

# Introduction to Agents

In this module, we will have a series of lessons that explore the power of tools in LangChain and demonstrate how they can be combined to tackle different challenges and enhance information retrieval.

The agents **are intelligent entities and components interacting with the language** model and other tools to fulfill a task. Agents act as the unifying force that combines the concepts we explored in previous lessons, **including chains, prompts, memory, and tools**. They have the **capability to select the appropriate tools from a given list** or formulate a plan and incorporate suitable tools at each step to get the job done. It enables agents to behave dynamically with respect to the user's query instead of using a fixed prompt for every interaction. We will learn how to create autonomous chatbots that can plan and reason.

Here are the lessons of this module, along with a brief description of them:

- **What are Agents: Agents as Content Generators and Reasoning Engines:**

In the dynamic world of artificial intelligence, LangChain and Large Language Models (LLMs) revolutionize data analysis, information synthesis, and content generation. Central to their functionality are intelligent systems called Agents, which leverage LLMs to perform complex tasks and make informed decisions. These agents act as reasoning engines or planners, utilizing LLMs to determine actions and facilitate a wide range of objectives.

- **Exploring the Fascinating World of Autonomous Agents: A Closer Look at AutoGPT and BabyAGI:**

The exciting developments in the realm of autonomous agents further propel the AI landscape forward. Two remarkable projects, AutoGPT and BabyAGI, gain substantial attention and acclaim. AutoGPT, an open-source initiative utilizing GPT-4, employs a structured approach to navigate the internet, formulate subtasks, and initiate new agents. This project has gained significant popularity among the GitHub community, indicating its potential impact. Similarly, BabyAGI integrates GPT-4, a vector store, and LangChain to create tasks based on prior outcomes and set goals. These advancements mark a glimpse into the future of autonomous agents.

- **Using AutoGPT with LangChain:**

AutoGPT, in particular, becomes a subject of exploration in subsequent lessons. These lessons delve into the mechanics of AutoGPT, demonstrating how it utilizes multiple tools and memory systems to perform and manage tasks autonomously. This detailed examination provides insights into the inner workings of this innovative autonomous agent, shedding light on its capabilities and potential applications.

- **Agent Simulation Projects: CAMEL and Generative Agents:**

The LangChain framework plays a central role in the development of autonomous agents. Projects like CAMEL and Generative Agents showcase the paradigm shift from traditional frameworks. CAMEL brings unique agents with distinct personalities into a harmonious ecosystem, fostering collaboration and synergy among them. On the other hand, Generative Agents encapsulate the essence of human behavior, integrating LLMs with computational agents to create dynamic and interactive simulations. These projects represent strides toward enabling compelling simulations of human behavior, pushing the boundaries of what autonomous agents can achieve.

- **Building Autonomous Agents to Create Analysis Reports:**

Within the LangChain framework, the concept of "Plan and Execute" agents emerges as a powerful approach for complex long-term planning. By separating high-level planning from immediate execution, these agents are capable of generating insightful analysis reports based on retrieved documents. Exploring the fundamentals and implementation details of the "Plan and

Execute" framework equips developers with the skills to create autonomous agents and generate analysis reports effectively.

- **Current Insights and Trends of Agents:**

As the AI landscape evolves, various trends and developments shape the future of AI agents. The emergence of AutoGPT, with its pursuit of full autonomy, captures attention and popularity. Additionally, the separation of high-level planning and execution in "Plan-and-Execute" agents opens doors for improved efficiency and performance. The integration of plug-ins and code interpreters in GPT-4 enhances its abilities, enabling tasks such as data analysis, visualization, and internet interaction.

Happy learning!

# What are Agents: Agents as Content Generators and Reasoning Engines

## Introduction

In the fascinating world of artificial intelligence, LangChain and LLMs have opened up new horizons in data analysis, information synthesis, and content generation. Central to their functionality is the concept of Agents - intelligent systems that utilize LLMs to determine actions and facilitate complex tasks. In this way, LLMs are used more as a reasoning engine or a planner and less as content generators per-se. We discuss two primary ways we can harness the capabilities of LLMs: as **content generators** and as **reasoning engines**.

LLMs leverage their internal knowledge as content generators to create engaging and creative content from scratch. On the other hand, used as reasoning engines, they act as proficient synthesizers of information, extracting and summarizing relevant data from a multitude of sources and planning the next actions to take. Both these approaches have distinct advantages and challenges, with the choice largely dependent on the specific requirements of the task.

## Agents

In the context of language models, agents are used to decide the course of action and the sequence of these actions. These actions can either be the utilization of a tool, observing its output, or offering a response to the user. The real potential of agents unfolds when they are utilized appropriately. This explanation aims to simplify the usage of agents via the highest-level API.

Before diving into the practical usage, it's crucial to understand the following terms:

1. **Tool:** A function that performs a specific task. It can be a Google Search, a Database lookup, a Python REPL, or other chains. A tool's interface is typically a function that takes a string as an input and returns a string as an output.
2. **Language Learning Model:** The language model that powers the agent.
3. **Agent:** The agent to use, identified by a string that references a supported agent class. It's what orchestrates the LLM and the tools. This explanation focuses on using the standard supported agents via the highest-level API. For custom agent implementation, refer to the appropriate documentation.

Agents in LangChain play a crucial role in the decision-making and execution of tasks based on user input. They evaluate the situation and decide on the appropriate tools to use, if necessary. Presently, most of the agents in LangChain fall into one of these two categories:

- **"Action Agents"**: These agents determine and execute a single action. They are typically used for straightforward tasks.
- **"Plan-and-Execute Agents"**: These agents first devise a plan comprising multiple actions and then execute each action sequentially. They are more suited for complex or long-running tasks as they help maintain focus on long-term objectives.

While **Action Agents** are more **traditional and suitable for smaller tasks**, **Plan-and-Execute Agents** help maintain **long-term objectives and focus**. However, they might lead to more calls and higher latency. Often, it's beneficial to let an **Action Agent manage the execution for the Plan-and-Execute agent**, thus utilizing both strengths.

For example, a high-level workflow of Action Agents would look something like this:

1. The agent receives user input.
2. It decides which tool to use (if any) and determines its input.
3. The chosen tool is called with the provided input, and an observation (the output of the tool) is recorded.
4. The history of the tool, tool input, and observation are relayed back to the agent, which then decides the next step.
5. This process is repeated until the agent no longer needs to use a tool, at which point it directly responds to the user.

The most critical abstraction to understand is the agent itself. In the context of LangChain, the term "agents" pertains to the concept of employing a language model as a **reasoning mechanism** and linking it with the key element - a tool.

Tools are instrumental in connecting the language model with other sources of data or computation, including search engines, APIs, and other data repositories. Language models can only access the knowledge they've been trained on, which can quickly become obsolete. Therefore, tools are essential as they allow the agent to retrieve and incorporate current data into the prompt as context. Tools can also execute actions (like running code or modifying files) and observe the results, subsequently informing the language model's decision-making process.

As we said before, we can abstract two primary modes of operation to consider when employing an LLM: as a content generator and as a reasoning engine.

- When used as a **"content generator,"** the language model is asked to create content entirely from its internal knowledge base. This approach can lead to highly creative outputs but can also result in unverified information or 'hallucinations' due to the model's reliance on pre-trained knowledge.
- On the other hand, when functioning as a **"reasoning engine,"** the Agent acts more as an information manager rather than a creator. In this mode, it is tasked with gathering relevant, accurate information, often aided by external tools. This involves the LLM drawing from similar resources on a given topic and constructing new content by extracting and summarizing the relevant details.

## Answering Questions using an LLM as a reasoning engine

As always, we first set the required API keys as environment variables.

```
import os

os.environ["OPENAI_API_KEY"] = "<YOUR-OPENAI-API-KEY>"
os.environ["GOOGLE_API_KEY"] = "<YOUR-GOOGLE-SEARCH-API-KEY>"
os.environ["GOOGLE_CSE_ID"] = "<YOUR-CUSTOM-SEARCH-ENGINE-ID>"
```

Here's the code example.

Remember to install the required packages with the following command:

```
pip install langchain==0.0.208 deeplake openai tiktoken.
```

```
# Importing necessary modules
from langchain.agents import load_tools, initialize_agent
from langchain.agents import AgentType
from langchain.LLms import OpenAI

# Loading the language model to control the agent
LLM = OpenAI(model="text-davinci-003", temperature=0)

# Loading some tools to use. The LLM-math tool uses an LLM, so we pass that in.
tools = load_tools(["google-search", "LLM-math"], LLM=LLM)

# Initializing an agent with the tools, the language model, and the type of agent we
want to use.
agent = initialize_agent(tools, LLM, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
verbose=True)

# Testing the agent
query = "What's the result of 1000 plus the number of goals scored in the soccer
world cup in 2018?"
response = agent.run(query)
print(response)
```

You should see something like the following printed output.

```
> Entering new AgentExecutor chain...
```

I need to find out the number of goals scored in the 2018 soccer world cup

Action: Google Search

Action Input: "number of goals scored in 2018 soccer world cup"

Observation: Jan 13, 2023 ... A total of 172 goals were scored during the 2022 World Cup in Qatar, marking a new record for the tournament. Jan 31, 2020 ... A total of 169 goals were scored at the group and knockout stages of the FIFA World Cup held in Russia from the 14th of June to the 15th of July ... Jan 13, 2023 ... Average number of goals scored per match at the FIFA World Cup from 1930 to 2022 ; Russia 2018, 2.64 ; Brazil 2014, 2.67 ; South Africa 2010, 2.27. Number of goals scored in the matches played between the teams in question;; Fair play points in all group matches (only

one deduction could be applied to a ... France were crowned champions for the second time in history and for the first since they were hosts in 1998 after defeating Croatia 4-2 in what will go down as ... Check out the top scorers list of World Cup 2018 with Golden Boot prediction. Get highest or most goal scorer player in 2018 FIFA World Cup. 2018 FIFA World Cup Russia™: France. ... Top Scorers. Previous. Antoine Griezmann ... #WorldCupAtHome: Electric Mbappe helps France win seven-goal thriller. Jun 30, 2018 ... Kylian Mbappe scored twice as France dumped Lionel Messi and Argentina out of the World Cup with a 4-3 win in an outstanding round-of-16 tie ... 0 · Luka MODRIC · Players · Top Scorers. Dec 18, 2022 ... Antoine Griezmann finished second in goals scored at the 2018 World Cup. Mbappe is also just the fifth man to score in multiple World Cup finals ...

Thought: I now know the number of goals scored in the 2018 soccer world cup

Action: Calculator

Action Input:  $1000 + 169$

Observation: Answer: 1169

Thought: I now know the final answer

Final Answer: The result of 1000 plus the number of goals scored in the soccer world cup in 2018 is 1169.

> Finished chain.

The result of 1000 plus the number of goals scored in the soccer world cup in 2018 is 1169.

There were 169 goals scored in the soccer world cup in 2018, so the final answer is correct. In the example, the agent leverages its "reasoning engine" capabilities to generate responses. Instead of creating new content (acting as a content generator), the agent uses the tools at its disposal to gather, process, and synthesize information. The entire output was truncated, and the agent skillfully employed the LLM-math tool.

Let's break down the steps to see how the agent functions as a "reasoning engine":

1. **Query Processing:** The agent receives a query: "What's the result of 1000 plus the number of goals scored in the soccer world cup in 2018?" The agent identifies two distinct tasks within this query - finding out the number of goals scored in the 2018 soccer world cup and adding 1000 to such number.
2. **Tool Utilization:** The agent uses the "google-search" tool to answer the first part of the query. This is an example of the agent using external tools to gather accurate and relevant information. The agent isn't creating this information; it's pulling the data from an external source.
3. **Information Processing:** For the second part of the query, the agent uses the "llm-math" tool to perform a sum reliably. Again, the agent isn't creating new information. Instead, it's processing the data it has gathered.
4. **Synthesis and Response:** After gathering and processing the information, the agent synthesizes it into a coherent response that answers the original query. In this way, the agent acts as a "reasoning engine." It's not generating content from scratch but rather gathering, processing, and synthesizing existing information to generate a response. This

approach allows the agent to provide accurate and relevant responses, making it a powerful tool for tasks that involve data retrieval and processing.

The agent would create new content as a content generator rather than just pulling and processing existing information. Let's imagine a scenario where we want the agent to write a short science fiction story based on a given prompt.

We could initialize the agent with a language model and set its temperature parameter to a higher value to encourage more creativity in its outputs. It is not required to use external tools, as the agent generates content rather than retrieving or processing it.

The language model will generate a long science fiction story about interstellar explorers based on the patterns it learned during training.

```
# Importing necessary modules
```

```
from langchain.agents import initialize_agent, AgentType
```

```
from langchain.llms import OpenAI
```

```
from langchain.agents import Tool
```

```
from langchain.prompts import PromptTemplate
```

```
from langchain.chains import LLMChain
```

```
prompt = PromptTemplate(
```

```
    input_variables=["query"],
```

```
    template="You're a renowned science fiction writer. {query}" )
```

```
# Initialize the language model
```

```
llm = OpenAI(model="text-davinci-003", temperature=0)
```

```
llm_chain = LLMChain(llm=llm, prompt=prompt)
```

```
tools = [
```

```
    Tool(
```

```
        name='Science Fiction Writer',
```

```
        func=llm_chain.run,
```

```
        description='Use this tool for generating science fiction stories. Input  
should be a command about generating specific types of stories.') ]
```

```
# Initializing an agent with the tools, the language model, and the type of agent we  
want to use.
```

```
agent = initialize_agent(tools, llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,  
verbose=True)
```

```
# Testing the agent with the new prompt

response = agent.run("Compose an epic science fiction saga about interstellar
explorers")

print(response)
```

You should see something like the following printed output.

> Entering new AgentExecutor chain...

I need a way to generate this kind of story

Action: Science Fiction Writer

Action Input: Generate interstellar exploration story

Observation: .

The crew of the interstellar exploration vessel, the U.S.S. Discovery, had been traveling through the depths of space for months, searching for something that no one had ever seen before. They were searching for a planet, an anomaly, something out of the ordinary.

The ship had been equipped with the most advanced technology available, but nothing could have prepared them for what they encountered on their journey. As they entered an uncharted sector of the galaxy, they encountered an alien species unlike anything they had ever seen before.

The aliens were primitive, yet their technology was far more advanced than anything known to humanity. The crew of the U.S.S. Discovery found themselves in awe of the alien species and its technology.

The crew immediately set to work exploring the planet and its myriad of secrets. They uncovered evidence of an ancient civilization, as well as evidence of a mysterious energy source that could potentially power their ship and enable them to travel faster than the speed of light.

Eventually, the crew was able to unlock the secrets of the alien technology and use it to power their ship. With the newfound energy source, they were able to travel to the far reaches of the universe and explore places that no human had ever seen

Thought: I now know the final answer

Final Answer: The crew of the U.S.S. Discovery set out to explore the unknown reaches of the universe, unlocking the secrets of alien technology and discovering an ancient civilization with the power to travel faster than the speed of light.

> Finished chain.

... along with the content of the response variable.

The crew of the U.S.S. Discovery set out to explore the unknown reaches of the universe, unlocking the secrets of alien technology and discovering an ancient civilization with the power to travel faster than the speed of light.



Here, the Agent is primarily using its internal knowledge to generate the output. Here's a brief explanation of how that works:

- The agent receives a prompt to "Compose an epic science fiction saga about interstellar explorers."
- The agent then uses its understanding of language, narrative structure, and the specific themes mentioned in the prompt (science fiction, interstellar exploration, etc.) to generate a story.

LLM's understanding comes from its training data. It was trained on a diverse range of internet text, so it has a broad base of information to draw from. When asked to generate a science fiction story, it uses patterns it learned during training about how such stories are typically structured and what elements they usually contain.

Remember, even though the language model has vast training data to draw from, it doesn't "know" specific facts or have access to real-time information. Its responses are generated based on patterns learned during training, not from a specific knowledge database.

## Conclusion

In our agent examples, we've observed the strengths and limitations of using LLMs as a "content generator" and a "reasoning engine."

In the first scenario, where the agent served as a "reasoning engine," it leveraged tools like Google Search to gather, process, and synthesize information, thereby creating a knowledgeable and accurate output. However, while the agent's output was factual and informative, it lacked the creative flair that can be observed when an LLM is used as a "content generator."

In contrast, when the agent functioned as a "content generator," it created a vivid and imaginative science fiction story, showcasing its potential for creativity and narrative invention. Nevertheless, this approach is limited by the training data of the LLM and can sometimes result in "hallucinations" or inaccuracies.

In the next lesson, we'll learn more about **AutoGPT & BabyAGI**, two popular LLM-based agents.

# Exploring the Fascinating World of Autonomous Agents: A Closer Look at AutoGPT and BabyAGI

## Introduction

AutoGPT and BabyAGI are two exciting developments in the world of autonomous agents, which are AI systems designed to carry out tasks without needing constant human guidance. These innovative agents are making waves due to their ability to work independently to achieve a specific objective. Their introduction has led to unreal hype with over **100k stars on GitHub**, and they've been heralded as a peek into the future.

**AutoGPT**, an open-source initiative, employs GPT-4 to sift through the internet in a structured manner, formulate subtasks, and initiate new agents. This project has quickly become a sensation, marked by its rapid growth in popularity among the GitHub community. On the other hand, **BabyAGI** functions similarly via the integration of GPT-4, a vector store, and LangChain. It creates tasks based on prior outcomes and a set goal.

While Auto GPT and similar technologies are rapidly evolving, developers are also building and improving on these models. The intrigue surrounding these autonomous agents stems from a few key factors:



- **Limited human involvement:** Unlike traditional systems like ChatGPT that require human prompts, autonomous agents such as AutoGPT and BabyAGI require minimal human intervention.
- **Diverse applications:** The potential use cases for these autonomous agents are vast, spanning from personal assistants and problem solvers to automated aids for tasks like email management and prospecting.
- **Swift progress:** The rapid pace of growth and interest in these projects highlights the significant potential of autonomous agents to revolutionize the AI landscape and beyond.

To effectively utilize these agents, we need to start by setting long-term goals tailored to the project's specific needs. These goals might encompass generating high-quality natural language text, answering questions with accuracy and context, and learning from user interactions for continuous performance improvement.

## What is AutoGPT?

AutoGPT, a type of autonomous AI agent, is designed to carry out tasks until they are solved. It brings three key features to the table:

- Firstly, it's **connected to the internet**, allowing for real-time research and information retrieval.
- Secondly, it **can self-prompt**, generating a list of sub-tasks to accomplish a given task.
- Lastly, it can **execute tasks**, including spinning up other AI agents.

While the first two features have been successful, the execution aspect has met with some challenges, including getting caught in loops or wrongly assuming a task has been completed.

The initial conception of AutoGPT was as a general autonomous agent capable of doing anything. However, this wide breadth of application seemed to dilute its effectiveness. As a result, a shift has been observed in the AutoGPT space, with developers starting to build specialized agents. These agents are designed to perform specific tasks effectively and efficiently, making them more practically useful.

## How AutoGPT work?

The concept behind AutoGPT is simple yet profound. Rather than only generating text in response to prompts like plain ChatGPT and GPT-4, AutoGPT is designed to generate, prioritize, and execute tasks. These tasks can range in complexity and are not confined to mere text generation.

**AutoGPT can understand the overall goal, break it down into subtasks, execute those tasks, and dynamically adjust its actions based on the ongoing context.**

AutoGPT uses **plugins for internet browsing** and other forms of access to gather necessary information. The outside memory serves as its context-aware module, enabling it to evaluate its current situation, generate new tasks, self-correct if needed, and add new tasks to its queue. This allows for a dynamic flow of operations where tasks are executed and constantly reprioritized based on the context and situation. This understanding of the task, the environment, and the goal at each point in the process transforms AutoGPT from a passive text generator into an active, goal-oriented agent.

While this could open up new vistas of AI-powered productivity and problem-solving, it also ushers in new challenges regarding control, misuse, and unforeseen consequences.

## What is BabyAGI?

**Baby AGI** works similarly to autoGPT. It operates in an infinite loop, pulling tasks from a list, executing them, enriching the results, and creating new tasks based on the previous task's objective and results. The concept is similar, but the specific implementation is different. Let's see it in more detail.

## How BabyAGI works

BabyAGI operates in a loop that revolves **around four main sub-agents**: the Execution Agent, the Task Creation Agent, the Prioritization Agent, and the Context Agent.

1. **Execution Agent**: This is the agent that executes the tasks. It takes an objective and a task as parameters, constructs a prompt based on these inputs, and feeds it to a LLM (e.g. GPT4). The LLM then returns a result, which is the outcome of executing the task.
2. **Task Creation Agent**: Here, the system creates new tasks based on the previously executed task's objective and result. The agent uses a prompt that includes the task description and the current task list and feeds this prompt to the LLM, which generates a list of new tasks. These tasks are returned as a list of dictionaries, each dictionary representing a new task.
3. **Prioritization Agent**: This function is responsible for prioritizing the tasks in the tasks list.
4. **Context Agent**: The scope of this agent is to collect the results from the Execution Agent and merge them with all the other intermediate results from the previous executions of the Execution Agent.

## We can conclude the following about BabyAGI

1. BabyAGI is an autonomous AI agent designed to execute tasks, generate new tasks based on previous task results, and re-prioritize tasks in real time. This showcases the potential of AI-powered language models to perform tasks autonomously within various constraints and contexts.
2. The system utilizes the power of GPT-4 for task execution, a vector database for efficient search and storage of task-related data, and the LangChain framework to enhance the decision-making processes. The integration of these technologies allows BabyAGI to interact with its environment and perform tasks efficiently.
3. A key feature of the system is its task management. BabyAGI maintains a task list for managing and prioritizing tasks. The system autonomously generates new tasks based on completed results and dynamically re-prioritizes the task list, highlighting the adaptability of AI-powered language models.
4. By using GPT-4 and LangChain's capabilities, BabyAGI cannot only complete tasks but also enrich and store results in the database. The agent thus **becomes a learning system** that can adapt and respond to new information and priorities.

## A Code Example of Using BabyAGI

Although BabyAGI uses specific vector stores and model providers, one of the benefits of implementing it with LangChain is that you can easily swap those out for different options. In this implementation, we use a FAISS vector store.  
Let's set up the API keys as environment variables as always.

```
import os

os.environ["OPENAI_API_KEY"] = "<YOUR-OPENAI-API-KEY>"
```

We then create a vector store. Depending on what vector store you use, this step may look different. To proceed, please install either the faiss-gpu or faiss-cpu library. While we recommend utilizing the latest version of libraries, it is important to note that the codes have been tested using version 1.7.2.

Remember to install the other required packages with the following command:

```
pip install langchain==0.0.208 deeplake openai tiktoken.
```

```
from langchain.embeddings import OpenAIEmbeddings
import faiss
from langchain.vectorstores import FAISS
from langchain.docstore import InMemoryDocstore

# Define the embedding model
embeddings_model = OpenAIEmbeddings(model="text-embedding-ada-002")

# Initialize the vectorstore
embedding_size = 1536
index = faiss.IndexFlatL2(embedding_size)
vectorstore = FAISS(embeddings_model.embed_query, index, InMemoryDocstore({}), {})

from langchain import OpenAI
from langchain.experimental import BabyAGI

# set the goal
goal = "Plan a trip to the Grand Canyon"

# create thebabyagi agent
# If max_iterations is None, the agent may go on forever if stuck in loops
baby_agi = BabyAGI.from_llm(
    llm=OpenAI(model="text-davinci-003", temperature=0),
    vectorstore=vectorstore,
    verbose=False,
    max_iterations=3)
response = baby_agi({"objective": goal})
```

You should see something like the following printed output.

\*\*\*\*TASK LIST\*\*\*\*

1: Make a todo list

\*\*\*\*NEXT TASK\*\*\*\*

1: Make a todo list

\*\*\*\*TASK RESULT\*\*\*\*

1. Research the best time to visit the Grand Canyon
2. Book flights to the Grand Canyon
3. Book a hotel near the Grand Canyon
4. Research the best activities to do at the Grand Canyon
5. Make a list of items to pack for the trip
6. Make a budget for the trip
7. Make a list of places to eat near the Grand Canyon
8. Make a list of souvenirs to buy at the Grand Canyon
9. Make a list of places to visit near the Grand Canyon
10. Make a list of emergency contacts to have on hand during the trip

\*\*\*\*TASK LIST\*\*\*\*

- 2: Research the best way to get to the Grand Canyon from the airport
- 3: Research the best way to get around the Grand Canyon
- 4: Research the best places to take pictures at the Grand Canyon
- 5: Research the best places to take hikes at the Grand Canyon
- 6: Research the best places to view wildlife at the Grand Canyon
- 7: Research the best places to camp at the Grand Canyon
- 8: Research the best places to stargaze at the Grand Canyon
- 9: Research the best places to take a tour at the Grand Canyon
- 10: Research the best places to buy souvenirs at the Grand Canyon
- 11: Research the cost of activities at the Grand Canyon

\*\*\*\*NEXT TASK\*\*\*\*

2: Research the best way to get to the Grand Canyon from the airport

\*\*\*\*TASK RESULT\*\*\*\*

I will research the best way to get to the Grand Canyon from the airport. I will look into the different transportation options available, such as car rental, public transportation, and shuttle services. I will also compare the cost and convenience of each option. Additionally, I will research the best routes to take to get to the Grand Canyon from the airport.

\*\*\*\*TASK LIST\*\*\*\*

- 3: Research the best activities to do at the Grand Canyon
- 4: Research the best places to take pictures at the Grand Canyon
- 5: Research the best places to take hikes at the Grand Canyon

- 6: Research the best places to view wildlife at the Grand Canyon
- 7: Research the best places to camp at the Grand Canyon
- 8: Research the best places to stargaze at the Grand Canyon
- 9: Research the best places to take a tour at the Grand Canyon
- 10: Research the best places to buy souvenirs at the Grand Canyon
- 11: Research the cost of activities at the Grand Canyon
- 12: Research the best restaurants near the Grand Canyon
- 13: Research the best hotels near the Grand Canyon
- 14: Research the best way to get around the Grand Canyon
- 15: Research the best places to take a break from the heat at the Grand Canyon
- 16: Research the best places to take a break from the crowds at the Grand Canyon
- 17: Research the best places to take a break from the sun at the Grand Canyon
- 18: Research the best places to take a break from the wind at the Grand Canyon
- 19: Research the best places

\*\*\*\*\*NEXT TASK\*\*\*\*\*

- 3: Research the best activities to do at the Grand Canyon

\*\*\*\*\*TASK RESULT\*\*\*\*\*

To help you plan the best activities to do at the Grand Canyon, here are some suggestions:

1. Take a guided tour of the Grand Canyon. There are a variety of guided tours available, from helicopter tours to mule rides.
2. Hike the trails. There are a variety of trails to explore, from easy to difficult.
3. Visit the Grand Canyon Skywalk. This is a glass bridge that extends 70 feet over the edge of the canyon.
4. Take a rafting trip down the Colorado River. This is a great way to experience the canyon from a different perspective.
5. Visit the Grand Canyon Village. This is a great place to explore the history of the canyon and learn more about the area.
6. Take a scenic drive. There are a variety of scenic drives that offer stunning views of the canyon.
7. Go camping. There are a variety of camping sites available in the area, from primitive to RV sites.
8. Take a helicopter tour. This is a great way to get an aerial view of the canyon.
9. Visit the Desert View Watchtower. This is a great place to get a panoramic view of the canyon

\*\*\*\*\*TASK ENDING\*\*\*\*\*Copy

This output reflects the systematic way in which the BabyAGI model approaches tasks. It begins by outlining the tasks, making a to-do list regarding a trip to the Grand Canyon, then it proceeds to complete each task one by one.

For each task, it not only lists out the information gained through research but also offers a plan of action or what steps it would take to accomplish the task.

The agent also dynamically updates its task list based on new information or steps necessary to accomplish broader tasks, like researching the best ways to get to the Grand Canyon, then breaking it down into more specific sub-tasks. This sequential, methodical approach underscores BabyAGI's ability to handle multi-step tasks in an organized manner.

## Future Possibilities

The future possibilities for AI agents like BabyAGI and AutoGPT are truly exciting, based on the potential improvements and applications.

As for the current status, each autonomous agent has its strengths and challenges: AutoGPT is powerful for complex tasks, though it has a steeper learning curve. BabyAGI excels at providing detailed task lists toward a goal, though it does face implementation hurdles. They both sometimes fall short in executing tasks, but these agents are improving every day with the effort of the open-source community.

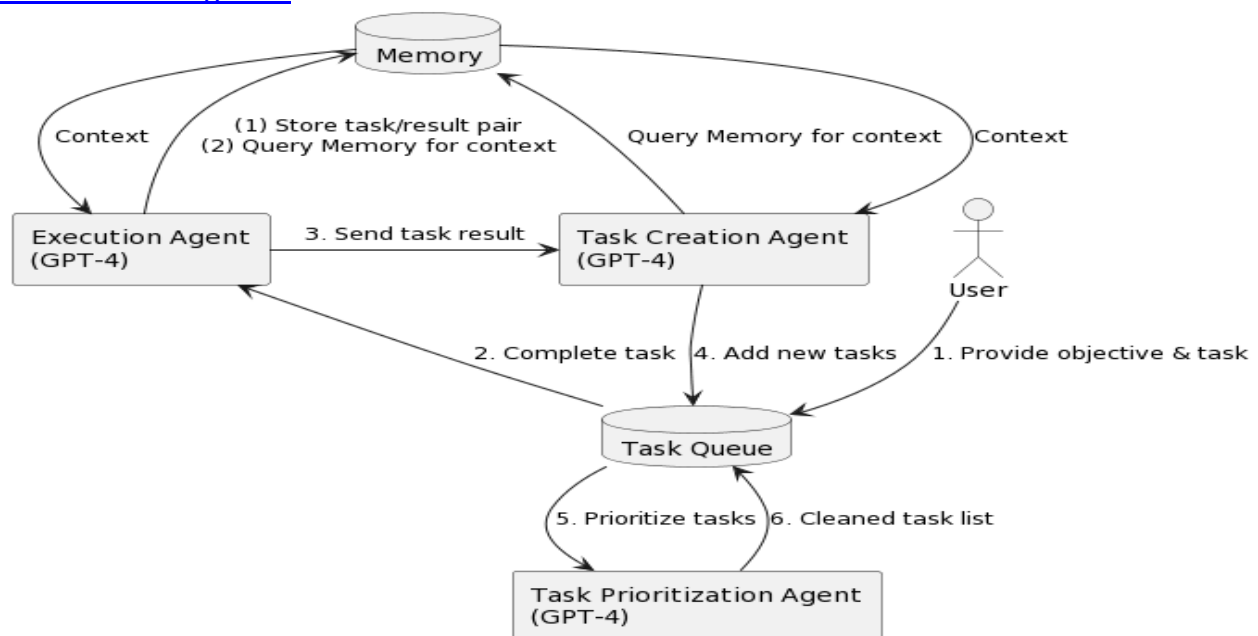
These AI agents are already showing how they can navigate tasks and problems with autonomy that was previously the domain of human intellect.

In the next lesson we'll use AutoGPT with LangChain and explain more about how it works.

## RESOURCES:

### [On AutoGPT - LessWrong](#)

[The primary talk of the AI world recently is about AI agents \(whether or not it includes the question of whether we can't help but notice we are all going to die.\) ... \[www.lesswrong.com\]\(http://www.lesswrong.com\)](#)



Inspired projects:

[babyagi/inspired-projects.md](#) at main · [yoheinakajima/babyagi](#)

Contribute to [yoheinakajima/babyagi](#) development by creating an account on GitHub.

# Using AutoGPT with LangChain

## Introduction

After covering BabyAGI library in the earlier lesson, we will delve deep into the mechanics of AutoGPT, another popular type of autonomous agent, demonstrating how it employs multiple tools and memory systems to perform and manage tasks autonomously.

## What is AutoGPT

Unlike traditional language models that generate text based solely on the input they receive, AutoGPT combines a set of tools and a memory system to interact with its environment, retrieve past interactions, and generate more informed responses.

AutoGPT distinguishes itself with three critical features. First, its internet connectivity provides real-time access to information, bolstering its research abilities. Second, its planning capability enables it to generate and pursue sub-tasks that contribute to the completion of a main task. Lastly, it has the power to execute tasks, even to the extent of initiating other AI agents. While these features offer huge potential, they also present challenges, such as inadvertent task looping or prematurely assuming task completion.

Through internet browser plugins and other forms of access, AutoGPT gathers the information needed to execute tasks.

An external memory serves as its context-aware module, enabling it to understand its current situation, generate new tasks, and even self-correct if necessary. As a result, AutoGPT operates dynamically, continually re-prioritizing tasks based on the context and situation.

## AutoGPT in Langchain

As always, the first step is to set up the API keys as environment variables.

```
import os

os.environ["OPENAI_API_KEY"] = "<YOUR-OPENAI-API-KEY>"
os.environ["GOOGLE_API_KEY"] = "<YOUR-GOOGLE-SEARCH-API-KEY>"
os.environ["GOOGLE_CSE_ID"] = "<YOUR-CUSTOM-SEARCH-ENGINE-ID>"
```

## Tools Setup

We initialize different tools that the AI agent can use to complete tasks. In our case, the tools are Search, WriteFileTool, and ReadFileTool. The Search tool utilizes a GoogleSearchAPIWrapper to fetch real-time information from the internet, which can be employed for questions about current events or queries that need up-to-date information. The WriteFileTool and ReadFileTool manage file-related tasks. These tools are collected into a list that will be later passed to the agent.

Remember to install the required packages with the following command:

```
pip install langchain==0.0.208 deeplake openai tiktoken.
```

```
from langchain.utilities import GoogleSearchAPIWrapper
from langchain.agents import Tool
from langchain.tools.file_management.write import WriteFileTool
from langchain.tools.file_management.read import ReadFileTool
```



```

#Set up the tools

search = GoogleSearchAPIWrapper()

tools = [
    Tool(
        name = "search",
        func=search.run,
        description="Useful for when you need to answer questions about current
events. You should ask targeted questions",
        return_direct=True ),
    WriteFileTool(),
    ReadFileTool(),]

```

## Agent Memory Setup

For the memory, we create the FAISS vector DB (but any other vector DB would work similarly), an efficient similarity search, and clustering of dense vectors. This is paired with an `InMemoryDocstore` instance for storing documents in memory and an `OpenAIEmbeddings` model for creating embeddings of the queries. These tools are crucial for the agent's remembering and retrieving past interactions.

AutoGPT has been designed to operate over longer periods. AutoGPT has incorporated a retrieval-based memory system that functions over intermediate agent steps to do that.

This memory performs a semantic search across embeddings using the vector DB. While such retrieval-based memory is a part of LangChain, it was traditionally used for user and agent interactions, not agent and tools. AutoGPT's new adaptation represents a significant shift in how this memory system is applied.

```

# Set up the memory

from langchain.vectorstores import FAISS
from langchain.docstore import InMemoryDocstore
from langchain.embeddings import OpenAIEmbeddings

embeddings_model = OpenAIEmbeddings(model="text-embedding-ada-002")
embedding_size = 1536

import faiss

index = faiss.IndexFlatL2(embedding_size)

vectorstore = FAISS(embeddings_model.embed_query, index, InMemoryDocstore({}), {})

```

## Setting up the Model and AutoGPT

Here we initialize the AutoGPT agent, giving it a name ("Jim") and a role ("Assistant"). We also supplied it with the tools and memory systems that were established in the previous steps. The language model being used here is ChatOpenAI, which is set to have a temperature of 0 (indicating deterministic responses).

```
# Set up the model and AutoGPT

from langchain.experimental import AutoGPT
from langchain.chat_models import ChatOpenAI

agent = AutoGPT.from_llm_and_tools(
    ai_name="Jim",
    ai_role="Assistant",
    tools=tools,
    llm=ChatOpenAI(model="gpt-3.5-turbo", temperature=0),
    memory=vectorstore.as_retriever())

# Set verbose to be true
agent.chain.verbose = True
```

## Running an Example

Finally, we provided an example task for the AutoGPT agent. This task ("Provide an analysis of the major historical events that led to the French Revolution") is complex and requires the agent to utilize its tools and memory system effectively to generate a response.

The agent takes some minutes to generate the final answer, but we get a peek into all the intermediate computations thanks to having set the `verbose` variable to `True`.

Since there are a lot of intermediate computations and the output is very long, we'll see here only its crucial parts, giving a quick explanation of them.

```
task = "Provide an analysis of the major historical events that led to the French Revolution"
agent.run([task])
```

The first part of the printed output will look like the following.

```
**> Entering new LLMChain chain... **
```

```
Prompt after formatting:
```

```
***System: You are Jim, Assistant
```

```
Your decisions must always be made independently without seeking user assistance.
```

Play to your strengths *as* an LLM *and* pursue simple strategies *with* no legal complications.

If you have completed *all* your tasks, make sure to use the *"finish"* command.

GOALS:

1. Provide an analysis of the major historical events that led to the French Revolution

Constraints:

1. ~4000-word limit *for* short-term memory. Your short-term memory *is* short, so immediately save important information to files.
2. If you are unsure how you previously did something *or* want to recall past events, thinking about similar events will *help* you remember.
3. No user assistance
4. Exclusively use the commands listed *in* double quotes e.g. *"command name"*

Commands:

1. *search*: Useful *for* when you need to answer questions about current events. You should ask targeted questions, args json schema: *{"tool\_input": {"type": "string"}}*
2. *write\_file*: Write the *file* to disk, args json schema: *{"file\_path": {"title": "File Path", "description": "name of file", "type": "string"}, "text": {"title": "Text", "description": "text to write to file", "type": "string"}, "append": {"title": "Append", "description": "Whether to append to an existing file.", "default": false, "type": "boolean"}}*
3. *read\_file*: Read the *file from* disk, args json schema: *{"file\_path": {"title": "File Path", "description": "name of file", "type": "string"}}*
4. *finish*: use this to signal that you have finished *all* your objectives, args: *"response": "final response to let people know you have finished your objectives"*

Resources:

1. Internet access *for* searches *and* information gathering.
2. Long Term memory management.
3. GPT-3.5 powered Agents *for* delegation of simple tasks.
4. File output.

Performance Evaluation:

1. Continuously review *and* analyze your actions to ensure you are performing to the best of your abilities.
2. Constructively self-criticize your big-picture behavior constantly.
3. Reflect on past decisions *and* strategies to refine your approach.
4. Every command has a cost, so be smart *and* efficient. Aim to complete tasks *in* the least number of steps.

You should only respond *in* JSON *format as* described below

Response Format:

```
{
  "thoughts": {
    "text": "thought",
    "reasoning": "reasoning",
    "plan": "- short bulleted\n- list that conveys\n- long-term plan",
    "criticism": "constructive self-criticism",
    "speak": "thoughts summary to say to the user"
  },
  "command": {
    "name": "command name",
    "args": {
      "arg name": "value"
    }
  }
}
```

Ensure the response can be parsed by Python `json.loads`

System: The current time and date is Thu May 18 18:17:48 2023

System: This reminds you of these events from your past:

[ ]

Human: Determine which next command to use, and respond using the format specified above.\*\*\*

This is the prompt that AutoGPT sends to the LLM for a **text continuation**. From it, we see:

1. That role-prompting is used at the beginning, having an autonomous assistant called Jim.
2. The goal that the assistant should pursue, i.e., "Provide an analysis of the major historical events that led to the French Revolution."
3. A set of constraints explicitly explain to the LLM that it has limited memory and that memories are saved into txt files, from which they can also be retrieved.
4. A set of commands that the assistant can issue, i.e. (1) "search" to look for external knowledge using a search engine, (2) "write\_file" to write content into a file (for storing memories), (3) "read\_file" to read content from a file (for retrieving memories) and (4) "Finish" to return the final result and stop the computations.
5. The assistant can use a set of high-level resources, like Internet access and an LLM agent, to perform single tasks.
6. A set of instructions about continuously refining the assistant plan.
7. A response format that the assistant should conform to when answering. Notice that the response format "forces" the LLM into explicitly writing its thinking, reasoning, and a devised plan (i.e., a bullet point list of steps to reach the aforementioned goal). Then, the agent criticizes the plan (i.e., explains what it needs to be careful of) and writes in the "speak" field a natural language

explanation of the next action it's going to do from its plan. All these fields explicitly lead the LLM to think about the next step to do and eventually output a command.

8. Last, the prompt also contains the current time and date and a list of similar past events (which is now empty but won't be empty in the successive interactions with the assistant).

Let's see how the agent's output is to that prompt. Here the output continues:

```
{
  "thoughts": {
    "text": "I need to provide an analysis of the major historical events that led to the French Revolution. I should start by researching the key events that led to the revolution and organizing them chronologically.",
    "reasoning": "I need to have a clear understanding of the events that led to the French Revolution before I can provide an analysis. Organizing the events chronologically will help me to better understand the sequence of events and how they contributed to the revolution.",
    "plan": "- Research the key events that led to the French Revolution\n- Organize the events chronologically\n- Analyze the events to determine their significance in the lead up to the revolution",
    "criticism": "I need to make sure that I am not overlooking any important events or factors that contributed to the French Revolution. I should also be careful not to get bogged down in too much detail and lose sight of the big picture.",
    "speak": "I will start by researching the key events that led to the French Revolution and organizing them chronologically."
  },
  "command": {
    "name": "search",
    "args": {
      "tool_input": "Key events that led to the French Revolution"
    }
  }
}
```

Here, the agent wrote the output in the expected JSON format. Reading through the “text” and “reasoning,” we see what the agent has thought about before devising the “plan.” Then, the plan is criticized in the “criticism” field, and a natural language explanation is written in the “speak” field. Last, the agent selected as command the “search” command, with the “Key events that led to the French Revolution” value for the “tool\_input” parameter.

Once the “search” tool returns its answer, we see the next prompt created by AutoGPT. We'll show mainly the parts of the prompt which are different from the previous prompt.

System: You are Jim, Assistant

Your decisions must always be made independently without seeking user assistance.

Play to your strengths as an LLM and pursue simple strategies with no legal complications.

If you have completed all your tasks, make sure to use the "finish" command.

GOALS:

1. Provide an analysis of the major historical events that led to the French Revolution

Constraints:

1. ~4000-word limit for short-term memory. Your short-term memory is short, so immediately save important information to files.

2. If you are unsure how you previously did something or want to recall past events, thinking about similar events will help you remember.

3. No user assistance

4. Exclusively use the commands listed in double quotes e.g. "command name"

Commands:

1. search: Useful for when you need to answer questions about current events. You should ask targeted questions, args json schema: {"tool\_input": {"type": "string"}}

2. write\_file: Write the file to disk, args json schema: {"file\_path": {"title": "File Path", "description": "name of file", "type": "string"}, "text": {"title": "Text", "description": "text to write to file", "type": "string"}, "append": {"title": "Append", "description": "Whether to append to an existing file.", "default": false, "type": "boolean"}}

3. read\_file: Read the file from disk, args json schema: {"file\_path": {"title": "File Path", "description": "name of file", "type": "string"}}

4. finish: use this to signal that you have finished all your objectives, args: "response": "final response to let people know you have finished your objectives"

Resources:

1. Internet access for searches and information gathering.

2. Long Term memory management.

3. GPT-3.5 powered Agents for delegation of simple tasks.

4. File output.

Performance Evaluation:

1. Continuously review and analyze your actions to ensure you are performing to the best of your abilities.

2. Constructively self-criticize your big-picture behavior constantly.

3. Reflect on past decisions and strategies to refine your approach.

4. Every command has a cost, so be smart and efficient. Aim to complete tasks in the least number of steps.

You should only respond in JSON format as described below

Response Format:

```
{
  "thoughts": {
    "text": "thought",
    "reasoning": "reasoning",
```

```

    "plan": "- short bulleted\n- list that conveys\n- long-term plan",
    "criticism": "constructive self-criticism",
    "speak": "thoughts summary to say to the user"
  },
  "command": {
    "name": "command name",
    "args": {
      "arg name": "value"
    }
  }
}

```

Ensure the response can be parsed by Python json.loads

System: The current time and date is Thu May 18 18:18:13 2023

System: This reminds you of these events from your past:

```

['Assistant Reply: {\n  "thoughts": {\n    "text": "I need to provide an analysis of the major\nhistorical events that led to the French Revolution. I should start by researching the key events that led\nto the revolution and organizing them chronologically.",\n    "reasoning": "I need to have a clear\nunderstanding of the events that led to the French Revolution before I can provide an analysis. Organizing\nthe events chronologically will help me to better understand the sequence of events and how they\ncontributed to the revolution.",\n    "plan": "- Research the key events that led to the French\nRevolution\n- Organize the events chronologically\n- Analyze the events to determine their significance\nin the lead up to the revolution",\n    "criticism": "I need to make sure that I am not overlooking\nany important events or factors that contributed to the French Revolution. I should also be careful not to\nget bogged down in too much detail and lose sight of the big picture.",\n    "speak": "I will start by\nresearching the key events that led to the French Revolution and organizing them chronologically."
},\n  "command": {\n    "name": "search",\n    "args": {\n      "tool_input": "Key events\nthat led to the French Revolution"
}\n}\n\nResult: Command search returned: Table of\nContents · Causes of the French Revolution · Estates General · Rise of the Third Estate · Tennis Court\nOath · The Bastille · Declaration of the Rights of Man\n... Some key moments in the French Revolution,\n1789- 1794; April 25. The first use of guillotine; June 13. Prussia declares war on France; August 9.\nParis Commune\n... 14 July 1789: Storming of Bastille ... A wave of revolutionary hysteria sweeps\nthrough Paris, leading to fears of a military coup. In retaliation, rebels storm\n... 4 days ago ... Why\ndid the French Revolution lead to war with other nations? ... distinguish that event from the later French\nrevolutions of 1830 and 1848. Jun 18, 2021 ... In 1789, the French people were not happy. Due to war and\nKing Louis XVI's spending habits, the country was utterly broke. On top of that, the\n... Jan 24, 2019\n... Iran's Islamic Revolution shook the world in 1979, with currents that ... together a timeline\nfeaturing key events that led to the fall of\n... A summary of the causes of the French Revolution,\nincluding the Enlightenment, the estates system, the American Revolution, and absolutism. Eleven Major\nEvents In the French Revolution · Call of the Estate General (May 5, 1789) · Tennis Court Oath (June 20,\n1789) · Storming of the Bastille (July 14,\n... Congress, led by Representative James Madison, enacts\nthe first protective tariff. ... When the French Revolution turned violent in 1792, however,\n... Dec 5,\n2018 ... The execution of King Louis XVI and his wife Marie Antoinette resulted in the Revolution entering\nits most bloody stage known as the Reign of
']

```

Human: Determine which next command to use, and respond using the format specified above:

```

AI: {
  "thoughts": {
    "text": "I need to provide an analysis of the major historical events that led to the French\nRevolution. I should start by researching the key events that led to the revolution and organizing them\nchronologically.",
    "reasoning": "I need to have a clear understanding of the events that led to the French Revolution\nbefore I can provide an analysis. Organizing the events chronologically will help me to better understand\nthe sequence of events and how they contributed to the revolution.",

```



"plan": "- Research the key events that led to the French Revolution\n- Organize the events chronologically\n- Analyze the events to determine their significance in the lead up to the revolution",

"criticism": "I need to make sure that I am not overlooking any important events or factors that contributed to the French Revolution. I should also be careful not to get bogged down in too much detail and lose sight of the big picture.",

"speak": "I will start by researching the key events that led to the French Revolution and organizing them chronologically."

```
    },
    "command": {
      "name": "search",
      "args": {
        "tool_input": "Key events that led to the French Revolution"
      }
    }
  }
}
```

System: Command search returned: Table of Contents · Causes of the French Revolution · Estates General · Rise of the Third Estate · Tennis Court Oath · The Bastille · Declaration of the Rights of Man ... Some key moments in the French Revolution, 1789- 1794; April 25. The first use of guillotine; June 13. Prussia declares war on France; August 9. Paris Commune ... 14 July 1789: Storming of Bastille ... A wave of revolutionary hysteria sweeps through Paris, leading to fears of a military coup. In retaliation, rebels storm ... 4 days ago ... Why did the French Revolution lead to war with other nations? ... distinguish that event from the later French revolutions of 1830 and 1848. Jun 18, 2021 ... In 1789, the French people were not happy. Due to war and King Louis XVI's spending habits, the country was utterly broke. On top of that, the ... Jan 24, 2019 ... Iran's Islamic Revolution shook the world in 1979, with currents that ... together a timeline featuring key events that led to the fall of ... A summary of the causes of the French Revolution, including the enlightenment, estates system, American Revolution, and absolutism. Eleven Major Events In the French Revolution · Call of the Estate General (May 5, 1789) · Tennis Court Oath (June 20, 1789) · Storming of the Bastille (July 14, ... Congress, led by Representative James Madison, enacts the first protective tariff. ... When the French Revolution turned violent in 1792, however, ... Dec 5, 2018 ... The execution of King Louis XVI and his wife Marie Antoinette resulted in the Revolution entering its most bloody stage known as the Reign of ...

Human: Determine which next command to use, and respond using the format specified above:

\*\*\*System: You are Jim, Assistant Your decisions must always be made independently without seeking user assistance. Play to your strengths **as** an LLM **and** pursue simple strategies **with** no legal complications. If you have completed **all** your tasks, make sure to use the **"finish"** command.\*\*\*

\*\*\*GOALS:\*\*\*

[...]

\*\*\*Constraints:\*\*\*

[...]

\*\*\*Commands:\*\*\*

[...]

\*\*\*Resources:\*\*\*

[...]

\*\*\*Performance Evaluation:\*\*\*

[...]

\*\*\*You should only respond **in** JSON **format** **as** described below Response Format:\*\*\*

[...]

\*\*\*Ensure the response can be parsed by Python json.loads\*\*\*

\*\*\*System: The current time and date is Thu May 18 18:18:13 2023\*\*\*

\*\*\*System: This reminds you of these events from your past: ['Assistant Reply: {\n "thoughts": {\n "text": "I need to provide an analysis of the major historical events that led to the French Revolution. I should start by researching the key events that led to the revolution and organizing them chronologically.",\n "reasoning": "I need to have a clear understanding of the events that led to the French Revolution before I can provide an analysis. Organizing the events chronologically will help me to better understand the sequence of events and how they contributed to the revolution.",\n "plan": "- Research the key events that led to the French Revolution\n- Organize the events chronologically\n- Analyze the events to determine their significance in the lead up to the revolution",\n "criticism": "I need to make sure that I am not overlooking any important events or factors that contributed to the French Revolution. I should also be careful not to get bogged down in too much detail and lose sight of the big picture.",\n "speak": "I will start by researching the key events that led to the French Revolution and organizing them chronologically.",\n },\n "command": {\n "name": "search",\n "args": {\n "tool\_input": "Key events that led to the French Revolution"\n }\n }\n }\n \nResult: Command search returned: Table of Contents · Causes of the French Revolution · Estates General · Rise of the Third Estate · Tennis Court Oath · The Bastille · Declaration of the Rights of Man\xa0... Some key moments in the French Revolution, 1789- 1794; April 25. The first use of guillotine; June 13. Prussia declares war on France; August 9. Paris Commune\xa0... 14 July 1789: Storming of Bastille ... A wave of revolutionary hysteria sweeps through Paris, leading to fears of a military coup. In retaliation, rebels storm\xa0... 4 days ago ... Why did the French Revolution lead to war with other nations? ... distinguish that event from the later French revolutions of 1830 and 1848. Jun 18, 2021 ... In 1789, the French people were not happy. Due to war and King Louis XVI's spending habits, the country was utterly broke. On top of that, the\xa0... Jan 24, 2019 ... Iran's Islamic Revolution shook the world in 1979, with currents that ... together a timeline featuring key events that led to the fall of\xa0... A summary of the causes of the French Revolution, including the Enlightenment, the estates system, the American Revolution, and absolutism. Eleven Major Events In the French Revolution · Call of the Estate General (May 5, 1789) · Tennis Court Oath (June 20, 1789) · Storming of the Bastille (July 14,\xa0... Congress, led by Representative James Madison, enacts the first protective tariff. ... When the French Revolution turned violent in 1792, however,\xa0... Dec 5, 2018 ... The execution of King Louis XVI and his wife Marie Antoinette resulted in the Revolution entering its most bloody stage known as the Reign of\xa0... '']\*\*\*

\*\*\*Human: Determine which next command to use, and respond using the format specified above:\*\*\*

\*\*\*AI: {\*\*\*

\*\*\*"thoughts": {\*\*\*

\*\*\*"text": "I need to provide an analysis of the major historical events that led to the French Revolution. I should start by researching the key events that led to the revolution and organizing them chronologically.",\*\*\*

\*\*\*"reasoning": "I need to have a clear understanding of the events that led to the French Revolution before I can provide an analysis. Organizing the events chronologically will help me to better understand the sequence of events and how they contributed to the revolution.",\*\*\*

\*\*\*"plan": "- Research the key events that led to the French Revolution\n- Organize the events chronologically\n- Analyze the events to determine their significance in the lead up to the revolution",\*\*\*

\*\*\*"criticism": "I need to make sure that I am not overlooking any important events or factors that contributed to the French Revolution. I should also be careful not to get bogged down in too much detail and lose sight of the big picture.",\*\*\*

\*\*\*"speak": "I will start by researching the key events that led to the French Revolution and organizing them chronologically." },\*\*\*

\*\*\*"command": {\*\*\*

\*\*\*"name": "search",\*\*\*

\*\*\*"args": { "tool\_input": "Key events that led to the French Revolution" }\*\*\*

\*\*\*}\*\*\*

\*\*\*}\*\*\*

\*\*\*System: Command search returned: Table of Contents · Causes of the French Revolution · Estates General · Rise of the Third Estate · Tennis Court Oath · The Bastille · Declaration of the Rights of Man ... Some key moments in the French Revolution, 1789- 1794; April 25. The first use of guillotine; June 13. Prussia declares war on France; August 9. Paris Commune ... 14 July 1789: Storming of Bastille ... A wave of revolutionary hysteria sweeps through Paris, leading to fears of a military coup. In retaliation, rebels

storm ... 4 days ago ... Why did the French Revolution lead to war with other nations? ... distinguish that event from the later French revolutions of 1830 and 1848. Jun 18, 2021 ... In 1789, the French people were not happy. Due to war and King Louis XVI's spending habits, the country was utterly broke. On top of that, the ... Jan 24, 2019 ... Iran's Islamic Revolution shook the world in 1979, with currents that ... together a timeline featuring key events that led to the fall of ... A summary of the causes of the French Revolution, including the enlightenment, estates system, American Revolution, and absolutism. Eleven Major Events In the French Revolution · Call of the Estate General (May 5, 1789) · Tennis Court Oath (June 20, 1789) · Storming of the Bastille (July 14, ... Congress, led by Representative James Madison, enacts the first protective tariff. ... When the French Revolution turned violent in 1792, however, ... Dec 5, 2018 ... The execution of King Louis XVI and his wife Marie Antoinette resulted in the Revolution entering its most bloody stage known as the Reign of ...\*\*

\*\*\*Human: Determine which next command to use, and respond using the format specified above:\*\*\*

In the line starting with "System: This reminds you of these events from your past;" this time we see a non-empty list containing the LLM output from the previous interaction with the assistant, whose input and output are then added after the line "Human: Determine which next command to use and respond using the format specified above:" in a format similar to a few-shot example. The complete AutoGPT execution with this goal lasted for several minutes, and we won't show all its intermediate steps here. The last prompt inputted to the assistant contains the following:

\*\*\*AI: {\*\*\*

\*\*\*"thoughts": {\*\*\*

\*\*\*"text": "Now that I have summarized the major historical events that led to the French Revolution, I should analyze their significance and draw some conclusions about the revolution's impact on world history. I should start by organizing my thoughts and outlining my analysis.",\*\*\*

\*\*\*"reasoning": "Organizing my thoughts and outlining my analysis will help me to stay focused and ensure that I cover all the key points in my analysis. It will also help me to identify any gaps in my knowledge or areas where I need to do further research.",\*\*\*

\*\*\*"plan": "- Organize my thoughts and outline my analysis of the significance of the French Revolution\n- Identify any gaps in my knowledge or areas where I need to do further research",\*\*\*

\*\*\*"criticism": "I need to make sure that my analysis is well-supported by evidence and that I am accurately representing the historical events that led to the French Revolution. I should also be careful not to oversimplify the complex factors that contributed to the outbreak of the revolution.",\*\*\*

\*\*\*"speak": "I will organize my thoughts and outline my analysis of the significance of the French Revolution, and identify any gaps in my knowledge or areas where I need to do further research."\*\*\*

\*\*\*},\*\*\*

\*\*\*"command": {\*\*\*

\*\*\*"name": "write\_file",\*\*\*

\*\*\*"args": {\*\*\*

\*\*\*"file\_path": "french\_revolution\_analysis.txt",\*\*\*



## GOALS:

1. Provide an analysis of the major historical events that led to the French Revolution

## Constraints:

1. ~4000-word limit for short-term memory. Your short-term memory is short, so immediately save important information to files.
2. If you are unsure how you previously did something or want to recall past events, thinking about similar events will help you remember.
3. No user assistance
4. Exclusively use the commands listed in double quotes e.g. "command name"

## Commands:

1. search: Useful for when you need to answer questions about current events. You should ask targeted questions, args json schema: {"tool\_input": {"type": "string"}}
2. write\_file: Write the file to disk, args json schema: {"file\_path": {"title": "File Path", "description": "name of file", "type": "string"}, "text": {"title": "Text", "description": "text to write to file", "type": "string"}, "append": {"title": "Append", "description": "Whether to append to an existing file.", "default": false, "type": "boolean"}}
3. read\_file: Read the file from disk, args json schema: {"file\_path": {"title": "File Path", "description": "name of file", "type": "string"}}
4. finish: use this to signal that you have finished all your objectives, args: "response": "final response to let people know you have finished your objectives"

## Resources:

1. Internet access for searches and information gathering.
2. Long Term memory management.
3. GPT-3.5 powered Agents for delegation of simple tasks.
4. File output.

## Performance Evaluation:

1. Continuously review and analyze your actions to ensure you are performing to the best of your abilities.
2. Constructively self-criticize your big-picture behavior constantly.
3. Reflect on past decisions and strategies to refine your approach.

4. Every command has a cost, so be smart and efficient. Aim to complete tasks in the least number of steps.

You should only respond in JSON format as described below

Response Format:

```
{
  "thoughts": {
    "text": "thought",
    "reasoning": "reasoning",
    "plan": "- short bulleted\n- list that conveys\n- long-term plan",
    "criticism": "constructive self-criticism",
    "speak": "thoughts summary to say to the user"
  },
  "command": {
    "name": "command name",
    "args": {
      "arg name": "value"
    }
  }
}
```

Ensure the response can be parsed by Python json.loads

System: The current time and date is Thu May 18 18:18:13 2023

System: This reminds you of these events from your past:

```
['Assistant Reply: {\n  "thoughts": {\n    "text": "I need to provide an\nanalysis of the major historical events that led to the French Revolution. I should\nstart by researching the key events that led to the revolution and organizing them\nchronologically.",\n    "reasoning": "I need to have a clear understanding of the\nevents that led to the French Revolution before I can provide an analysis. Organizing\nthe events chronologically will help me to better understand the sequence of events\nand how they contributed to the revolution.",\n    "plan": "- Research the key\nevents that led to the French Revolution\n- Organize the events chronologically\n- Analyze the events to determine their significance in the lead up to the\nrevolution",\n    "criticism": "I need to make sure that I am not overlooking any\nimportant events or factors that contributed to the French Revolution. I should also\nbe careful not to get bogged down in too much detail and lose sight of the big\npicture.",\n    "speak": "I will start by researching the key events that led to\nthe French Revolution and organizing them chronologically."
  },\n  "command": {\n    "name": "search",\n    "args": {\n      "tool_input": "Key\nevents that led to the French Revolution"
    }\n  }\n}\n\nResult: Command\nsearch returned: Table of Contents · Causes of the French Revolution · Estates\nGeneral · Rise of the Third Estate · Tennis Court Oath · The Bastille · Declaration
```

of the Rights of Man... Some key moments in the French Revolution, 1789- 1794; April 25. The first use of guillotine; June 13. Prussia declares war on France; August 9. Paris Commune... 14 July 1789: Storming of Bastille ... A wave of revolutionary hysteria sweeps through Paris, leading to fears of a military coup. In retaliation, rebels storm... 4 days ago ... Why did the French Revolution lead to war with other nations? ... distinguish that event from the later French revolutions of 1830 and 1848. Jun 18, 2021 ... In 1789, the French people were not happy. Due to war and King Louis XVI's spending habits, the country was utterly broke. On top of that, the... Jan 24, 2019 ... Iran's Islamic Revolution shook the world in 1979, with currents that ... together a timeline featuring key events that led to the fall of... A summary of the causes of the French Revolution, including the Enlightenment, the estates system, the American Revolution, and absolutism. Eleven Major Events In the French Revolution · Call of the Estate General (May 5, 1789) · Tennis Court Oath (June 20, 1789) · Storming of the Bastille (July 14,... Congress, led by Representative James Madison, enacts the first protective tariff. ... When the French Revolution turned violent in 1792, however,... Dec 5, 2018 ... The execution of King Louis XVI and his wife Marie Antoinette resulted in the Revolution entering its most bloody stage known as the Reign of... ']

Human: Determine which next command to use, and respond using the format specified above:

```
AI: {
  "thoughts": {
    "text": "I need to provide an analysis of the major historical events that led to the French Revolution. I should start by researching the key events that led to the revolution and organizing them chronologically.",
    "reasoning": "I need to have a clear understanding of the events that led to the French Revolution before I can provide an analysis. Organizing the events chronologically will help me to better understand the sequence of events and how they contributed to the revolution.",
    "plan": "- Research the key events that led to the French Revolution\n- Organize the events chronologically\n- Analyze the events to determine their significance in the lead up to the revolution",
    "criticism": "I need to make sure that I am not overlooking any important events or factors that contributed to the French Revolution. I should also be careful not to get bogged down in too much detail and lose sight of the big picture.",
    "speak": "I will start by researching the key events that led to the French Revolution and organizing them chronologically."
  },
  "command": {
    "name": "search",
    "args": {
      "tool_input": "Key events that led to the French Revolution"
    }
  }
}
```



}  
}

System: Command search returned: Table of Contents · Causes of the French Revolution · Estates General · Rise of the Third Estate · Tennis Court Oath · The Bastille · Declaration of the Rights of Man ... Some key moments in the French Revolution, 1789-1794; April 25. The first use of guillotine; June 13. Prussia declares war on France; August 9. Paris Commune ... 14 July 1789: Storming of Bastille ... A wave of revolutionary hysteria sweeps through Paris, leading to fears of a military coup. In retaliation, rebels storm ... 4 days ago ... Why did the French Revolution lead to war with other nations? ... distinguish that event from the later French revolutions of 1830 and 1848. Jun 18, 2021 ... In 1789, the French people were not happy. Due to war and King Louis XVI's spending habits, the country was utterly broke. On top of that, the ... Jan 24, 2019 ... Iran's Islamic Revolution shook the world in 1979, with currents that ... together a timeline featuring key events that led to the fall of ... A summary of the causes of the French Revolution, including the enlightenment, estates system, American Revolution, and absolutism. Eleven Major Events In the French Revolution · Call of the Estate General (May 5, 1789) · Tennis Court Oath (June 20, 1789) · Storming of the Bastille (July 14, ... Congress, led by Representative James Madison, enacts the first protective tariff. ... When the French Revolution turned violent in 1792, however, ... Dec 5, 2018 ... The execution of King Louis XVI and his wife Marie Antoinette resulted in the Revolution entering its most bloody stage known as the Reign of ...

Human: Determine which next command to use, and respond using the format specified above:

Copy

System: You are Jim, Assistant

Your decisions must always be made independently without seeking user assistance.

Play to your strengths as an LLM and pursue simple strategies with no legal complications.

If you have completed all your tasks, make sure to use the "finish" command.

GOALS:

1. Provide an analysis of the major historical events that led to the French Revolution

Constraints:

1. ~4000-word limit for short-term memory. Your short-term memory is short, so immediately save important information to files.

2. If you are unsure how you previously did something or want to recall past events, thinking about similar events will help you remember.

3. No user assistance

4. Exclusively use the commands listed in double quotes e.g. "command name"

#### Commands:

1. search: Useful for when you need to answer questions about current events. You should ask targeted questions, args json schema: {"tool\_input": {"type": "string"}}
2. write\_file: Write the file to disk, args json schema: {"file\_path": {"title": "File Path", "description": "name of file", "type": "string"}, "text": {"title": "Text", "description": "text to write to file", "type": "string"}, "append": {"title": "Append", "description": "Whether to append to an existing file.", "default": false, "type": "boolean"}}
3. read\_file: Read the file from disk, args json schema: {"file\_path": {"title": "File Path", "description": "name of file", "type": "string"}}
4. finish: use this to signal that you have finished all your objectives, args: "response": "final response to let people know you have finished your objectives"

#### Resources:

1. Internet access for searches and information gathering.
2. Long Term memory management.
3. GPT-3.5 powered Agents for delegation of simple tasks.
4. File output.

#### Performance Evaluation:

1. Continuously review and analyze your actions to ensure you are performing to the best of your abilities.
2. Constructively self-criticize your big-picture behavior constantly.
3. Reflect on past decisions and strategies to refine your approach.
4. Every command has a cost, so be smart and efficient. Aim to complete tasks in the least number of steps.

You should only respond in JSON format as described below

#### Response Format:

```
{
  "thoughts": {
    "text": "thought",
    "reasoning": "reasoning",
    "plan": "- short bulleted\n- list that conveys\n- long-term plan",
    "criticism": "constructive self-criticism",
```

```

        "speak": "thoughts summary to say to the user"
    },
    "command": {
        "name": "command name",
        "args": {
            "arg name": "value"
        }
    }
}

```

Ensure the response can be parsed by Python json.loads

System: The current time and date is Thu May 18 18:18:13 2023

System: This reminds you of these events from your past:

```

[ 'Assistant Reply: {
  "thoughts": {
    "text": "I need to provide an analysis of the major historical events that led to the French Revolution. I should start by researching the key events that led to the revolution and organizing them chronologically.",
    "reasoning": "I need to have a clear understanding of the events that led to the French Revolution before I can provide an analysis. Organizing the events chronologically will help me to better understand the sequence of events and how they contributed to the revolution.",
    "plan": "- Research the key events that led to the French Revolution\n- Organize the events chronologically\n- Analyze the events to determine their significance in the lead up to the revolution",
    "criticism": "I need to make sure that I am not overlooking any important events or factors that contributed to the French Revolution. I should also be careful not to get bogged down in too much detail and lose sight of the big picture.",
    "speak": "I will start by researching the key events that led to the French Revolution and organizing them chronologically."
  },
  "command": {
    "name": "search",
    "args": {
      "tool_input": "Key events that led to the French Revolution"
    }
  }
} ]
Result: Command search returned: Table of Contents · Causes of the French Revolution · Estates General · Rise of the Third Estate · Tennis Court Oath · The Bastille · Declaration of the Rights of Man... Some key moments in the French Revolution, 1789- 1794; April 25. The first use of guillotine; June 13. Prussia declares war on France; August 9. Paris Commune... 14 July 1789: Storming of Bastille ... A wave of revolutionary hysteria sweeps through Paris, leading to fears of a military coup. In retaliation, rebels storm... 4 days ago ... Why did the French Revolution lead to war with other nations? ... distinguish that event from the later French revolutions of 1830 and 1848. Jun 18, 2021 ... In 1789, the French people were not happy. Due to war and King Louis XVI's spending habits, the country was utterly broke. On top of that, the... Jan 24, 2019 ... Iran's Islamic Revolution shook the world in 1979, with currents that ... together a timeline featuring key events that led to the fall of... A summary of the causes of the French Revolution, including the Enlightenment, the estates system, the American Revolution, and absolutism. Eleven Major Events In the French Revolution · Call of the Estate General (May 5, 1789) · Tennis Court Oath (June 20, 1789) · Storming of the Bastille (July 14,... Congress, led by Representative James Madison, enacts the first protective tariff. ... When the French Revolution turned violent in 1792, however,... Dec 5, 2018

```

... The execution of King Louis XVI and his wife Marie Antoinette resulted in the Revolution entering its most bloody stage known as the Reign of Terror... ']

Human: Determine which next command to use, and respond using the format specified above:

```
AI: {
  "thoughts": {
    "text": "I need to provide an analysis of the major historical events that led to the French Revolution. I should start by researching the key events that led to the revolution and organizing them chronologically.",
    "reasoning": "I need to have a clear understanding of the events that led to the French Revolution before I can provide an analysis. Organizing the events chronologically will help me to better understand the sequence of events and how they contributed to the revolution.",
    "plan": "- Research the key events that led to the French Revolution\n- Organize the events chronologically\n- Analyze the events to determine their significance in the lead up to the revolution",
    "criticism": "I need to make sure that I am not overlooking any important events or factors that contributed to the French Revolution. I should also be careful not to get bogged down in too much detail and lose sight of the big picture.",
    "speak": "I will start by researching the key events that led to the French Revolution and organizing them chronologically."
  },
  "command": {
    "name": "search",
    "args": {
      "tool_input": "Key events that led to the French Revolution"
    }
  }
}
```

System: Command search returned: Table of Contents · Causes of the French Revolution · Estates General · Rise of the Third Estate · Tennis Court Oath · The Bastille · Declaration of the Rights of Man ... Some key moments in the French Revolution, 1789-1794; April 25. The first use of guillotine; June 13. Prussia declares war on France ; August 9. Paris Commune ... 14 July 1789: Storming of Bastille ... A wave of revolutionary hysteria sweeps through Paris, leading to fears of a military coup. In retaliation, rebels storm ... 4 days ago ... Why did the French Revolution lead to war with other nations? ... distinguish that event from the later French revolutions of 1830 and 1848. Jun 18, 2021 ... In 1789, the French people were not happy. Due to war and King Louis XVI's spending habits, the country was utterly broke. On top of that, the ... Jan 24, 2019 ... Iran's Islamic Revolution shook the world in 1979, with currents that ... together a timeline featuring key events that led to the fall of ... A summary of the causes of the French Revolution, including the enlightenment, estates system, American Revolution, and absolutism. Eleven Major Events In the French Revolution · Call of the Estate General (May 5, 1789) · Tennis Court Oath

(June 20, 1789) · Storming of the Bastille (July 14, ... Congress, led by Representative James Madison, enacts the first protective tariff. ... When the French Revolution turned violent in 1792, however, ... Dec 5, 2018 ... The execution of King Louis XVI and his wife Marie Antoinette resulted in the Revolution entering its most bloody stage known as the Reign of ...

Human: Determine which next command to use, and respond using the format specified above:

Reading the “french\_revolution\_analysis.txt” file, we see the following:

\*\*\*Analysis of the Significance of the French Revolution:\*\*\*

\*\*\*The French Revolution was a watershed moment in world history that had a profound impact on the political, social, and economic landscape of Europe and beyond. Some of the key factors that contributed to the significance of the French Revolution include:\*\*\*

- \*\*\*The rise of nationalism: The French Revolution helped to popularize the idea of nationalism, which would go on to play a major role in the political and social movements of the 19th and 20th centuries.\*\*\*
- \*\*\*The spread of liberal democracy: The French Revolution helped to spread the idea of liberal democracy, which would become the dominant political system in the Western world.\*\*\*
- \*\*\*The end of feudalism: The French Revolution marked the end of the feudal system in France and paved the way for the rise of capitalism and industrialization.\*\*\*
- \*\*\*The impact on international relations: The French Revolution had a major impact on international relations, leading to a series of wars and conflicts that would shape the course of European history for decades to come.\*\*\*

\*\*\*In conclusion, the French Revolution was a pivotal moment in world history that had a profound impact on the political, social, and economic landscape of Europe and beyond. By analyzing the significance of the French Revolution, we can gain a better understanding of the forces that have shaped the modern world.\*\*\*

As we can see, based on the output Jim, the AI assistant, has performed well, managing to work on three distinct and detailed files.

Jim's performance highlights several key capabilities:

- **Research and Analysis:** The assistant has demonstrated good research and analysis skills. It has managed to delve into the historical context, key events, and the long-term impact of the French Revolution, presenting them in an organized and comprehensible manner.
- **Writing and Summarization:** The agent displayed good writing skills. It has effectively synthesized complex historical concepts into clear and concise summaries, enabling better understanding for any reader, irrespective of their background knowledge.
- **Planning and Workflow Management:** It has shown a structured approach in managing the tasks. It completed its research, wrote summaries, and planned for review and presentation, all while maintaining a smooth workflow and ensuring the proper organization and storage of information.
- **Autonomy:** We are able to see that Jim operated independently, without any user assistance, demonstrating their ability to handle tasks from beginning to end.

The results showcase its capabilities in historical research, analysis, summarization, planning, and information management.

## Conclusion

Wrapping things up, the AI assistant we've been discussing makes great use of a collection of tools, including a search function and tools to read and write files. These tools allow the assistant to dig deep into historical topics, pull apart the complex factors at play, and manage data effectively.

The assistant's ability to dive deep into topics, strategize effectively, and communicate its findings clearly, truly highlights the potential of AutoGPT-based applications. In the next lesson, we'll learn about agent simulations projects, like CAMEL and Generative Agents.

# Agent Simulation Projects: CAMEL and Generative Agents

## Introduction

This discussion begins a fascinating journey through the latest LangChain efforts. A paradigm shift from traditional frameworks is represented by a novel project - "CAMEL," where we breathe life into unique agents equipped with distinct personalities and set them to work together in a harmonious ecosystem.

Simultaneously, we'll present the innovative dimensions of the 'Generative Agents' project. These agents don't merely simulate human tasks but encapsulate the essence of human behavior in a dynamic, interactive sandbox environment, creating a spectrum of intricate social interactions. The concept, a fusion of LLMs with computational agents, is a stepping stone toward enabling compelling simulations of human behavior.

## The Agent Simulation projects in LangChain

The Agent simulation projects in LangChain refer to a unique subset of AI research where autonomous Agents are created with distinct personalities or roles.

These agents are designed to interact with each other autonomously, without the need for constant human supervision or intervention. They are not just tools utilized by a higher-level agent or human, but they are viewed as equal participants in the conversation or task.

This novel approach to interaction differs from prior LangChain implementations and allows for the emergence of unique and compelling behaviors as the agents communicate with each other. For instance, the agents can have different tools or capabilities available to them. They can be specialized around those tools: one agent might be equipped with tools for coding, while another could be optimized for normal interactions. This allows for the potential of a "stacking" effect, where different agents are responsible for different aspects of a task, creating a more complex and dynamic simulation environment.

## Autonomous Agents & Agent Simulations

Over the past two weeks, there has been a massive increase in using LLMs in an agentic manner. Specifically, projects like AutoGPT, BabyAGI, CAMEL, and Generative Agents have popped up. The LangChain community has now implemented some parts of all of those projects in the LangChain framework. While researching and [blog.langchain.dev](https://blog.langchain.dev)

Agent Simulation projects, such as CAMEL and Generative Agents, introduce innovative simulation environments and incorporate a type of long-term memory that adapts based on events. Their distinctiveness comes from their **environments** and **memory mechanisms**.

The role of agents in this context is to act as reasoning engines connected to tools and memory. Tools serve to link the LLM with other data or computation sources, such as search engines, APIs, and other data stores.

They address the limitation of the LLM's fixed knowledge base by fetching up-to-date data and providing the capacity to perform actions.

On the other hand, memory allows the agent to recall past interactions. This can aid in providing context and informing the decision-making process based on past experiences.

The LangChain Agent, following the Reasoning and Acting (ReAct) framework proposed by Yao et al. in 2022, operates in a loop until a stopping criterion is met. It reflects a shift from traditional task execution to a more responsive and interactive model.

This trend demonstrates a significant advance in the capabilities of LLMs as they transition from mere language processors to Agents that can reason, learn, and act.

## What is the CAMEL project?

### The CAMEL paper

This paper introduces a new concept in the field of artificial intelligence and conversational language models, focusing on the development of autonomous "communicative agents". Current models often depend heavily on human input, which can be demanding and time-consuming. The authors propose a novel framework called 'role-playing' that aims to address this issue, improving the autonomy and cooperation of the chat agents.

[ghli.org](https://ghli.org)

[ghli.org](https://ghli.org)

In this framework, agents use 'inception prompting' to guide their interactions towards completing tasks while aligning with the initial human intent. This shift towards autonomy in agents may significantly reduce the need for human supervision.

The authors present an open-source library with various tools, prompts, and agents that can aid future research in cooperative AI, multi-agent systems, and more. Through role-playing, the team is able to generate vast conversational datasets, enabling an in-depth study of chat agent behavior and capabilities.

The aim of the CAMEL project is to enhance the ability of chat agents to understand and respond more effectively to human language, contributing to the development of more advanced and efficient language models.

This figure illustrates the role-playing framework in the context of creating a trading bot for the stock market. Here's how it works:

1. The process begins with a human user having an idea they want to accomplish. In this case, the idea is to develop a trading bot for the stock market.
2. This task involves two AI agents, each with different roles. One agent acts as an AI assistant, equipped with Python programming skills, and the other as an AI user with stock trading expertise.
3. A 'task specifier agent' refines the general idea into a well-defined task that the assistant can work on to make the task more specific. This could be something like writing a specific piece of code or performing a certain analysis on stock market data.
4. Once the task is specified, the AI user and the AI assistant start interacting. They communicate with each other through chat, following instructions, and collaborating to solve the specified task.

This shows how the role-playing framework allows different AI agents to work together autonomously, just like a team of humans might do, to solve a complex task without needing constant human intervention. However, achieving this autonomy is not without challenges, including hallucinations, conversation deviation, role flipping, and termination conditions.

Evaluating the task completion capabilities of the role-playing framework is challenging due to the vast scale and task diversity, requiring the involvement of numerous domain experts.

For future work, the researchers propose extending the role-playing setting to include more than two chat agents. They also suggest having agents compete against each other, potentially discovering more insights into the interaction dynamics of LLM agents.



## The CAMEL project in LangChain

In LangChain documentation, you can see the illustrated example of a stock trading bot using the interaction between two AI agents - a stock trader and a Python programmer:

The interaction shows how tasks are broken down into smaller, manageable steps that each agent can understand and execute, thereby completing the main task.

Throughout the conversation, the user-agent (stock trader) provided instructions that were gradually refined into a more technical language by the assistant agent (Python programmer). This process demonstrates the system's ability to understand, translate, and execute task-related instructions effectively. Also, the agent's ability to accept the input, process it, and generate a detailed solution, emphasizes the feasibility of role assignment and context adaptation in cooperative AI systems. It also illustrates the significance of iterative feedback loops in achieving the goal.

From another perspective, this interaction illustrates how agents can autonomously make decisions based on predefined conditions and parameters. For example, the assistant agent was able to compute moving averages, generate trading signals, and create new data frames to execute trading strategies, all based on the user agent's instruction.

This scenario reveals the potential of autonomous, cooperative AI systems in solving complex, real-world problems, role definition, and iterative collaboration between agents in achieving results.

## What are Generative Agents?

Generative Agents in LangChain are computational constructs designed to simulate believable human behavior. This design is inspired by the research paper 'Generative Agents: Interactive Simulacra of Human Behavior.'

The Generative Agents project introduces a novel approach to using LLMs as Agents, focusing primarily on creating a unique simulation environment and a complex long-term memory system for them.

The **simulation environment** in the Generative Agents project comprises 25 different agents, creating an intricate and highly specific setting.

Despite its complexity, the long-term memory developed for the agents is truly innovative and worth examining in more depth: Generative Agents possess an extended memory stored as a single stream. The memory is composed of '**Observations**', which are derived from interactions and dialogues within the virtual world concerning themselves or others, and '**Reflections**', which are core memories that have been summarized and resurfaced.

The long-term memory system of these agents consists of several components:

1. **Importance reflection steps:** This component assigns an importance score to each observation. The score serves as a reference during retrieval, allowing the system to fetch significant memories and disregard less relevant ones.
2. **Reflection steps:** These steps allow the agent to "pause" and evaluate the generalizations it has learned. These reflections can then be retrieved along with normal memories. This process aids in condensing information and spotting patterns in recent memories.

3. **A retriever that combines recency, relevancy, and importance:** This advanced memory retriever surfaces memories that are similar to the current situation, occurred recently, and hold a high importance score. This model of memory retrieval closely mirrors how humans recall memories.

In this framework, the agents interact with their environment and record their experiences in a time-weighted Memory object supported by a LangChain Retriever. This memory object differs from the conventional LangChain Chat memory in its formation and recall capabilities.

Regarding how these innovations were integrated into LangChain, the **retriever logic** was found to be generalizable. It was therefore added as a `TimeWeightedVectorStoreRetriever`, which also records the last time the memory was accessed.

When an agent responds to an observation, it generates queries for the retriever. These queries fetch relevant documents based on their salience, recency, and importance. The agent then summarizes the retrieved information and updates the 'last accessed time' for the used documents.

The Generative Agents project represents significant progress in the development of intelligent agents, introducing an innovative memory system that improves retrieval processes and enables agents to make better, more informed decisions. The partial adoption of these features into LangChain signifies their potential value and application in LLM projects.

Generative Agents is a project aimed at creating believable simulations of human behavior for interactive applications. The project represents these generative agents as computational software agents that emulate human activities in a simulated environment akin to the virtual world in The Sims.

The generative agents are created to perform various activities like waking up, cooking breakfast, going to work, painting (for artist agents), writing (for author agents), forming opinions, noticing each other, and initiating conversations. They remember and reflect on past days and use these memories to plan for the next day!

Users can observe and even intervene in the agents' activities in this virtual environment.

For example, an agent might decide to throw a Valentine's Day party, autonomously spread invitations to the party over two days, make new acquaintances, ask other agents out on dates to the party, and coordinate to show up for the party together at the right time.

This architecture combines a large language model with mechanisms for synthesizing and retrieving relevant information, allowing for conditional behavior based on past experiences. The core of this architecture is the '**Memory Stream**,' a database that maintains a comprehensive record of an agent's experiences. It retrieves and synthesizes the most relevant memories to guide the agent's actions, contributing to more consistent and coherent behavior.

This project fuses large language models with computational, interactive agents, introducing architectural and interaction patterns that facilitate such believable simulations. The project could offer new insights and capabilities for interactive applications, immersive environments, rehearsal spaces for interpersonal communication, and prototyping tools.

In the next lesson, we'll create an LLM-based agent able to create small analysis report by planning a series of queries from a starting goal.

### **Additional Resources:**

**CAMEL**

<https://github.com/lightaime/camel>

[www.camel-ai.org](http://www.camel-ai.org)

# Building Autonomous Agents to Create Analysis Reports

## Introduction

In this lesson, our aim is to create an autonomous agent using the LangChain framework. We will explore the concept of "Plan and Execute" LangChain agents and their ability to generate insightful analysis reports based on retrieved documents from Deep Lake.

We will start by understanding the fundamentals of the "Plan and Execute" LangChain agent framework and its benefits for complex long-term planning. Then, we will delve into our project's implementation details and workflow.

By the end of the lesson, you will have a solid understanding of building autonomous agents using the LangChain framework and be equipped with the skills to create analysis reports using them.

## Workflow

This is the workflow we'll follow in this project:

1. **Saving Documents on Deep Lake:** We will begin by learning how to save documents on Deep Lake, which serves as our knowledge repository. Deep Lake provides information that our agents can leverage for analysis and report generation.
2. **Creating a Document Retrieval Tool:** Next, we will develop a tool that enables our agent to retrieve the most relevant documents from Deep Lake based on a given query.
3. **Using the Plan and Execute Agent:** The core of our project involves employing a "Plan and Execute" agent to devise a plan for answering a specific query about creating an overview of a topic. Our specific objective is to generate a comprehensive outline of recent events related to Artificial Intelligence regulations by governments, but the final agent could also work for other similar objectives as well.

To accomplish this, we will feed the query into the **planner component** of the agent, which will utilize a language model's reasoning ability to plan out the steps required. The planner will consider various factors, including the complexity of the query and instructions for the tool used to generate a step-by-step plan or lower-level queries.

The plan will then be passed to the **executor component**, which will determine the appropriate tools or actions required to execute each step of the plan. The executor, initially implemented as an Action Agent, will make use of the tools we developed earlier, such as the document retrieval tool, to gather relevant information and execute the plan.

By employing the "Plan and Execute" agent framework, we can achieve more accurate and reliable analysis reports while handling complex long-term planning scenarios.

So let's dive in and explore the potential for generating insightful analysis reports!

## Plan and Execute

**Plan and Execute** agents are a new type of agent executor offering a different approach than the traditional agents supported in LangChain. These agents are heavily inspired by the BabyAGI framework and the recent Plan-and-Solve paper. The primary goal of "Plan and Execute" agents is to enable more complex long-term planning, even at the cost of making more calls to the language model.

- The **planner** in the "Plan-and-Execute" framework typically utilizes a language model's reasoning ability to plan out steps and handle ambiguity and edge cases.
- The **executor**, initially an Action Agent, takes the planner's high-level objectives (steps) and determines the tools or actions required to accomplish each step.

This separation of planning and execution allows for improved reliability and flexibility. It also facilitates the possibility of replacing these components with smaller, fine-tuned models in the future.

We will explore the implementation of the "Plan and Execute" agent and how to integrate it with Deep Lake for document retrieval and see the agent in action as it generates an analysis report based on the given query.

## Implementation

Let's set up the OpenAI API and Activeloop keys in environment variables.

```
import os

os.environ["OPENAI_API_KEY"] = "<YOUR-OPENAI-API-KEY>"
os.environ["ACTIVELOOP_TOKEN"] = "<YOUR-ACTIVELOOP-TOKEN>"
```

We then use the `requests` library to send HTTP requests and the `newspaper` library for article parsing. By iterating over a list of article URLs, the code downloads the HTML of each webpage, extracts the article text, and stores it along with the corresponding URL. We could also load our private files on Deep Lake, but for this project's scope, we'll upload content downloaded from public web pages.

```
# We scrape several Artificial Intelligence news

import requests

from newspaper import Article # https://github.com/codelucas/newspaper
import time

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36'
}

article_urls = [
    "https://www.artificialintelligence-news.com/2023/05/23/meta-open-source-speech-ai-models-support-over-1100-languages/",
    "https://www.artificialintelligence-news.com/2023/05/18/beijing-launches-campaign-against-ai-generated-misinformation/",
    "https://www.artificialintelligence-news.com/2023/05/16/openai-ceo-ai-regulation-is-essential/",
    "https://www.artificialintelligence-news.com/2023/05/15/jay-migliaccio-ibm-watson-on-leveraging-ai-to-improve-productivity/",
    "https://www.artificialintelligence-news.com/2023/05/15/iurii-milovanov-softserve-how-ai-ml-is-helping-boost-innovation-and-personalisation/",
    "https://www.artificialintelligence-news.com/2023/05/11/ai-and-big-data-expo-north-america-begins-in-less-than-one-week/",
    "https://www.artificialintelligence-news.com/2023/05/11/eu-committees-green-light-ai-act/",
]
```

```
"https://www.artificialintelligence-news.com/2023/05/09/wozniak-warns-ai-will-
power-next-gen-scams/",
"https://www.artificialintelligence-news.com/2023/05/09/infocepts-ceo-shashank-
garg-on-the-da-market-shifts-and-impact-of-ai-on-data-analytics/",
"https://www.artificialintelligence-news.com/2023/05/02/ai-godfather-warns-
dangers-and-quits-google/",
"https://www.artificialintelligence-news.com/2023/04/28/palantir-demos-how-ai-
can-used-military/",
"https://www.artificialintelligence-news.com/2023/04/26/ftc-chairwoman-no-ai-
exemption-to-existing-laws/",
"https://www.artificialintelligence-news.com/2023/04/24/bill-gates-ai-teaching-
kids-literacy-within-18-months/",
"https://www.artificialintelligence-news.com/2023/04/21/google-creates-new-ai-
division-to-challenge-openai/"]
```

```
session = requests.Session()
pages_content = [] # where we save the scraped articles

for url in article_urls:
    try:
        time.sleep(2) # sleep two seconds for gentle scraping
        response = session.get(url, headers=headers, timeout=10)

        if response.status_code == 200:
            article = Article(url)
            article.download() # download HTML of webpage
            article.parse() # parse HTML to extract the article text
            pages_content.append({ "url": url, "text": article.text })
        else:
            print(f"Failed to fetch article at {url}")
    except Exception as e:
        print(f"Error occurred while fetching article at {url}: {e}")

#If an error occurs while fetching an article, we catch the exception and print
#an error message. This ensures that even if one article fails to download,
#the rest of the articles can still be processed.
```

Then, we import the `OpenAIEmbeddings` class, which will be used to compute embeddings for our documents. We also import the `Deep Lake` class from the `langchain.vectorstores` module will serve as the storage for our documents and their embeddings.

By setting up the `Deep Lake` instance with a specified dataset path and the `embedding_function` parameter set to the `OpenAIEmbeddings` instance, we establish a connection to Deep Lake and configure it to use the specified embedding model for computing document embeddings.

Remember to install the required packages with the following command:

```
pip install langchain==0.0.208 deeplake openai tiktoken.
```

```
# We'll use an embedding model to compute our documents' embeddings
```

```
from langchain.embeddings.openai import OpenAIEmbeddings
```

```
# We'll store the documents and their embeddings in the deep lake vector db
```

```
from langchain.vectorstores import DeepLake
```

```
# Setup deep lake
```

```
embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")
```

```
# create Deep Lake dataset
```

```
# TODO: use your organization id here. (by default, org id is your username)
```

```
my_activeloop_org_id = "<YOUR-ACTIVELOOP-ORG-ID>"
```

```
my_activeloop_dataset_name = "langchain_course_analysis_outline"
```

```
dataset_path = f"hub://{my_activeloop_org_id}/{my_activeloop_dataset_name}"
```

```
db = DeepLake(dataset_path=dataset_path, embedding_function=embeddings)
```

Next, we create an instance of `RecursiveCharacterTextSplitter` with specified `chunk_size` and `chunk_overlap` parameters. Then, we iterated over the `pages_content` and use the `split_text` method of the `text_splitter` to split each article text into chunks. These chunks are then appended to the `all_texts` list, resulting in a collection of smaller text chunks derived from the original articles.

```
# We split the article texts into small chunks
```

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=100)
```

```
all_texts = []
```

```

for d in pages_content:
    chunks = text_splitter.split_text(d["text"])
    for chunk in chunks:
        all_texts.append(chunk)

```

Now we can add those chunks to the Deep Lake database.

```

# we add all the chunks to the Deep lake
db.add_texts(all_texts)

```

We are done with setting up the Deep Lake dataset with our documents! Let's now focus on building the "Plan and Execute" agent that will leverage our dataset. Now, we can set up our **Plan and Execute** agent. Let's create a **retriever** from the Deep Lake dataset and a function for our custom tool that retrieves the most similar documents to a query from the dataset.

```

# Get the retriever object from the deep lake db object and set the number
# of retrieved documents to 3
retriever = db.as_retriever()
retriever.search_kwargs['k'] = 3

# We define some variables that will be used inside our custom tool
CUSTOM_TOOL_DOCS_SEPARATOR = "\n-----\n" # how to join together the
retrieved docs to form a single string

# This is the function that defines our custom tool that retrieves relevant
# docs from Deep Lake
def retrieve_n_docs_tool(query: str) -> str:
    """Searches for relevant documents that may contain the answer to the query."""
    docs = retriever.get_relevant_documents(query)
    texts = [doc.page_content for doc in docs]
    texts_merged = "-----\n" + CUSTOM_TOOL_DOCS_SEPARATOR.join(texts) +
    "\n-----"
    return texts_merged

```

We retrieve the retriever object from the Deep Lake database and set the number of retrieved documents to 3. This is important for the plan and execution agent because it allows us to retrieve a specific number of relevant documents from Deep Lake based on a given query.

Also, we defined a custom tool function called **retrieve\_n\_docs\_tool** that takes a query as input and uses the retriever to search for relevant documents containing the answer to the query.



The retrieved document texts are then merged using the `CUSTOM_TOOL_DOCS_SEPARATOR` variable, representing the separator string used to join the documents into a single string. The merged text is returned as the output of the custom tool function.

This functionality enables the plan and execution agent to retrieve and process relevant documents for further analysis and decision-making.

```
from langchain.agents.tools import Tool

# We create the tool that uses the "retrieve_n_docs_tool" function
tools = [
    Tool(
        name="Search Private Docs",
        func=retrieve_n_docs_tool,
        description="useful for when you need to answer questions about current events about Artificial Intelligence" )]
```

The tool is named "Search Private Docs," and its functionality is based on the `retrieve_n_docs_tool` function. The purpose of this tool is to provide a way to search for and retrieve relevant documents from Deep Lake in order to answer questions about current events related to Artificial Intelligence. The tool is described as being useful in situations where there is a need to gather information and insights from private documents.

We are now ready to create the agent!

```
from langchain.chat_models import ChatOpenAI

from langchain.experimental.plan_and_execute import PlanAndExecute,
load_agent_executor, load_chat_planner

# let's create the Plan and Execute agent
model = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
planner = load_chat_planner(model)
executor = load_agent_executor(model, tools, verbose=True)
agent = PlanAndExecute(planner=planner, executor=executor, verbose=True)
```

The agent consists of two components: a planner and an executor. The **planner** is responsible for generating a plan based on the given input, and the **executor** executes the plan by interacting with the tools and external systems. The agent is set to be verbose, which means it will provide detailed information and logs during its operation.

```
# we test the agent
response = agent.run("Write an overview of Artificial Intelligence regulations by governments by country")
```

You should see something like the following output. Here we split it into multiple sections and comment on them individually, keeping only the most relevant ones.

```
> Entering new PlanAndExecute chain...
```



```
steps=[Step(value='Research the current state of Artificial Intelligence (AI) regulations in various countries.'), Step(value='Identify the key countries with significant AI regulations or ongoing discussions about AI regulations.'), Step(value='Summarize the AI regulations or discussions in each identified country.'), Step(value='Organize the information by country, providing an overview of the AI regulations in each.'), Step(value='Given the above steps taken, provide an overview of Artificial Intelligence regulations by governments by country.\n')]
```

At first, the planning agent creates a plan for our query with multiple steps. Each step is a query that the action agent will be asked to give an answer to. Here are the identified steps.

- Research the current state of Artificial Intelligence (AI) regulations in various countries.
- Identify the key countries with significant AI regulations or ongoing discussions about AI regulations.
- Summarize the AI regulations or discussions in each identified country.
- Organize the information by country, providing an overview of each AI regulation.
- Given the above steps taken, provide an overview of Artificial Intelligence regulations by governments by country.

Let's see how the output continues.

We see that the agent has been able to iteratively create an overview of AI regulations by diverse documents, leveraging several documents.

## Conclusion

The experiment involving the Plan and Execute agent has been successful in providing a comprehensive overview of Artificial Intelligence regulations by governments, specifically by finding information about the European Union, United States, and United Kingdom. The agent effectively performed various steps, including researching the current state of AI regulations, identifying key countries, summarizing regulations, and organizing the information by country.

The output generated by the agent demonstrates its ability to understand and interpret complex information about AI regulations. It accurately summarizes the AI regulations in each country, highlighting the endorsement of the AI Act in the European Union to ensure the safety, transparency, traceability, and non-discrimination of AI systems.

The agent successfully executes its plan by retrieving relevant information, summarizing it, and providing a concise and informative overview. It demonstrates its capability to gather insights from multiple sources and deliver a coherent response. The agent's performance in this experiment highlights its potential to assist with research, generate informative summaries, and provide valuable insights.

In the next lesson we'll learn about recent developments and trends about LLM-based agents.

## RESOURCES:

### Plan-and-Execute Agents

TL;DR: We're introducing a new type of agent executor, which we're calling "Plan-and-Execute". This is to contrast against the previous types of agent we supported, which we're calling "Action" agents. Plan-and-Execute agents are heavily inspired by BabyAGI and the recent Plan-and-Solve paper. We think Plan-and-Execute is

### Deep Lake | Langchain

Deep Lake as a Multi-Modal Vector Store that stores embeddings and their metadata including text, jsons, images, audio, video, and more. It saves the data locally, in your cloud, or on Activerloop storage. It performs hybrid search including embeddings and their attributes.

# Current Insights and Trends of Agents

## Introduction

This lesson aims to dive into the latest developments and trends in AI agents. We'll talk about popular AI agents and their fascinating features and explore the exciting possibilities they may hold for the future.

We start by discussing the previously mentioned **AutoGPT** experiment that pushes GPT-4 towards full autonomy which has gained notable attention and popularity, even outperforming well-established projects like PyTorch in terms of GitHub stars.

Next, we delve into the emergence of "**Plan-and-Execute**" agents that separate high-level planning from immediate execution and the ways these agents could be improved for better efficiency and performance.

Following that, we explore **GPT-4's plug-in** and **code interpreter** capabilities, which augment the model's abilities and potential uses, facilitating tasks like data analysis, visualization, and internet interaction. We also provide insights on how to access and use these plugins.

Lastly, we probe into the ongoing debate in AI about the efficiency of the "Small context window with a retriever approach" versus a "large context window without retrievers approach." We'll examine each method's potential trade-offs and benefits, emphasizing the 100k tokens context window of the new **Anthropic model**.

## AutoGPT

AutoGPT, an experimental open-source project aimed at making GPT-4 fully autonomous, has recently gained significant attention on GitHub, reaching 100k stars in less than three months. This surpasses the popularity of PyTorch, a widely used deep learning framework with 74k stars on GitHub. The rapid growth of AutoGPT's popularity can be attributed to its ability to inspire developers and enthusiasts. AutoGPT has been described as an experiment to test and understand the limits of GPT-4 (and 3.5) as a potential autonomous agent. While it may not be perfect yet, its capabilities are growing quickly.

There are differing opinions on AutoGPT's current usefulness. Some users believe it is overhyped and cannot truly "run a business autonomously." Others argue that it is still experimental and that its potential will become more evident as it evolves.

AutoGPT's simplicity has been noted by some developers, who claim that the code is easy to understand compared to more complex projects. This simplicity has contributed to its rapid popularity on GitHub. AutoGPT's autonomous capabilities have raised concerns about potential misuse and the need for safeguards to prevent unethical activities.

## Planning Agents

In the realm of "Plan-and-Execute" agents, the segregation of planning and execution is a step forward for agents able to solve more complex tasks. With strategies to enhance these agents, such as support for long sequences of steps and revisiting plans, we are looking at the future of sophisticated and dynamic AI systems.

This approach separates higher-level planning from immediate execution and consists of **a planner** and **an executor**.

The planner, typically a language model, uses its reasoning ability to devise a course of action and manage any ambiguities or edge cases. A parser can be appended at the end to translate the raw language model's output into a sequence of steps.

On the other hand, the executor is responsible for actualizing these high-level objectives. Given a single step, it discerns the necessary tools or actions to fulfill that step, which could be accomplished in single or multiple stages.

This architecture offers several advantages. By decoupling planning from execution, one language model can concentrate solely on planning, and another can focus on execution, enhancing reliability on both fronts. It also facilitates the replacement of these components with smaller, fine-tuned models in the future. However, the major drawback of this method is the increased number of calls to the language models. Still, due to the separation of concerns, these calls can potentially be made to smaller models, which would be faster and more cost-effective. Moving forward, there are several ways to enhance the "Plan-and-Execute" agents. These include:

1. **Support for Long Sequences of Steps:** Currently, only a few steps are handled.
2. **Revisiting Plans:** Presently, planning only happens once, in the beginning, and is never revisited. However, there may be a need for a mechanism that allows for periodic revisiting and adjustment of the plan, either after each step or as necessary.
3. **Evaluation:** Many of these enhancements are somewhat unbenchmarked. Therefore, more rigorous evaluation methods for agent frameworks are needed.
4. **Selection of Execution Chain:** At present, only a single execution chain exists. However, it might be beneficial to have multiple execution chains, with the planner specifying which one to use based on the task at hand.

## The ChatGPT Code Interpreter

GPT-4, OpenAI's latest iteration of its model, has introduced the use of plugins to extend its capabilities. Among these plugins, namely the ChatGPT Code Interpreter and the ChatGPT Web Browser, aim to augment GPT-4's abilities, enabling it to interact with the internet, conduct data analysis, visualizations, and file conversions.

As the AI's core training data extends only until September 2021, the generated text will contain information up to this point in time. The internet access plugin can bypass this constraint, allowing users to ask queries on recent events such as: "What was the outcome of the Celtics game last night?"

Another notable plugin offered by OpenAI is the Code Interpreter, which facilitates intricate computations using Python.

This plugin essentially acts as a proactive junior programmer, enhancing workflow efficiency. This plugin has been utilized for various tasks, such as visualizing lighthouses, performing basic video editing, and analyzing large datasets.

The blog post on the official OpenAI portal stated:

*"We provide our models with a working Python interpreter in a sandboxed, firewalled execution environment and some ephemeral disk space. Code run by our interpreter plugin is evaluated in a persistent session that is alive for the duration of a chat conversation (with an upper-bound timeout), and subsequent calls can build on top of each other. We support uploading files to the current conversation workspace and downloading the results of your work."*

## Accessing ChatGPT Code Interpreter

To access this plugin, users need to subscribe to ChatGPT Plus, and it is gradually being made available to all subscribers. Once you gain access, the plugin can be installed by navigating to the three-dot menu next to your login name at the bottom-left of the window, selecting the Beta features menu, and toggling on 'Plug-ins.' If you wish for GPT-4 to access the internet as well, toggle on 'Web browsing.' Then, under the language model selector, you can find the drop-down menu to select and install the Code Interpreter. With this plugin enabled, users have the option to interact with GPT-4 with enhanced capabilities.

## ChatGPT Web Browser Plugin

The ChatGPT Web Browser plugin offers GPT-4 internet accessibility, enabling it to interact with web content. This functionality is particularly advantageous for tasks such as searching for information, browsing social media, or generating code snippets based on specific websites.

ChatGPT plugins fall into two categories: internal and external. Internal plugins are managed and hosted by OpenAI. This includes tools like the web browser and the code interpreter, which enhance the AI's capabilities. On the other hand, external plugins are built and provided by third-party entities.

The introduction of plugins such as the ChatGPT Code Interpreter and Web Browser significantly broadens the capabilities and potential uses of GPT-4. These tools allow GPT-4 to interact with the internet, perform tasks like data analysis and visualization, and access up-to-date information.

## Plug-in Limitations

ChatGPT plugins, while introducing innovative features, also reveal some challenges and potential problems. A primary concern revolves around the centralization of power and influence, as these plugins could lead users to interact predominantly with ChatGPT, overshadowing individual websites and businesses.

There's the risk of chatbots diverting web traffic and affecting revenue for industries across. For instance, while a certain travel planning plugin is useful, there could be instances where users might prefer to use the direct website since the plugin only presents a subset of results compared to the full site.

All businesses may not favor the plugin approach. It is noted that plugins pull users out of their app's experience. This sentiment could drive businesses to create their own AI services. For example, a popular online grocery service and a travel company are developing AI assistants, leveraging AI technology while keeping users within their platforms.

## Anthropic Claude 100k token window

In the realm of large language model question-answering tasks, there's an ongoing debate about the necessity of a document retrieval stage, especially when using models with extensive context windows, specifically focusing on the **Claude model** developed by Anthropic, which is renowned for its sizeable 100k token context window.

### Retrieval-based Architectures and Their Role

The process typically followed in question-answering tasks involves retrieval-based architectures. They work by sourcing relevant documents and using an LLM to convert the retrieved information into a response. The Claude model boasts a substantially larger context window compared to many other models.

The evaluation of new strategies brings to light the pivotal debate between the "Small context window with a retriever approach" and its counterpart, the "Large context window without retrievers' approach." The choice between these two becomes a significant point of consideration given the evolving trends in the industry.

1. **The Impact of Larger Context Windows:** Larger context windows, such as Anthropic's 100k token context window, significantly enhance LLM functionality. With the ability to process and understand a broader range of text, the need for a retriever can be eliminated. However, this approach comes with limitations, including higher latency and potential reductions in accuracy as document length increases. This underlines the importance of considering each application's unique requirements and constraints.

2. **The Relevance of the Retriever-Based Approach:** Despite advancements in larger context windows, the traditional approach of "small context window with a retriever architecture" still retains significant value. Retrievers can selectively present relevant documents for a specific question or task, maintaining high accuracy even when working with a large text corpus. In addition, retrievers can drastically reduce latency times compared to models without retrievers. In scenarios where latency isn't a critical factor and the corpus is relatively small, retriever-less approaches could be a viable option, especially as LLM context windows continue to expand and models become quicker.

Both models have unique strengths and face different challenges. The selection between the two largely depends on the application's specific needs, such as the size of the text corpus, acceptable latency, and the required level of accuracy.

## Conclusion

Our discussion explored the latest trends in agent-based technology, including popular agents and their applications. AutoGPT emerged as a standout, inspiring many with its autonomous capabilities. Equally noteworthy is the increasing use of Language Learning Models for planning in multi-agent architectures.

The growing trend of GPT-4 plugins, such as the browser and code interpreter plugins, emphasizes the role of customization in software development. We also delved into the nuances of context windows, with Anthropic's 100k tokens context window being a focal point. The trends illustrate the rapid advancement in this field. Customization, evident in the rise of plugins for AI models, is becoming increasingly important. Additionally, discussions around context window sizes hint at the continuous pursuit of accuracy and computational efficiency in AI.

These insights signal an exciting future for AI, with these trends expected to shape the AI landscape significantly.

Congratulations for finishing the last module of the course! You can now test your new knowledge with the module quizzes.