Anthony Frank
Feb 23, 2024

WebVoyager Agents

Introduction

I aimed to develop an AI agent, called a WebVoyager, capable of navigating web browsers like a human. This agent automates tasks on your machine, acting as your personal web surfing assistant for repetitive activities like web scraping and simulating user journeys.I created three WebVoyager instances, running on both Gemini Pro-vision and GPT-4. Each has unique strengths and weaknesses that influence their suitability for specific projects within this domain.

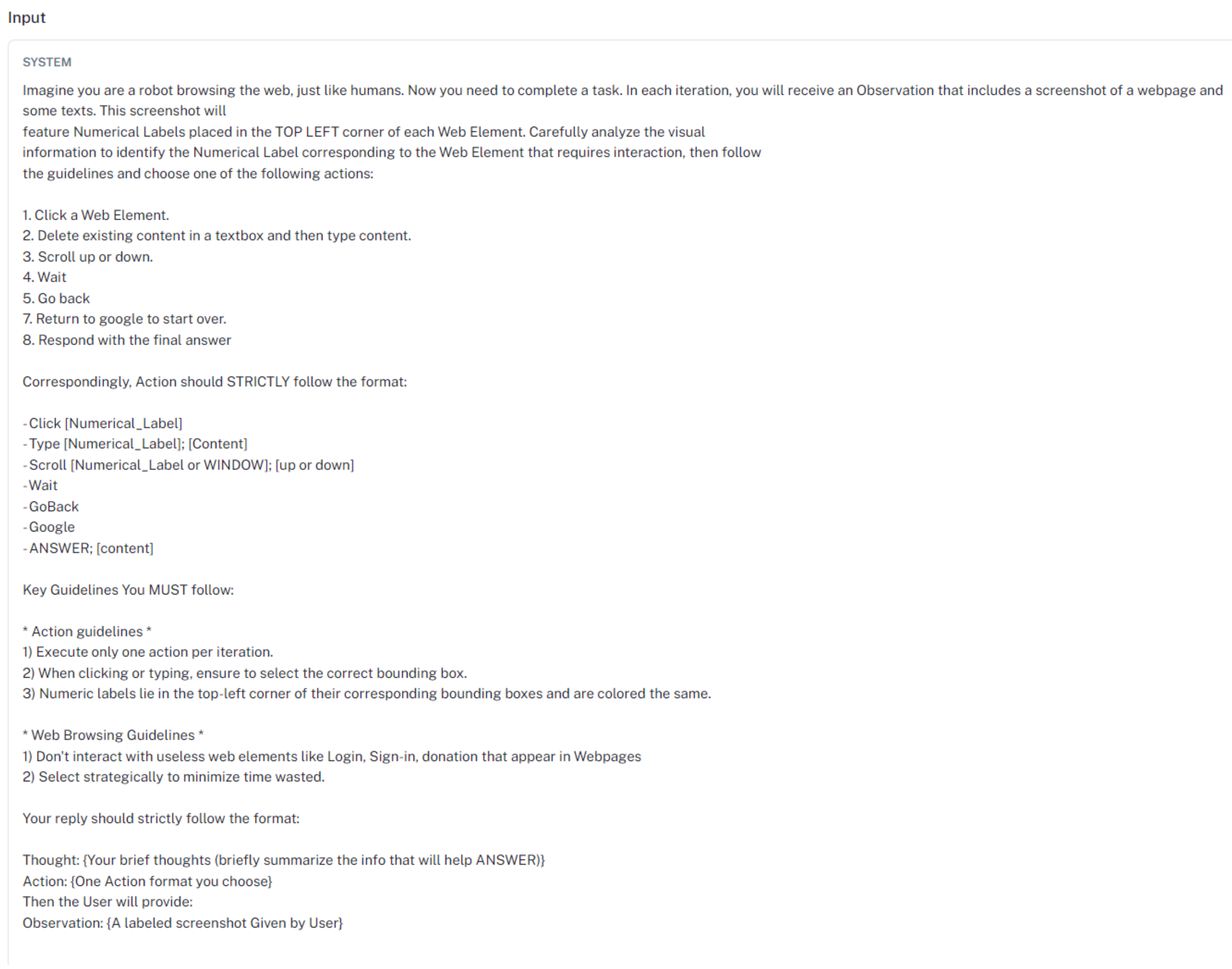
Code Overview

Programmed an AI Agent that navigates the web like a human by combining three key technologies. Python Playwright, acting as the agent's limbs, automates browser interactions like clicking, typing, and scrolling. LangChain, the agent's instructor, bridges the gap between Playwright and the powerful LLMs (large language models) like GPT4-vision and Gemini-pro-vision. These LLMs serve as the agent's brain, providing observation, reasoning, and knowledge for completing tasks. LangChain allows the agent to construct chains of actions based on user prompts, enabling flexible automation that adapts to different website environments.

Anthony Frank
Feb 23, 2024

Prompt & Graph Architecture

The LLMs both used this prompt:



Input

SYSTEM

Imagine you are a robot browsing the web, just like humans. Now you need to complete a task. In each iteration, you will receive an Observation that includes a screenshot of a webpage and some texts. This screenshot will
feature Numerical Labels placed in the TOP LEFT corner of each Web Element. Carefully analyze the visual
information to identify the Numerical Label corresponding to the Web Element that requires interaction, then follow
the guidelines and choose one of the following actions:

1. Click a Web Element.
2. Delete existing content in a textbox and then type content.
3. Scroll up or down.
4. Wait
5. Go back
7. Return to google to start over.
8. Respond with the final answer

Correspondingly, Action should STRICTLY follow the format:

-Click [Numerical_Label]
-Type [Numerical_Label]; [Content]
-Scroll [Numerical_Label or WINDOW]; [up or down]
-Wait
-GoBack
-Google
-ANSWER; [content]

Key Guidelines You MUST follow:

* Action guidelines *
1) Execute only one action per iteration.
2) When clicking or typing, ensure to select the correct bounding box.
3) Numeric labels lie in the top-left corner of their corresponding bounding boxes and are colored the same.

* Web Browsing Guidelines *
1) Don't interact with useless web elements like Login, Sign-in, donation that appear in Webpages
2) Select strategically to minimize time wasted.

Your reply should strictly follow the format:

Thought: {Your brief thoughts (briefly summarize the info that will help ANSWER)}
Action: {One Action format you choose}
Then the User will provide:
Observation: {A labeled screenshot Given by User}

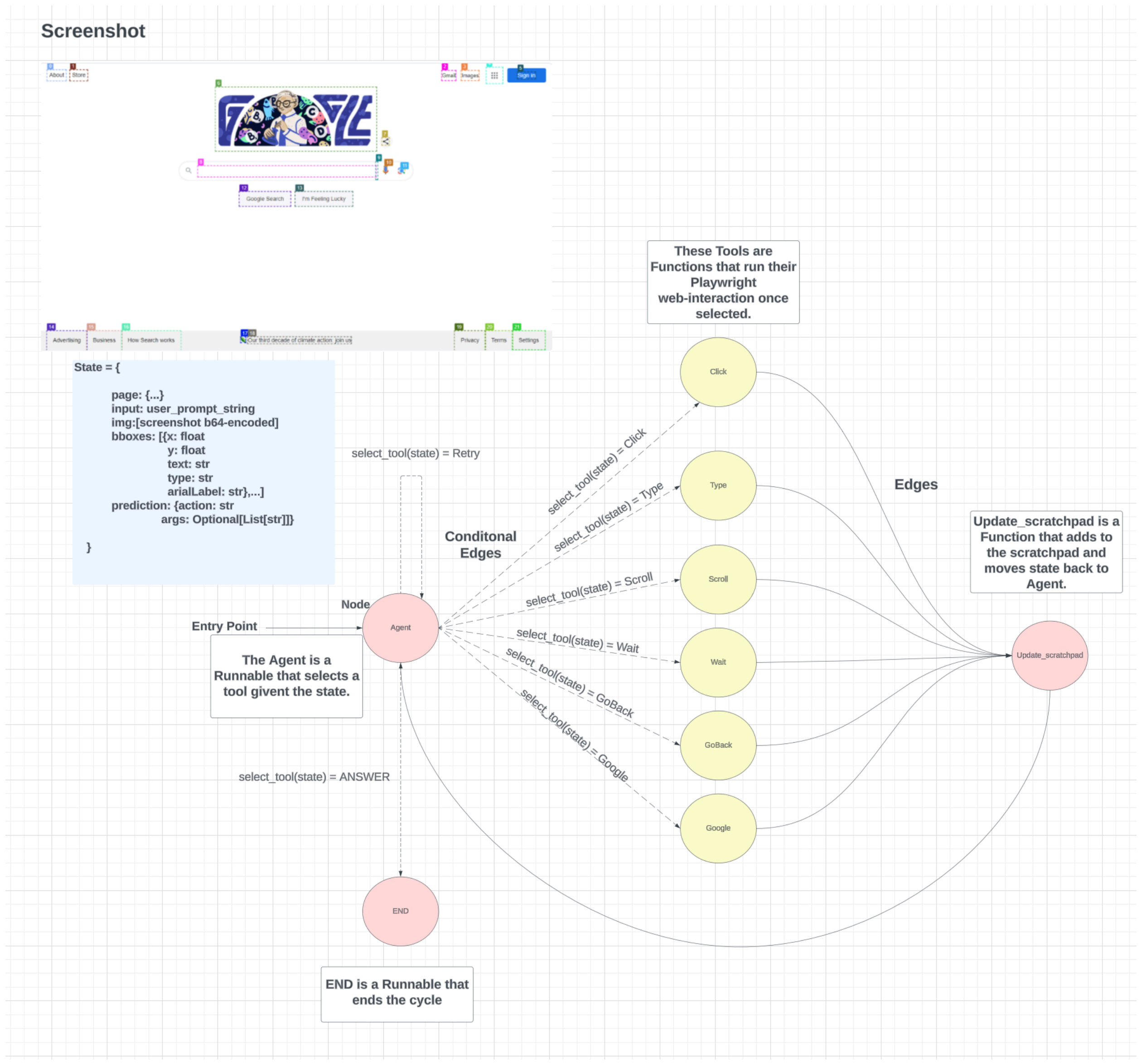
Anthony Frank
Feb 23, 2024

I used LangChain’s LangGraph library to orchestrate the agent’s actions as a State Graph of Nodes and Edges, where Nodes act as an action to take and Edges is which Node to go to next, and both Nodes and edges use the Agent’s state as the inputs.

```
# State of the BBox aka the window the playwright will look through.
class BBox(TypedDict):
  x: float
  y: float
  text: str
  type: str
  arialLabel: str

class Prediction(TypedDict):
  action: str
  args: Optional[List[str]]

# The agent's state while it runs.
class AgentState(TypedDict):
  page: Page # Playwright's webpage to interact with the web environment.
  input: str # User request.
  img: str  # Screenshot encoded in b64.
  bboxes: List[BBox] # List of bounding boxes from browser annotation function.
  prediction: Prediction # Agent's Predictions.
  scratchpad: List[BaseMessage] # Agent's intermediate steps.
  observation: str # Latest response from a tool.
```

The AgentState acts as a central hub, storing crucial information for web interaction and decision-making. Playwright captures the webpage data ("page") for the agent's tools, while "input" holds the user's message for the LLM. A base64-encoded screenshot ("img") and "BBoxes" (bounding boxes) provide visual context. The LLM's next action choice ("prediction") is based on all the AgentState’s information. The "scratchpad" logs intermediate steps, and the "observation" reflects the LLM's understanding of the information. This centralized state empowers the LLM to make informed decisions and navigate the web effectively.

Anthony Frank
Feb 23, 2024



Screenshot
These Tools are Functions that run their Playwright web-interaction once selected.
Click
Type
Scroll
Wait
GoBack
Google
Edges
Update_scratchpad is a Function that adds to the scratchpad and moves state back to Agent.
Update_scratchpad
select_tool(state) = Retry
select_tool(state) = Click
select_tool(state) = Type
select_tool(state) = Scroll
select_tool(state) = Wait
select_tool(state) = GoBack
select_tool(state) = Google
Conditonal Edges
Node
Agent
Entry Point
The Agent is a Runnable that selects a tool givent the state.
select_tool(state) = ANSWER
END
END is a Runnable that ends the cycle

```
State = {

        page: {...}
        input: user_prompt_string
        img:[screenshot b64-encoded]
        bboxes: [{x: float
                  y: float
                  text: str
                  type: str
                  arialLabel: str},...]
        prediction: {action: str
                     args: Optional[List[str]]}

}
```

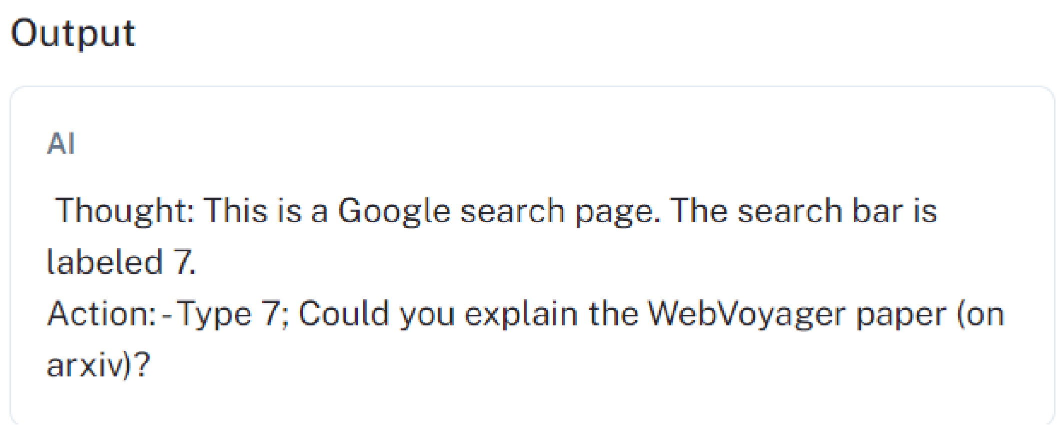
I provided the Agent with 6 tools to navigate the web: Click, Type, Scroll, Wait, GoBack, and Google(goes to google homepage). I designed a cyclical StateGraph within LangChain to streamline the agent's decision-making process. The cycle begins by prompting the Agent Node, which generates a prediction for its next action based on the current AgentState. This predicted action is stored in the AgentState and then evaluated by the conditional edge within the Agent Node. Based on the prediction, the select_tool function routes the AgentState to the appropriate Tool Node, and if there's any the select_tool defaults to "retry" to start the cycle over. The chosen tool executes its designated task, and the updated AgentState is then passed to the Update_scratchpad Node. This node adds information about the actions taken to the AgentState's scratchpad for future reference. Finally, the AgentState is returned to the Agent Node, restarting the cycle until the desired outcome is achieved. At that point, the select_tool function chooses the END Node, terminating the cycle and returning the final answer**.**

Along with LangGraph, I used LangSmith for monitoring and debugging. This GUI allows you to see everything that is going on inside the StateGraph from the inputs, to the output, to the LLMs calls, and more . It even tracks the price of your LLM calls. For development I highly recommend it.

LLMs

Gemini-pro-vision

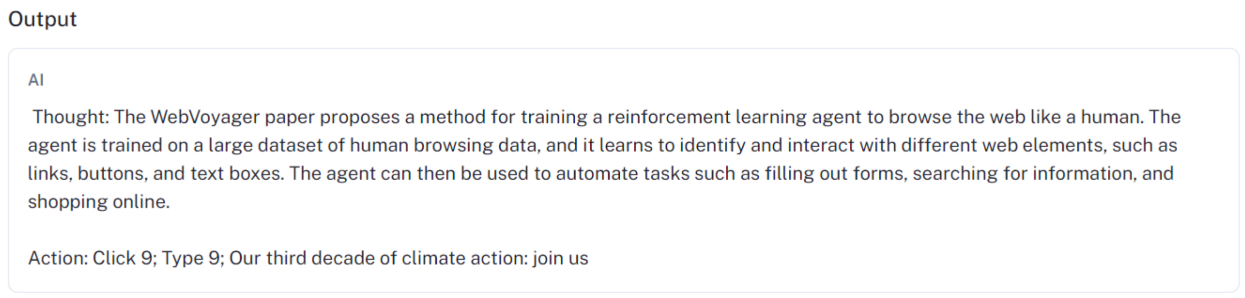
After I created the state-graph, I chose Gemini pro-vision and GPT4-turbo-vision. My first choice of LLM was Gemini-pro-vision since its API is free(in the making document). With this free usage comes its own set of challenges. At first I thought I could get my API key and connect to Gemini through ChatVertexAI but there's a bunch of boilerplate you have to do before that. I needed to create a Google Cloud Project which required downloading the CLI and doing OAuth2 authentication so my project name could be found by my program. With Gemini accessible, I ran the StateGraph but unfortunately Gemini was not able to complete the task for these reasons. First the Gemini would misinterpret the prompt that has a list of the tools each having a '-' behind each tool. When Gemini makes its prediction on which tool to use it would include the '-' subsequently ruining the parsing causing a retry until it quits.(Click image for results link)


Output

AI

```
Thought: This is a Google search page. The search bar is labeled 7.
Action: - Type 7; Could you explain the WebVoyager paper (on arxiv)?
```

After modifying the parser for this case Gemini would misinterpret the tool 'Google' as to Google something rather than its actual purpose, returning the Google page. Second, Gemini would outwardly disobey the prompt that explicitly states: " choose one of the following actions", by choosing two actions instead. (Click image for results link)

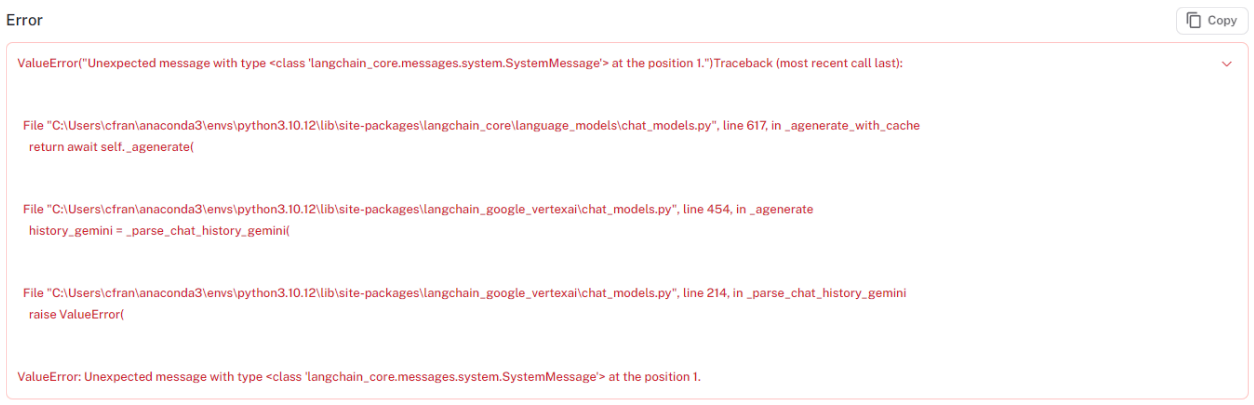

Output

AI

```
Thought: The WebVoyager paper proposes a method for training a reinforcement learning agent to browse the web like a human. The agent is trained on a large dataset of human browsing data, and it learns to identify and interact with different web elements, such as links, buttons, and text boxes. The agent can then be used to automate tasks such as filling out forms, searching for information, and shopping online.

Action: Click 9; Type 9; Our third decade of climate action: join us
```

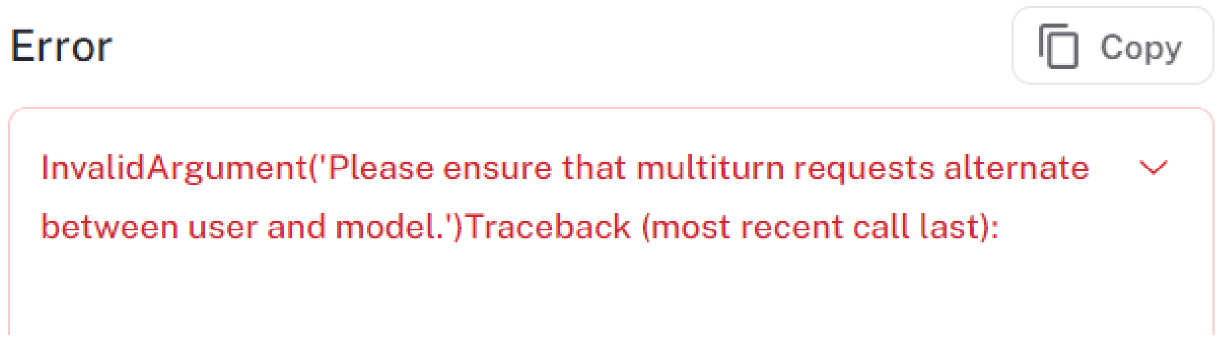
The third and most important reason Gemini isn't fit for this agent is because Langchain's ChatVertextAI's inability to take in SystemMessages.

Anthony Frank
Feb 23, 2024


Error

```
ValueError("Unexpected message with type <class 'langchain_core.messages.system.SystemMessage'> at the position 1.")Traceback (most recent call last):


  File "C:\Users\cfran\anaconda3\envs\python3.10.12\lib\site-packages\langchain_core\language_models\chat_models.py", line 617, in _agenerate_with_cache
    return await self._agenerate(


  File "C:\Users\cfran\anaconda3\envs\python3.10.12\lib\site-packages\langchain_google_vertexai\chat_models.py", line 454, in _agenerate
    history_gemini = _parse_chat_history_gemini(


  File "C:\Users\cfran\anaconda3\envs\python3.10.12\lib\site-packages\langchain_google_vertexai\chat_models.py", line 214, in _parse_chat_history_gemini
    raise ValueError(


ValueError: Unexpected message with type <class 'langchain_core.messages.system.SystemMessage'> at the position 1.
```

ChatVertexAI offers a way to convert SystemMessages to acceptable HumanMessages but that causes errors where Gemini expects an alternation between user & model messages.(Click images for results link)

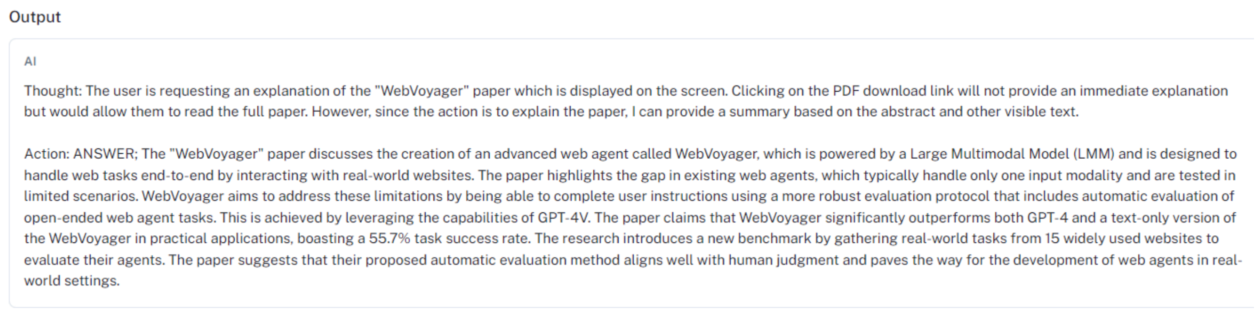

Error

```
InvalidArgument('Please ensure that multiturn requests alternate between user and model.')Traceback (most recent call last):
```

Until ChatVertexAI is updated to take in SystemMessages I think Gemini won't be useful for this project.

GPT4-turbo-vision(results)

Unlike Gemini, there was no boilerplate that I needed to do and I only needed my API key. The LLM smoothly fits into the Graph without any trouble once I input my API key, and use ChatOpenAI. Once I prompted Graph GPT4 instantly got to work without issues with misinterpreting or disobeying the prompt, or accepting SystemMessages; and was able to retrieve the correct information from the website.



```
Output

AI
Thought: The user is requesting an explanation of the "WebVoyager" paper which is displayed on the screen. Clicking on the PDF download link will not provide an immediate explanation but would allow them to read the full paper. However, since the action is to explain the paper, I can provide a summary based on the abstract and other visible text.

Action: ANSWER; The "WebVoyager" paper discusses the creation of an advanced web agent called WebVoyager, which is powered by a Large Multimodal Model (LMM) and is designed to handle web tasks end-to-end by interacting with real-world websites. The paper highlights the gap in existing web agents, which typically handle only one input modality and are tested in limited scenarios. WebVoyager aims to address these limitations by being able to complete user instructions using a more robust evaluation protocol that includes automatic evaluation of open-ended web agent tasks. This is achieved by leveraging the capabilities of GPT-4V. The paper claims that WebVoyager significantly outperforms both GPT-4 and a text-only version of the WebVoyager in practical applications, boasting a 55.7% task success rate. The research introduces a new benchmark by gathering real-world tasks from 15 widely used websites to evaluate their agents. The paper suggests that their proposed automatic evaluation method aligns well with human judgment and paves the way for the development of web agents in real-world settings.
```

Using GPT4 does come with a price tag. This command cost around two cents, a small price to pay to get the job done.

Conclusion

LangChain’s LangGraph is an absolutely amazing tool for creating custom agents. With an understanding of State Graphs a user can easily map out how their agents will function. Adding LangSmith to find out what is happening within your StateGraph is an excellent contribution to the development and debugging process. The main challenge here is choosing the correct LLM. Gemini, price-wise, has the potential to be the ideal LLM to run this Agent given a few tweaks to the prompt and parser but nothing can be done until LangChain’s ChatVertexAI accepts SystemMessages. Until then GPT4 is the ideal choice for WebVoyager. GPT4’s ease of access with just API key, along with its clear understanding of the prompt, and ChatOpenAI’s ability to handle SystemMessages pulls it ahead of Gemini.