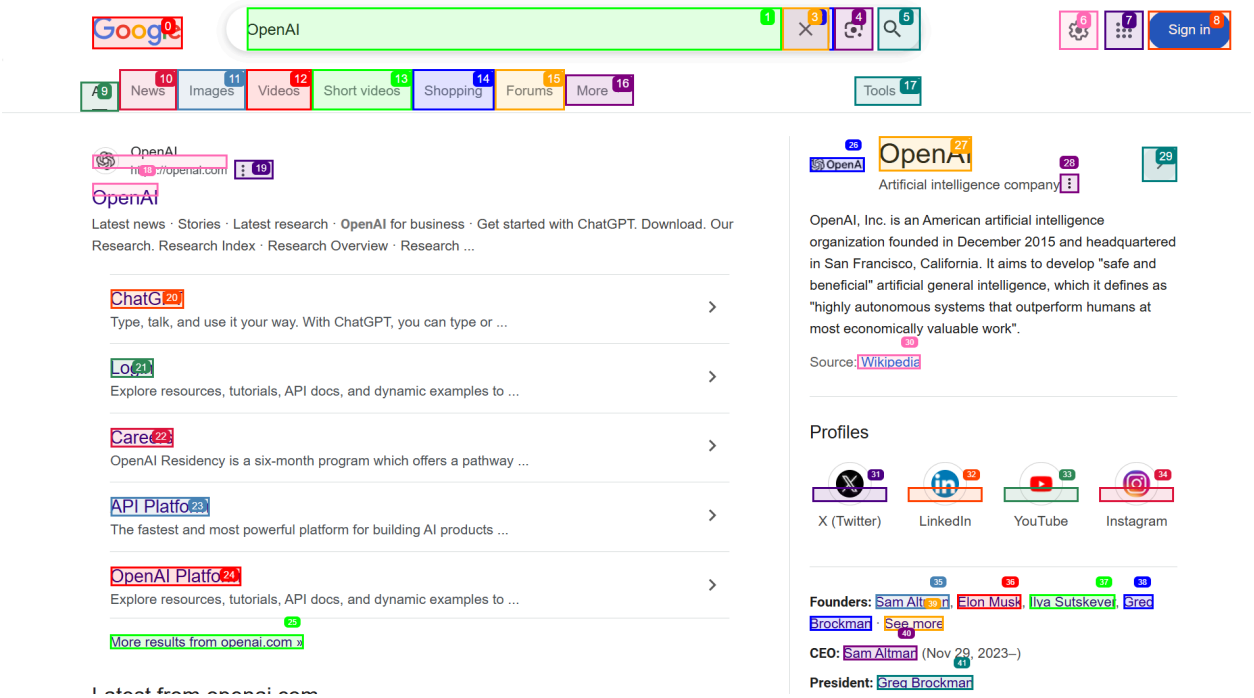
Woah! A new browser window opens, and the agent starts navigating to Google.com to search for "OpenAI". All without me having to do anything!



Devi Suhitha Chundru







Why are there coloured boxes around buttons and links?

The coloured boxes are a visual indicator that Playwright (our browser automation tool) is highlighting elements it can interact with. This helps us see what the agent is doing in real-time.

Behind the scenes, the agent is using the Gemini AI model to analyze each element Playwright finds, deciding which ones it should interact with to reach its end goal.

- Once my agent finds the first url result for "OpenAI" on Google, the new browser window collapses automatically.

What was the browser the agent was using?

The agent was using Chromium, which is the default browser of choice in WebUI.

Chromium is both a browser and the most popular browser **engine** - in fact, it's the engine that powers Google Chrome!

What is a browser engine?
A browser engine is the core software that:

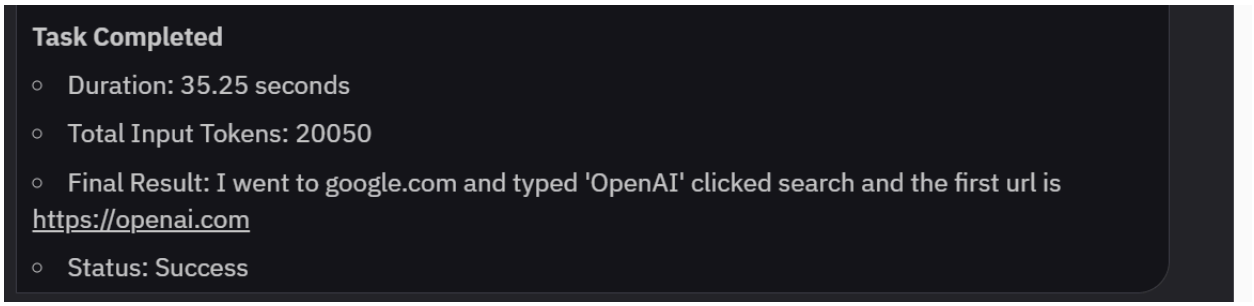
- Reads and understands webpage code (HTML, CSS, JavaScript)
- Turns that code into the visual pages you see
- Handles interactions like clicking and scrolling

Think of it as the "brain" of the browser. Chrome, Edge, and Brave all use Chromium as their engine, which is why they work similarly.

See the Results

- I headed back into the WebUI tab in my browser.
- I selected the **Run Agent tab.**
- I scrolled down and saw the results of my agent's actions - the first URL it found from searching "OpenAI" on google.com.

The exact URL might be different from the example, since Google's top result for "OpenAI" can change over time.



Devi Suhitha Chundru

Step 5: Advanced Prompt Writing for My Agent

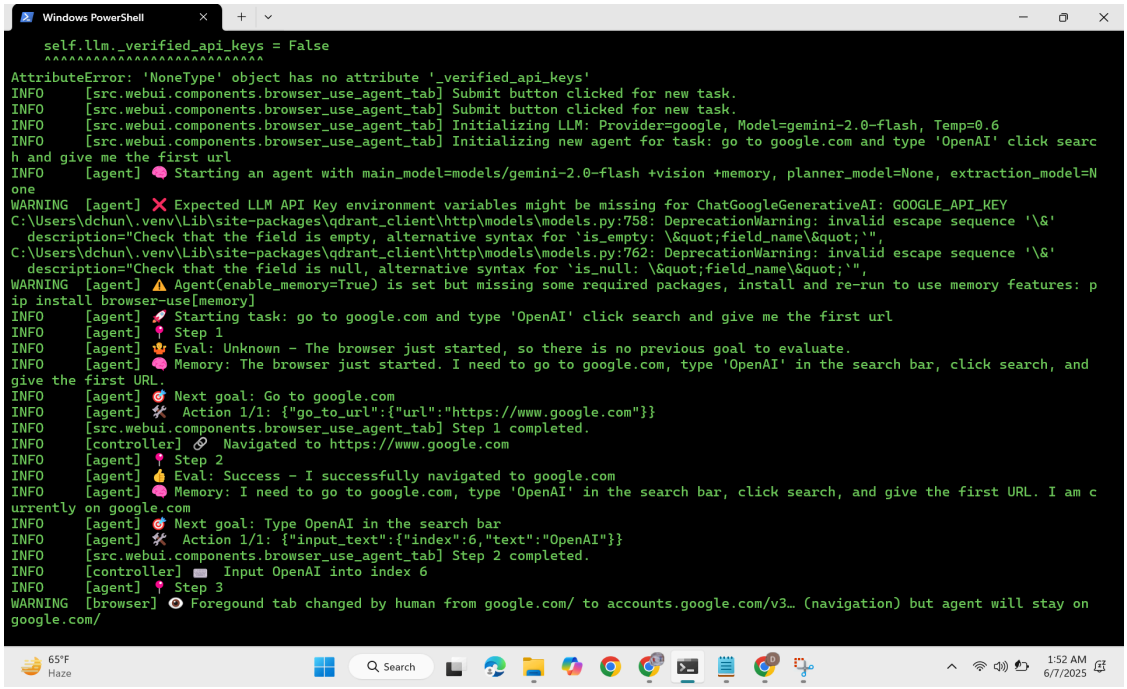
Writing **clear, specific** prompts is key to getting the results we want from your AI agent. Now that we've tried some basic prompts, let's learn how to write better ones and handle common errors.

In this step, I:

- Tried more complex prompts.
- Learn how to troubleshoot common issues.

Tried More Complex Prompts

- Head back to the **Run Agent** tab.
- Clear the **Task Description** field.
- Enter the following prompt:
- Open my Terminal.
- Wow! I might find that the Terminal has built up logs for the WebUI agent's actions.



Devi Suhitha Chundru

Got My Agent to Use LinkedIn as Me!

My mission, should I choose to accept it, is to get the agent to **use my website logins** for the ultimate browser automation - imagine an agent that can save, apply, download, and search on websites as if it was me!

In this secret mission, I :

- Got my agent to use websites using my personal login.
- Demonstrate my advanced **agent configuration** skills!

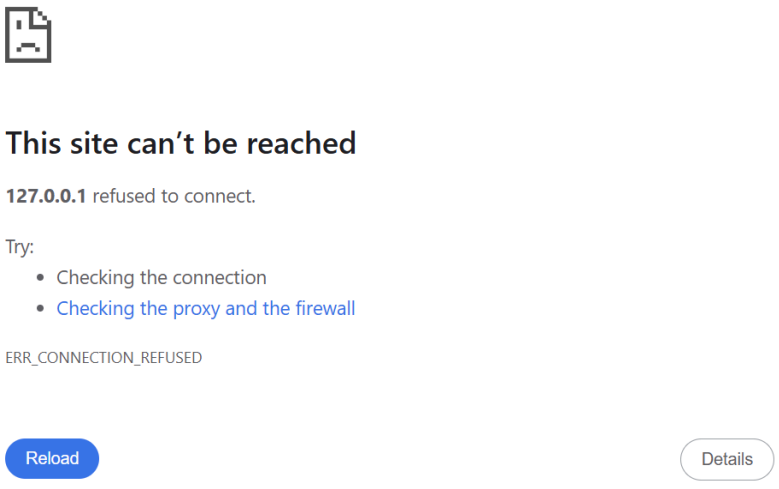
By default, WebUI uses a new browser profile i.e. Chromium. But, we can also tell the agent to use our ***own*** browser. This is a great option if we want our agent to access websites that require a login!

Edited the .env File

- Headed to my terminal.
- Exit the running WebUI process by pressing **Ctrl+C** on my keyboard.

```
Keyboard interruption in main thread... closing server.
(dchun) PS C:\Users\dchun\Documents\web-ui>
```

Reloaded the WebUI tab in my browser. I noticed that it no longer loads, which confirms I had stopped running WebUI.



- Opened my .env file by running the following command in my terminal: **Windows**

Devi Suhitha Chundru

notepad .env

```
(dchun) PS C:\Users\dchun\Documents\web-ui> notepad .env
(dchun) PS C:\Users\dchun\Documents\web-ui>
```

What is the .env file?

The .env file is a configuration file that stores environment variables for my project. In my case, it stores settings and tells WebUI how to run e.g. which AI model it should use.

Why are we making a copy?

We copy from .env.example to create our own .env file because:

- Each developer might need different settings (like different API keys or paths)
- The example file contains a template of all the settings we need
- We don't want to modify the example file directly since it's part of the project's template
- It's a common security practice to keep sensitive information (like API keys) in your own .env file that isn't shared with others

File Edit View

```
OPENAI_ENDPOINT=https://api.openai.com/v1
OPENAI_API_KEY=

ANTHROPIC_API_KEY=
ANTHROPIC_ENDPOINT=https://api.anthropic.com

GOOGLE_API_KEY=

AZURE_OPENAI_ENDPOINT=
AZURE_OPENAI_API_KEY=
AZURE_OPENAI_API_VERSION=2025-01-01-preview

DEEPSEEK_ENDPOINT=https://api.deepseek.com
DEEPSEEK_API_KEY=

MISTRAL_API_KEY=
MISTRAL_ENDPOINT=https://api.mistral.ai/v1

OLLAMA_ENDPOINT=http://localhost:11434

ALIBABA_ENDPOINT=https://dashscope.aliyuncs.com/compatible-mode/v1
ALIBABA_API_KEY=

MOONSHOT_ENDPOINT=https://api.moonshot.cn/v1
MOONSHOT_API_KEY=

UNBOUND_ENDPOINT=https://api.getunbound.ai
UNBOUND_API_KEY=

SiliconFLOW_ENDPOINT=https://api.siliconflow.cn/v1/
SiliconFLOW_API_KEY=

IBM_ENDPOINT=https://us-south.ml.cloud.ibm.com
IBM_API_KEY=
IBM_PROJECT_ID=

GROK_ENDPOINT="https://api.x.ai/v1"
GROK_API_KEY=
```

What's my username?

Run the following command in your terminal to find your username: **whoami**

```
(dchun) PS C:\Users\dchun\Documents\web-ui> whoami
suhith39\dchun
```

Access the WebUI: Open your web browser and navigate to http://127.0.0.1:7788.

Using Your Own Browser(Optional):

Set BROWSER_PATH to the executable path of your browser and BROWSER_USER_DATA to the user data directory of your browser. Leave BROWSER_USER_DATA empty if you want to use local user data.

Windows

BROWSER_PATH="C:\Program Files\Google\Chrome\Application\chrome.exe"

BROWSER_USER_DATA="C:\Users\YourUsername\AppData\Local\Google\Chrome\User Data"

Note: Replace **YourUsername** with your actual Windows username for Windows systems.

Can I use a different browser?

Unfortunately, WebUI only supports **Chrome** and **Chromium** browsers. If we want to use a different browser, we'll have to use Browser Use and set things up using code instead.

Check: have I replaced YOUR_USERNAME with my actual username?

Restarted WebUI

- Headed back to my terminal, launched WebUI again, I ran the following command to restart WebUI with my new **.env** settings:

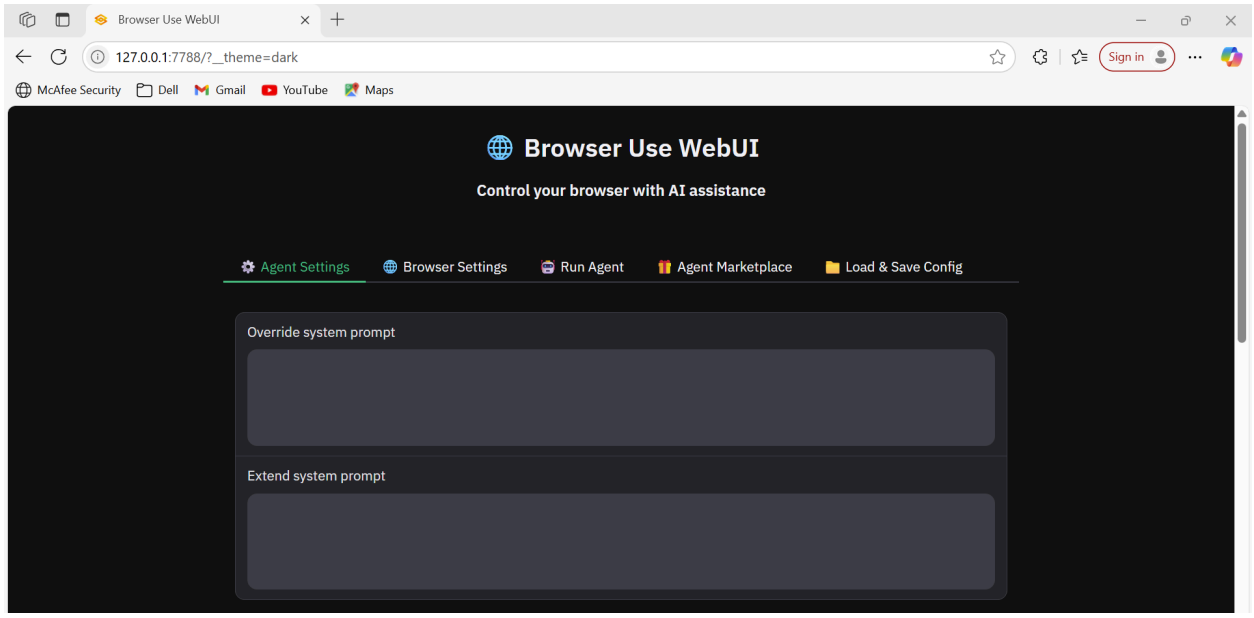
python webui.py --ip 127.0.0.1 --port 7788

```
(web-ui) PS C:\Users\dchun\Documents\web-ui> notepad .env
(web-ui) PS C:\Users\dchun\Documents\web-ui> python webui.py --ip 127.0.0.1 --port 7788
* Running on local URL:  http://127.0.0.1:7788

To create a public link, set `share=True` in `launch()`.
```

Devi Suhitha Chundru

This is my own browser(Microsoft Edge)



Set up WebUI

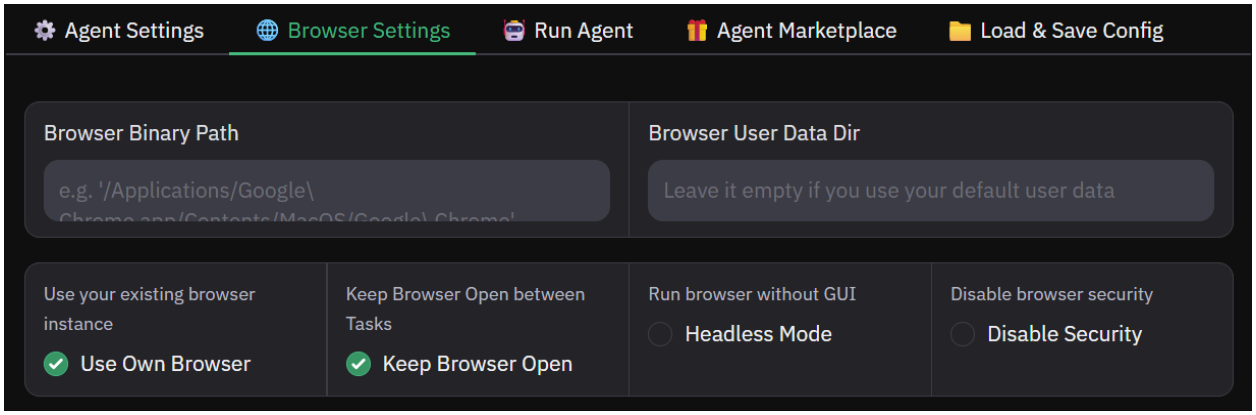
- Opened WebUI tab in my new browser i.e. http://127.0.0.1:7788
- Ishould see the WebUI page loaded again.
- Head to the **LLM Configuration** tab.
- Under **LLM Provider**, select **Gemini(Google for me)**
- Keep whichever default **Model Name** appears.
- Use the same API key you've used previously in the **API Key** field.

I've lost my API key!

No problem! We can always revisit the Google AI Studio dashboard, and select the key to see the full API Key again.

Go to the **Browser Settings** tab.

- Enabled the **Use Own Browser** setting. This tells WebUI to use my personal Chrome browser instead of launching Chromium.



Ran a Prompt

- Headed back to the **Run Agent** tab.
- Enter the following **Task Description:**

Go to LinkedIn and log in.

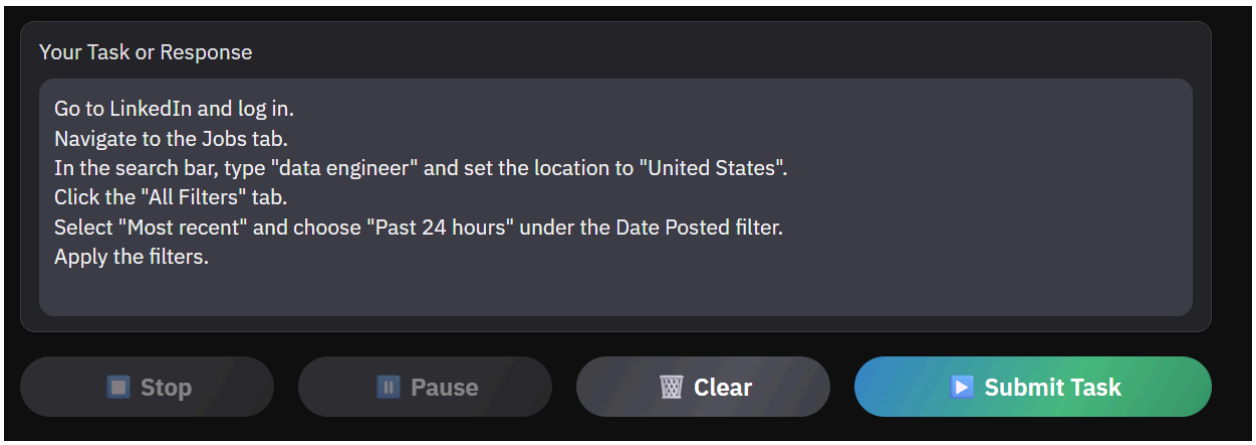
Navigate to the Jobs tab.

In the search bar, type "data engineer" and set the location to "United States".

Click the "All Filters" tab.

Select "Most recent" and choose "Past 24 hours" under the Date Posted filter.

Apply the filters.



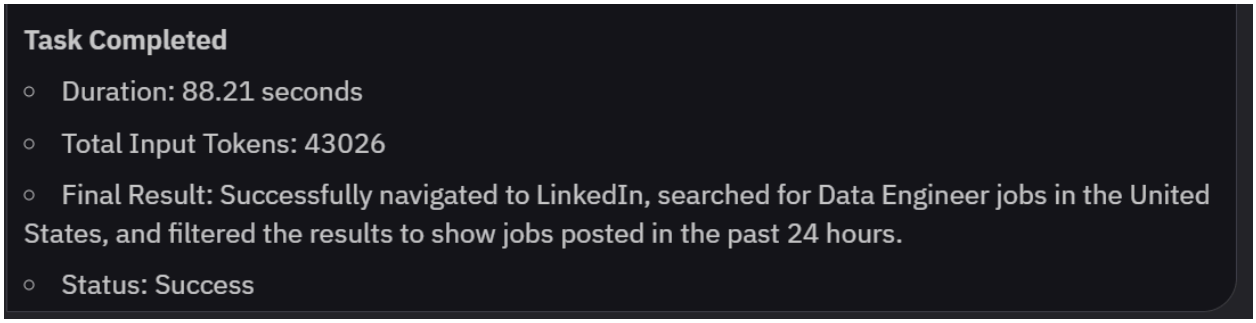
Devi Suhitha Chundru

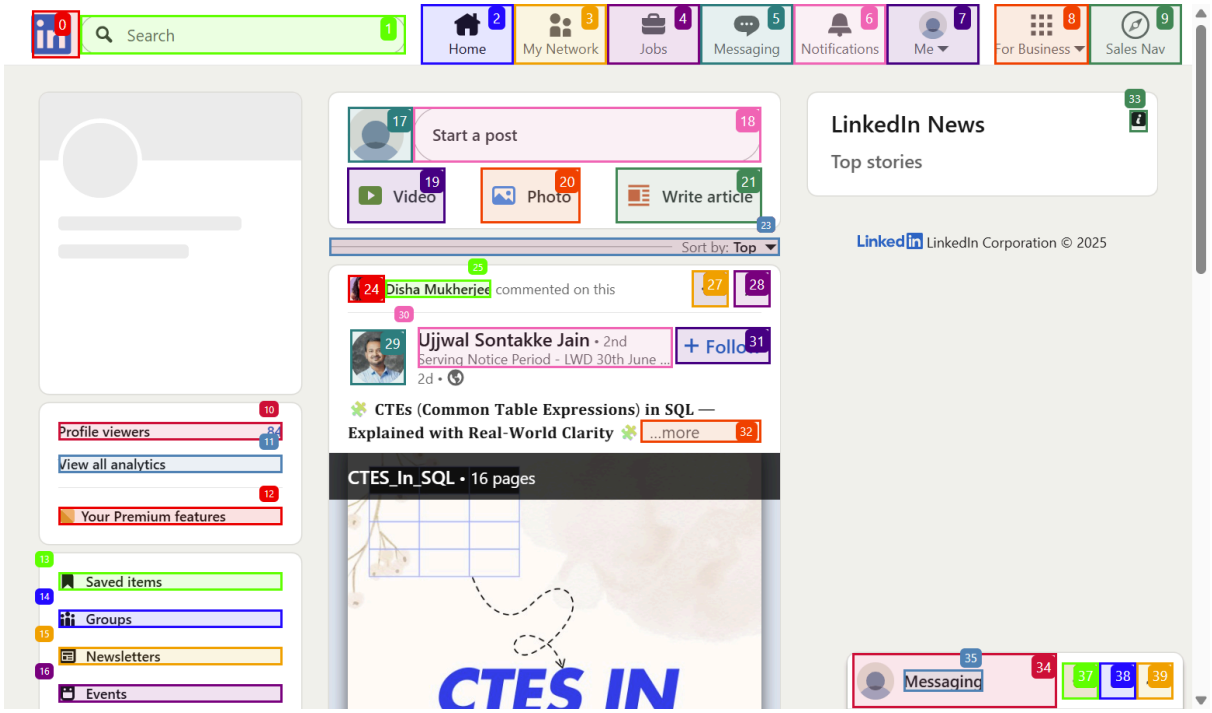
Why this prompt?

This prompt will show us the power of using our own browser:

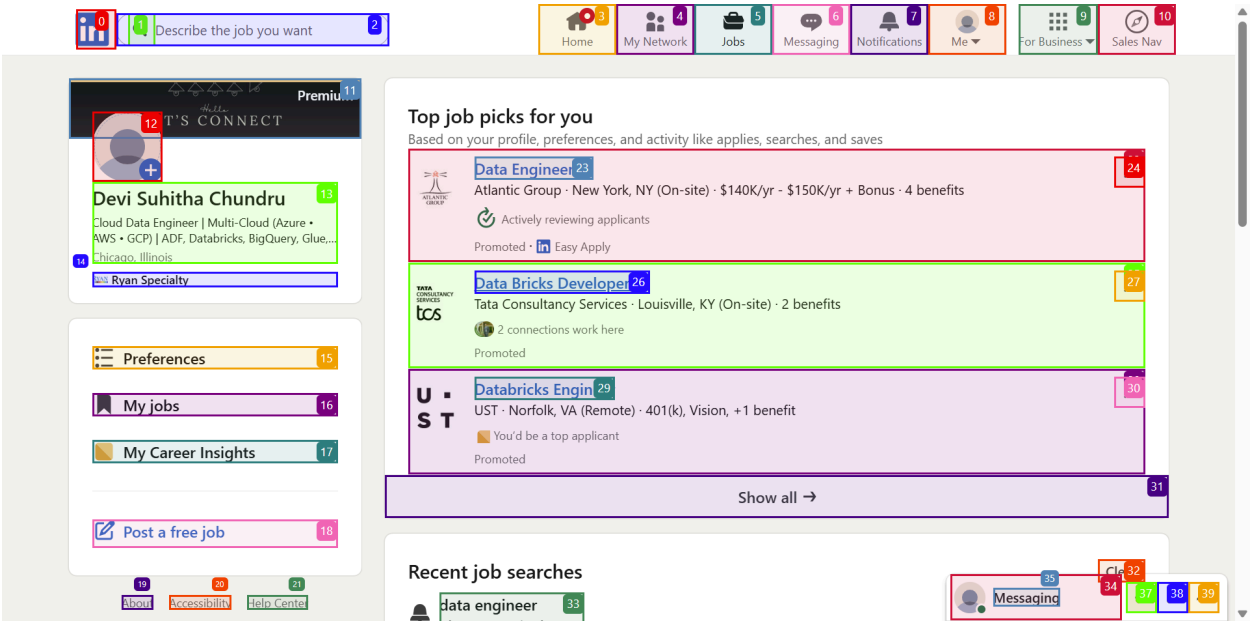
- This prompt demonstrates the power of using our own browser profile for job search automation:
- LinkedIn requires users to be logged in to access and apply job filters or view detailed job listings.
- If we used a fresh Chromium session, the agent would encounter a login wall.
- But by using our personal Chrome profile, the agent can tap into our existing login session, cookies, and saved settings."

This prompt showcases a key advantage of using our own browser: the agent can behave like a real user and carry out actions that would otherwise fail in a default, sandboxed environment.

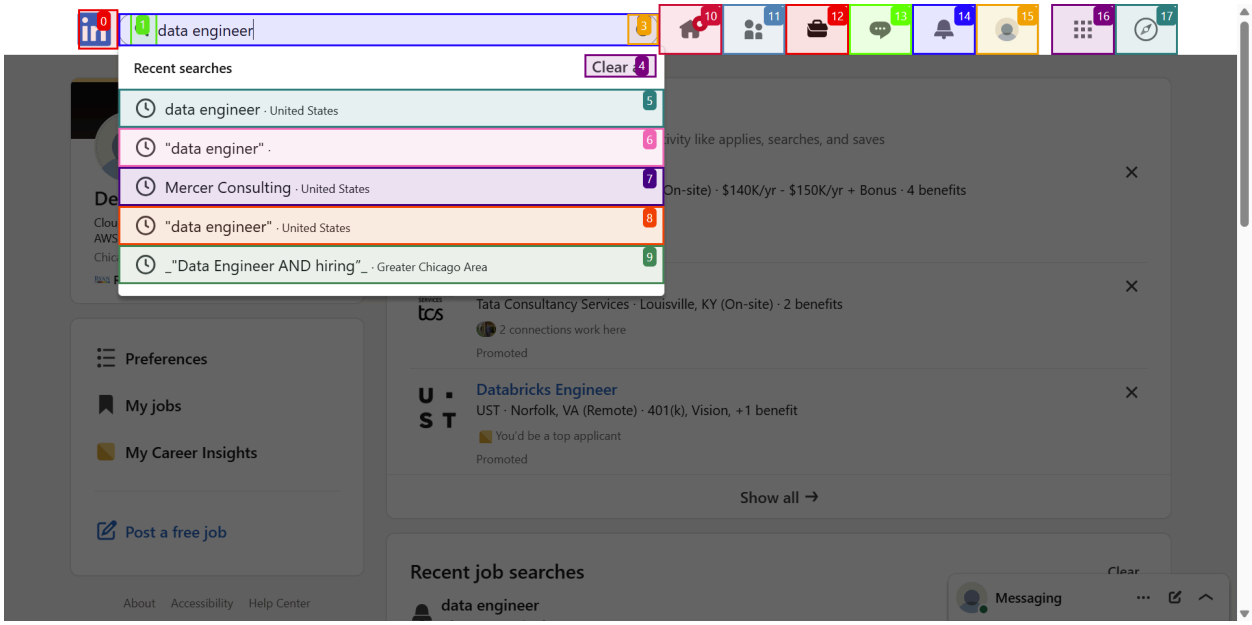


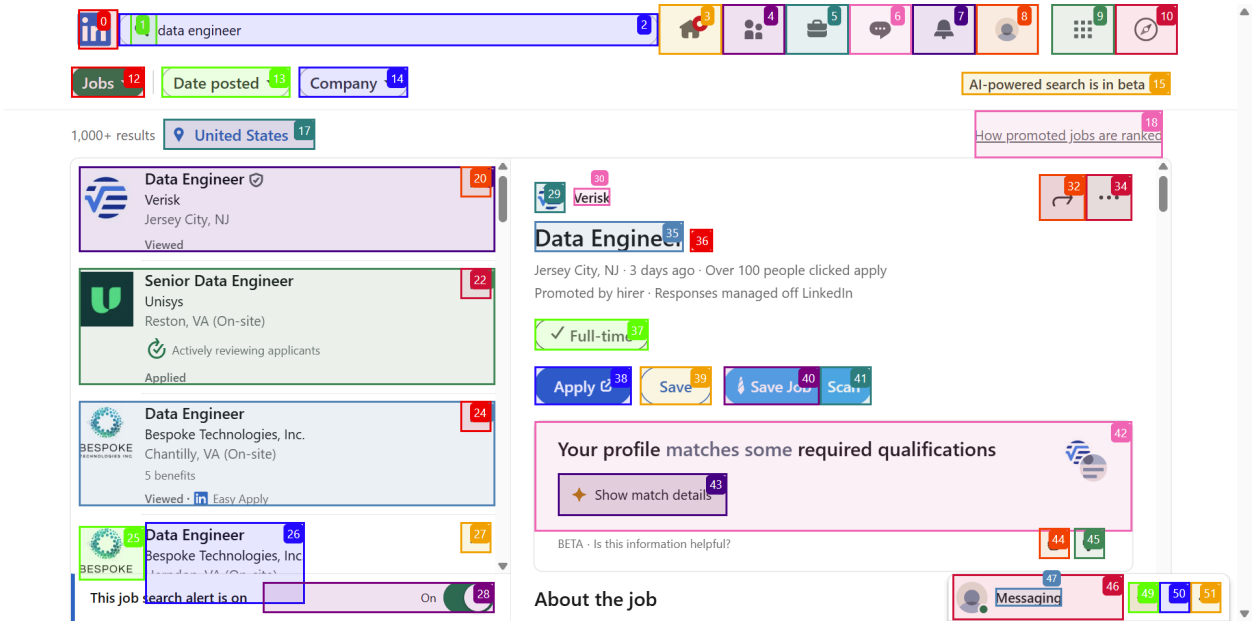


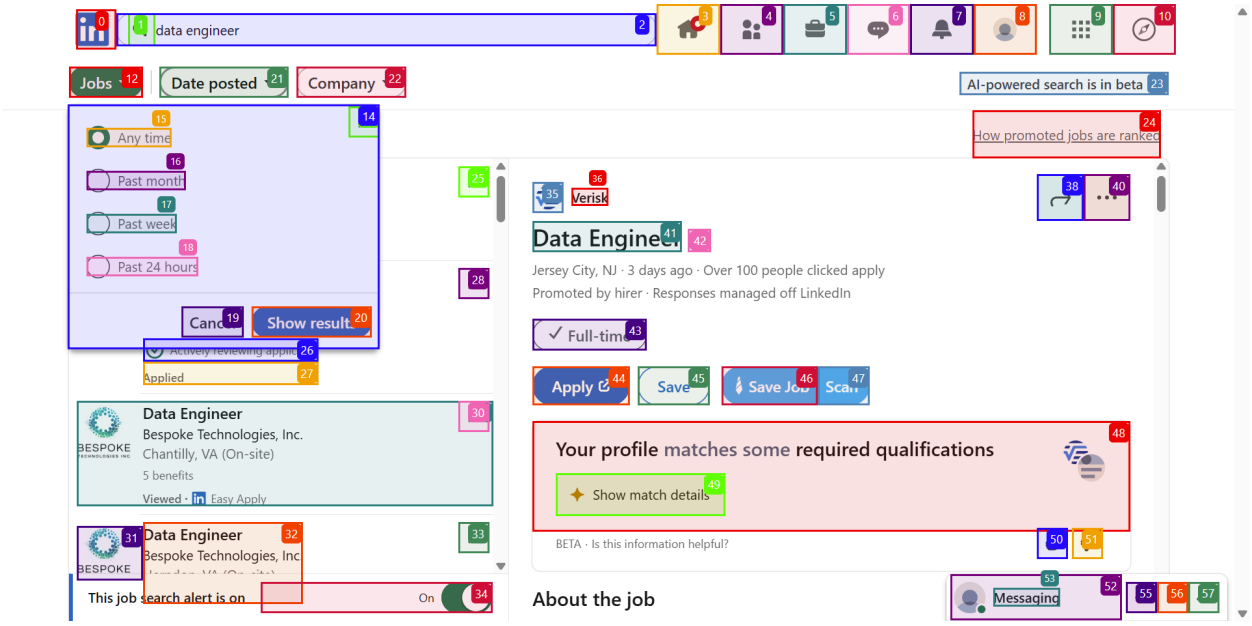

Search
Home
My Network
Jobs
Messaging
Notifications
Me
For Business
Sales Nav
Hello
LET'S CONNECT
Profile language
English
Public profile & URL
www.linkedin.com/in/devisuhithachundru
Devi Suhitha Chundru
She/Her
Add verification badge
Cloud Data Engineer | Multi-Cloud (Azure • AWS • GCP) | ADF, Databricks, BigQuery, Glue, Airflow, Kafka | PySpark, SQL, CI/CD | Batch & Streaming Pipelines | Data Modeling, Governance, APIs | Data Lakehouse |
Chicago, Illinois, United States · Contact info
422 connections
Open to
Add profile section
Add custom button
Resources
Open to work
Data Engineer, AWS Engineer, Cloud Engineer and Azure Engineer roles
Show details
Promoted
Devi Suhitha, explore relevant opportunities with FCSLA Life Insurance
Get the latest jobs and industry news
Follow
Who your viewers also viewed
Private to you
Messaging

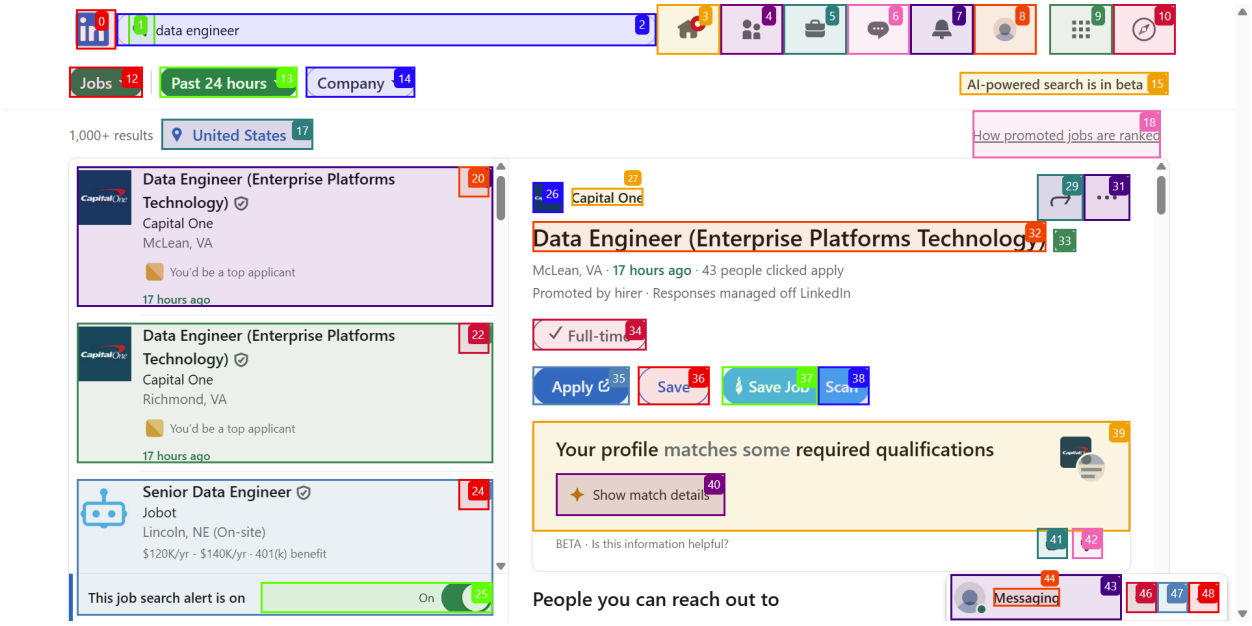

Describe the job you want
Home
My Network
Jobs
Messaging
Notifications
Me
For Business
Sales Nav
Devi Suhitha Chundru
Cloud Data Engineer | Multi-Cloud (Azure • AWS • GCP) | ADF, Databricks, BigQuery, Glue,...
Chicago, Illinois
Ryan Specialty
Preferences
My jobs
My Career Insights
Post a free job
About
Accessibility
Help Center
Top job picks for you
Based on your profile, preferences, and activity like applies, searches, and saves
Data Engineer
Atlantic Group · New York, NY (On-site) · $140K/yr - $150K/yr + Bonus · 4 benefits
Actively reviewing applicants
Promoted · Easy Apply
Data Bricks Developer
Tata Consultancy Services · Louisville, KY (On-site) · 2 benefits
2 connections work here
Promoted
Databricks Engin
UST · Norfolk, VA (Remote) · 401(k), Vision, +1 benefit
You'd be a top applicant
Promoted
Show all →
Recent job searches
data engineer
Messaging

Devi Suhitha Chundru









I'm not logged into LinkedIn in my Chrome browser!

If we're not logged into LinkedIn in our Chrome browser, our agent will be stuck! Feel free to change this prompt to something else that our agent has login access to.

For example, we can try this prompt about sending an email (replace gmail with your email provider):

Go to gmail and send a new email to ..... . The email header should say "Hello! I'm doing the NextWork project on WebUI". Find a cat meme, and attach the image to the body of the email.

Or, you can try this prompt about writing in a document (replace Google Docs with your document provider):

- Open Google Docs.
- Create a new document.
- Insert three motivational quotes.
- Save the document.
- Print it as a PDF.

Click the Run Agent button.

Look at that! We should see the agent performing the task in our own browser window this time.

🤦‍♀️ I ran into a runtime error!

If we ran into a runtime error, it's likely because we didn't close all our browser windows. Try closing all our browser windows, and running the prompt again.

If we have multiple profiles under Google Chrome, we might need to select the profile we want to use first and close the window (do not quit Chrome). It'd also help to deselect the Show on Startup checkbox in your Chrome settings.

Key Concepts Learned

- Writing effective prompts for AI agents
- Automating browser interaction with WebUI
- Handling user login flows and dynamic pages
- Structuring modular AI agent scripts
- Integrating AI with real-world use cases

Why This Project Matters

This project showcases how AI can assist in everyday tasks and provides a practical use case for personal productivity, testing automation, and data collection. It's also a foundational step for building more advanced agents in areas like RPA (Robotic Process Automation) and AI assistants.

Devi Suhitha Chundru

Common Errors & Fixes

1. Chrome already running

Error:

Failed to start: Chrome is already running.

Fix:

Close all Chrome windows (even those minimized or running in background).

Also check Task Manager → End any chrome.exe still running.

2. Permission Denied for Data Directory

Error:

Cannot access profile directory or permission denied.

Fix:

Make sure C:\tmp\chrome-debug exists and is writeable.

You can pre-create the folder:

powershell

mkdir "C:\tmp\chrome-debug"

Or use --no-first-run --no-default-browser-check to skip Chrome setup prompts:

"C:\Program Files\Google\Chrome\Application\chrome.exe" --remote-debugging-port=9222 --user-data-dir="C:\tmp\chrome-debug"

Devi Suhitha Chundru