Module 3
Multi-stage Reasoning

Learning Objectives

By the end of this module you will:

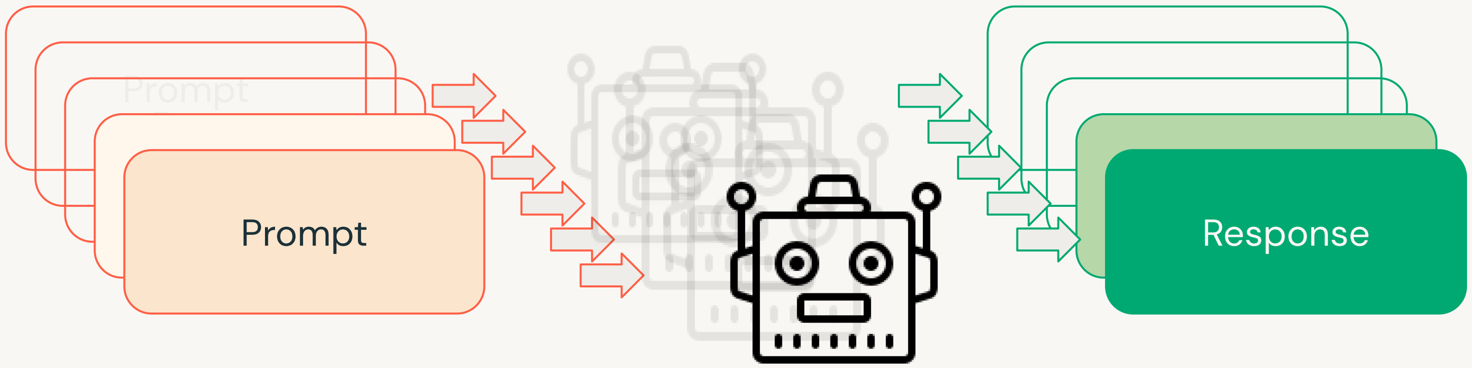
- Describe the flow of LLM pipelines with tools like LangChain.
- Apply LangChain to leverage multiple LLM providers such as OpenAI and Hugging Face.
- Create complex logic flow with agents in LangChain to pass prompts and use logical reasoning to complete tasks.

LLM Limitations

LLMs are great at single tasks… but we want more!

LLM Tasks vs. LLM-based Workflows

LLMs can complete a huge array of challenging tasks.



Summarization

Sentiment analysis

Translation

Zero-shot classification

Few-shot learning

Conversation / chat

Question-answering

Table question-answering

Token classification

Text classification

Text generation

...

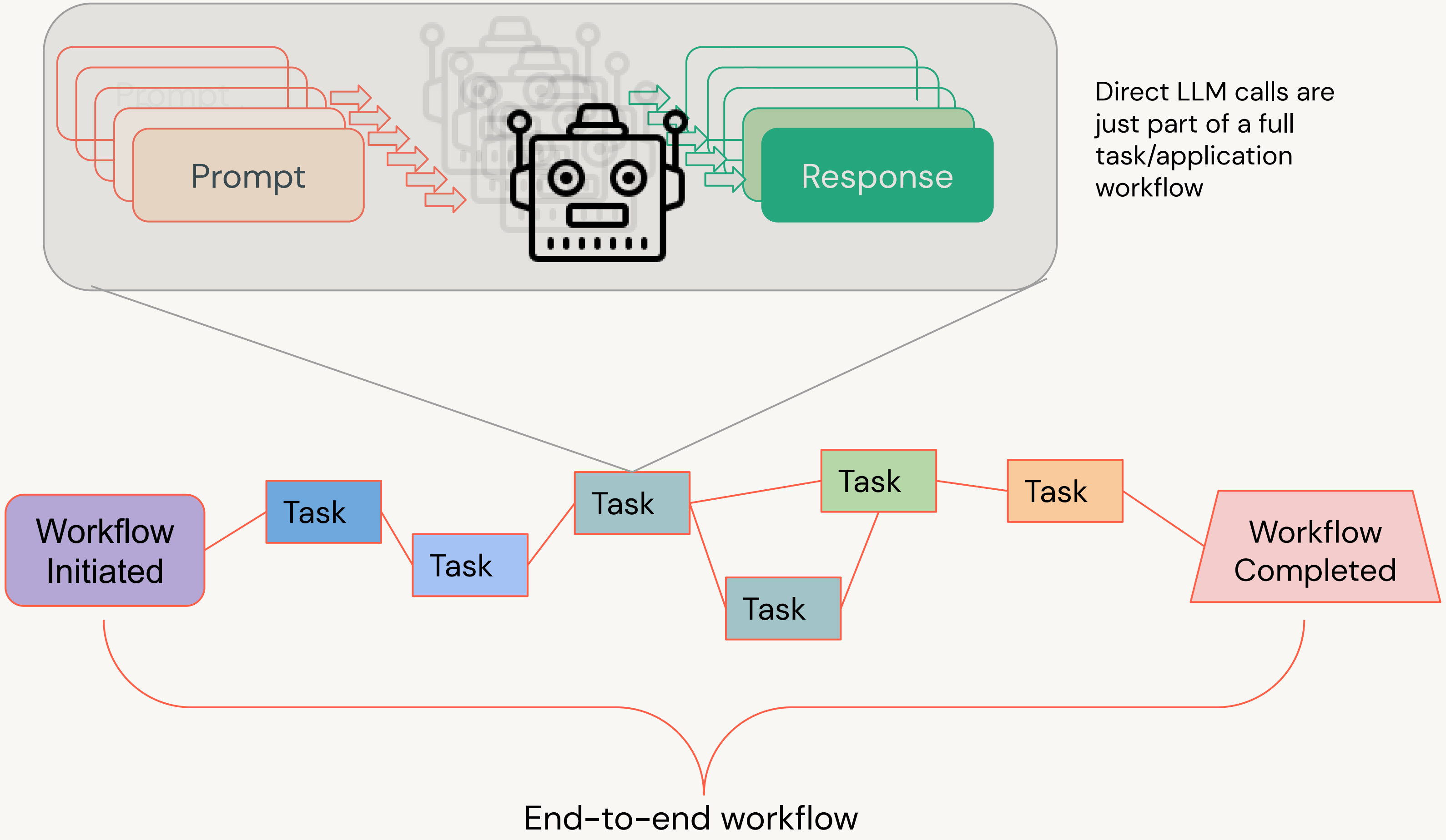
Image source: mrvian.com

LLM Tasks vs. LLM-based Workflows

Typical applications are more than just a prompt-response system.

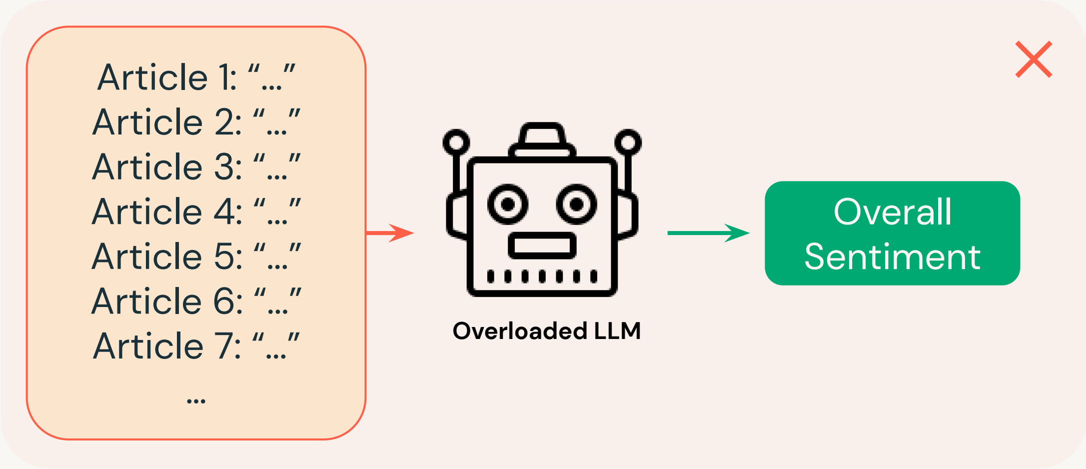
Tasks: Single interaction with an LLM

Workflow: Applications with more than a single interaction



Summarize and Sentiment

Example multi-LLM problem: get the sentiment of many articles on a topic

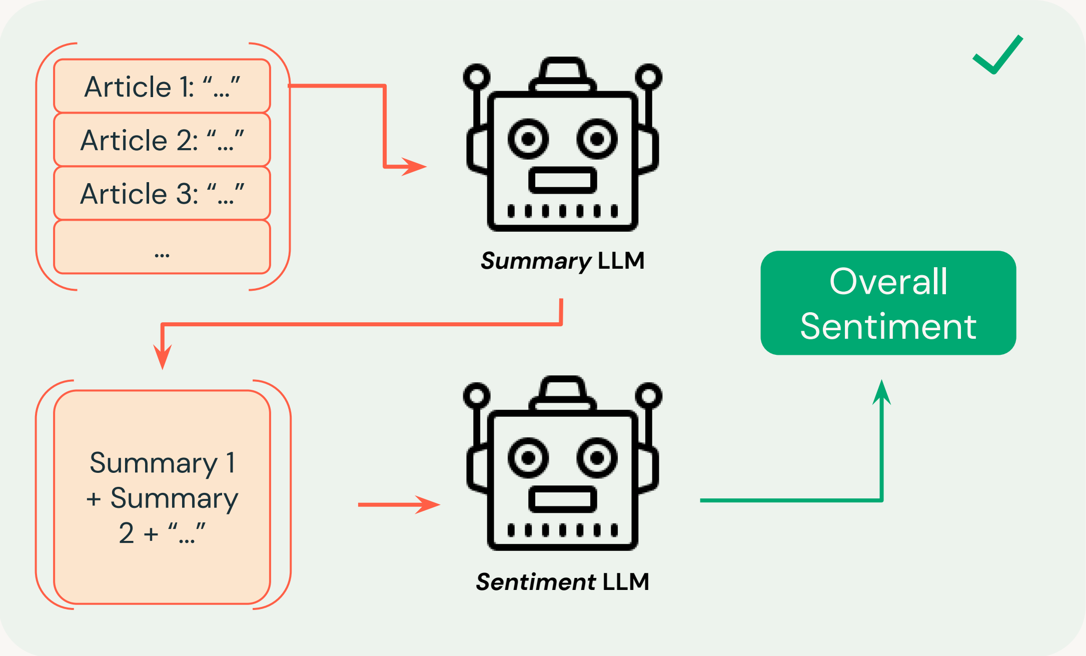


Initial solution

Put all the articles together and have the LLM parse it all

Issue

Can quickly overwhelm the model input length

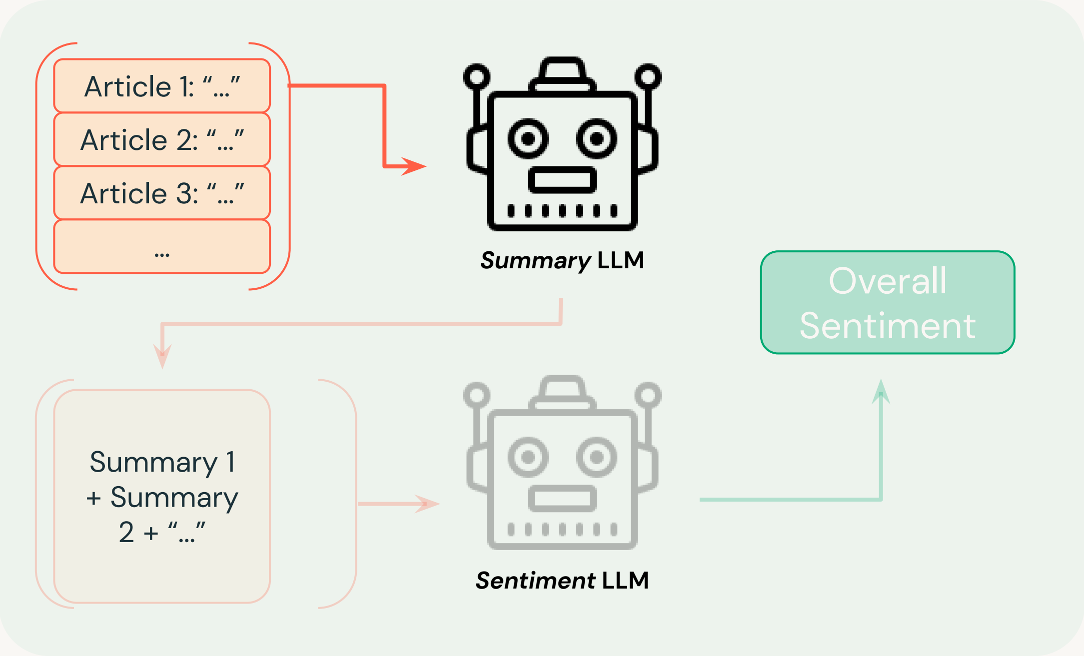


Better solution

A two-stage process to first summarize, then perform sentiment analysis.

Summarize and Sentiment

Step 1: Let's see how we can build this example.



Goal:

Create a reusable workflow for multiple articles.

For this we'll focus on the first task first.

How do we make this process systematic?

Prompt Engineering:

Crafting more elaborate prompts to get the most out of our LLM interactions

Prompt Engineering – Templating

Task: Summarization

```
# Example template for article summary
# The input text will be the variable {article}
summary_prompt_template = """
Summarize the following article, paying close attention to emotive phrases: {article}
Summary: """
```

{article} is the variable in the prompt template.

Prompt Engineering – Templating

Use generalized template for any article

```
# Example template for summarization
# The input text will be the variable {article}
summary_prompt_template = """
Summarize the following article, paying close attention to emotive phrases: {article}
Summary: """
###############################################################################################
# Now, construct an engineered prompt that takes two parameters: template and a list of input variables
(article)
summary_prompt = PromptTemplate(template=summary_prompt_template, input_variables=["article"])
```

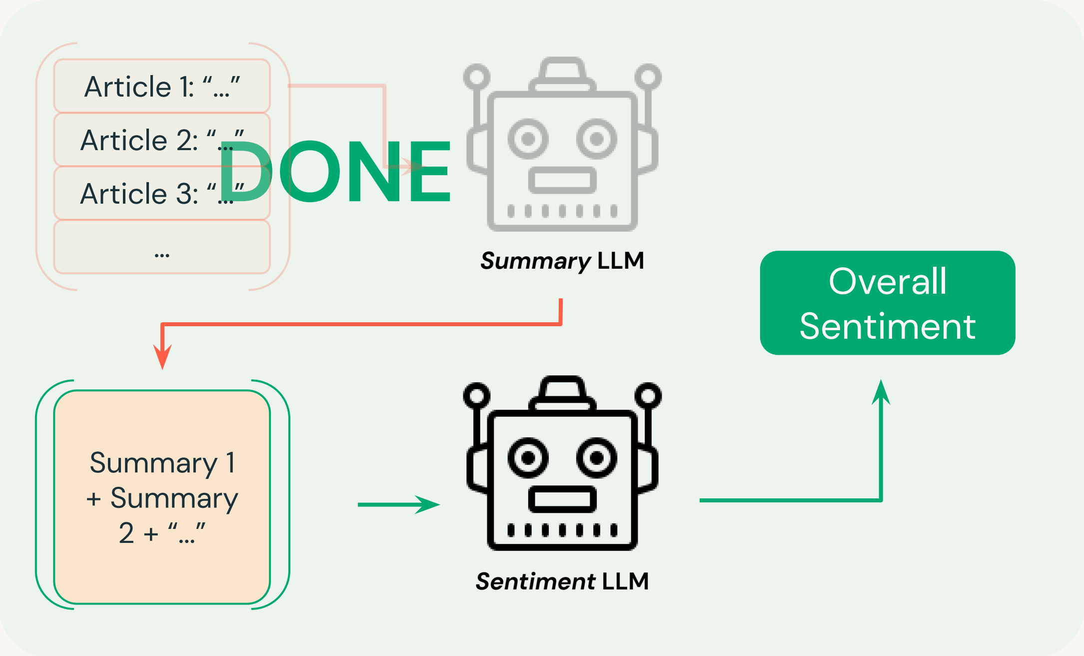
Prompt Engineering – Templating

We can create many prompt versions and feed them into LLMs

```
# Example template for summarization
# The input text will be the variable {article}
summary_prompt_template = """
Summarize the following article, paying close attention to emotive phrases: {article}
Summary: """
##############################################################################################
# Now, construct an engineered prompt that takes two parameters: template and a list of input variables
(article)
summary_prompt = PromptTemplate(template = summary_prompt_template, input_variables=["article"])
##############################################################################################
# To create an instance of this prompt with a specific article, we pass the article as an argument.
summary_prompt(article=my_article)
# Loop through all articles
for next_article in articles:
   next_prompt = summary_prompt(article=next_article)
   summary = llm(next_prompt)
```

Multiple LLM interactions in a sequence

Chain prompt outputs as input to LLM



Now we need the **output** from our new engineered prompts to be the **input** to the sentiment analysis LLM.

For this we're going to **chain** together these LLMs.

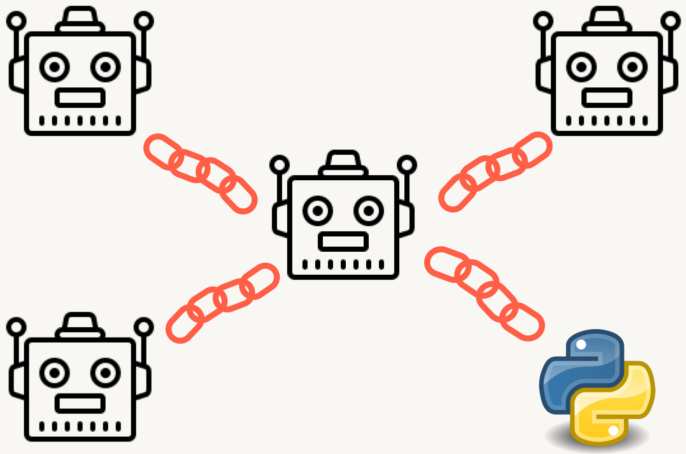
LLM Chains:

Linking multiple LLM interactions to build complexity and functionality

LLM Extension Libraries

LangChain

- Released in late 2022
- Useful for multi-stage reasoning, LLM-based workflows



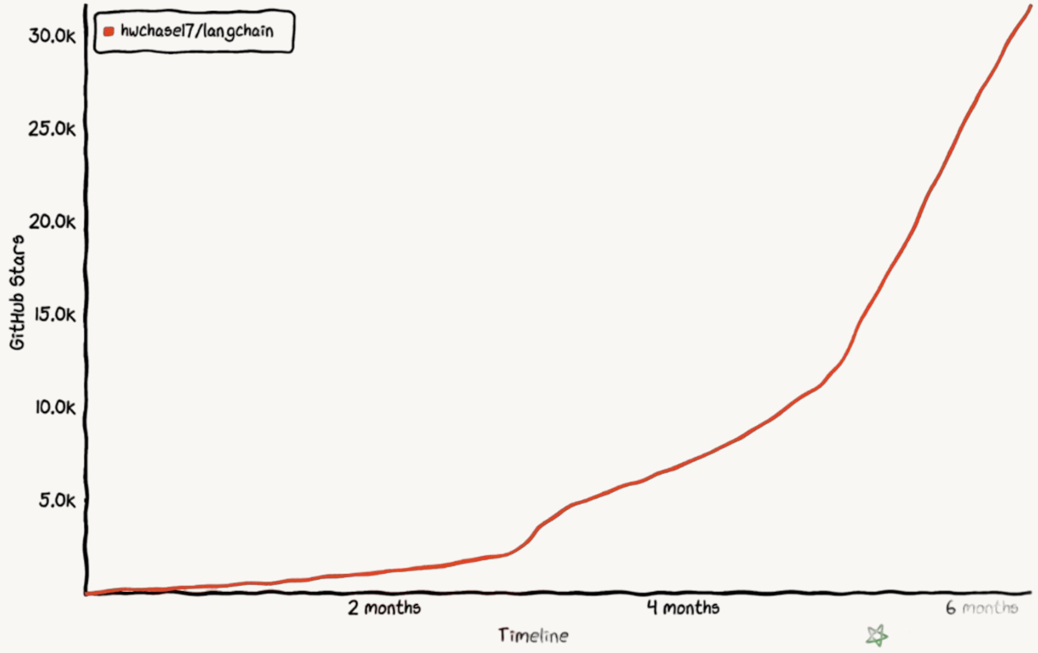


Image source: star-history.com

Multi-stage LLM Chains

Build a sequential flow: article summary output feeds into a sentiment LLM

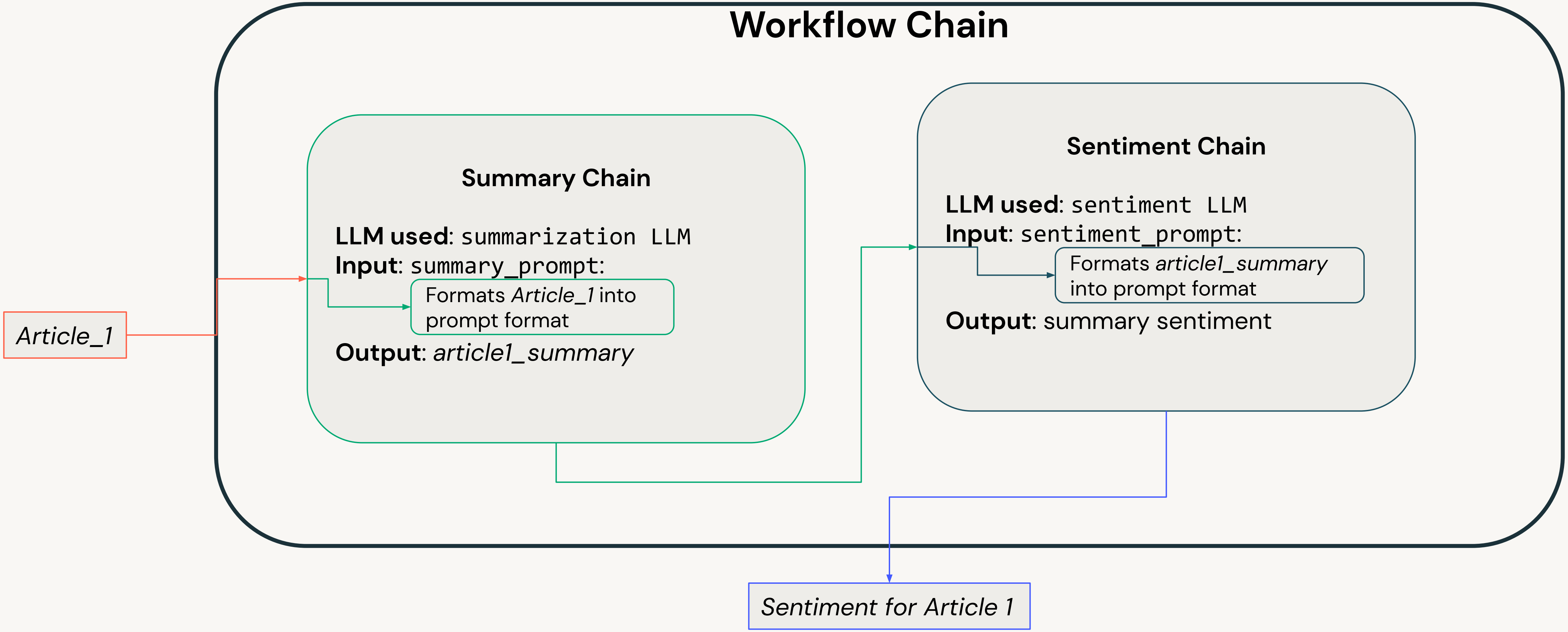
```
# Firstly let’s create our two llms
summary_llm = summarize()
sentiment_llm = sentiment()

# We will also need another prompt template like before, a new sentiment prompt
sentiment_prompt_template = """
Evaluate the sentiment of the following summary: {summary}
Sentiment: """

# As before we create our prompt using this template
sentiment_prompt = PromptTemplate(template=sentiment_prompt_template, input_variable=["summary"])
```

Multi-stage LLM Chains

Let's look at the logic flow of this LLM Chain



Chains with non-LLM tools?

Example: LLMMath in LangChain

Q: How to make an LLMChain that evaluates mathematical questions?

1. The LLM needs to take in the question and return executable code
2. Need to add an evaluation tool for correctness
3. The results need to be passed back

```
class LLMMathChain(Chain):
    """Chain that interprets a prompt and executes python code
to do math."""

    def _evaluate_expression(expression)
        output = str( numexpr.evaluate(expression))

    def process_llm_result(llm_output):
        text_match = re.search(r"^```text(.*?)```", llm_output,
re.DOTALL)
        if text_match:
            output = self._evaluate_expression(text_match)

    def _call(input,llm):
        llm_executor = LLMChain(prompt=input, llm=llm)
        llm_output = llm(input)
        return process_llm_result(llm_output)
```

2 Python library \`numexpr\` used to evaluate the numerical expression

1 LLM response is checked for code snippets that typically have a ``` **code** ``` format in most training datasets

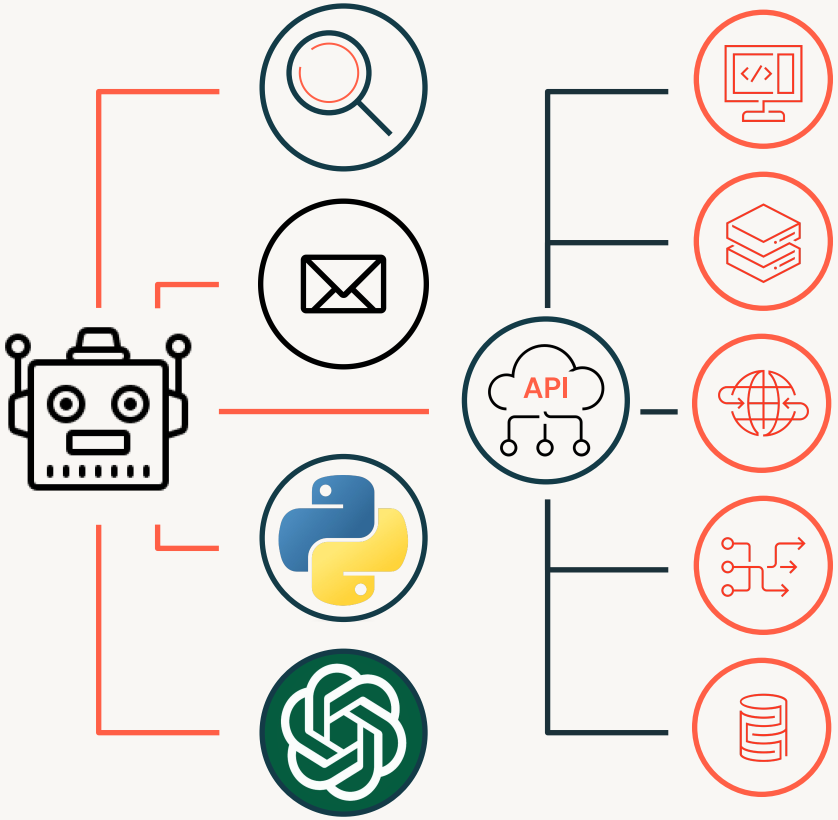
3 "**_call()**" function controls the logic of this custom LLMChain

Source: python.langchain.com

Going ever further

What if we want to use our LLM results to do more?

- Search the web
- Interact with an API
- Run more complex python code
- Send emails
- Even make more versions of itself!
- ……



For this, we will look at toolkits and agents!

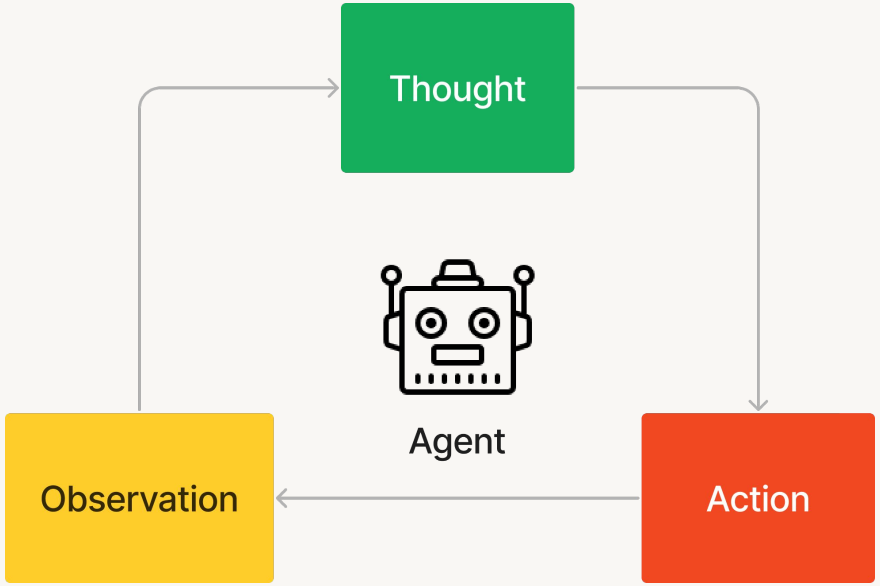
Agents:

Giving LLMs the ability to delegate tasks to specified tools.

LLM Agents

Building reasoning loops

Agents are LLM-based systems that execute the **Reason**/**Action** loop.



Simplified code from the LangChain Agent Source

```
def plan():
"""Given input, decided what to do.
intermediate_steps: Steps the LLM has taken to date, along with observations
"""
  output = self.llm_chain.run(intermediate_steps=intermediate_steps)
  return self.output_parser.parse(output)
def take_next_step() : """Take a single step in the thought-action-observation loop."""
 # Call the LLM to see what to do.
 output = self.agent.plan(intermediate_steps, **inputs)
 # If the tool chosen is the finishing tool, then we end and return.
 for agent_action in actions:
     self.callback_manager.on_agent_action(agent_action)
     # Otherwise we lookup the tool. Call the tool input to get an observation
     observation = tool.run(agent_action.tool_input)
def call(): """Run text through and get agent response."""
 iterations = 0
 # We now enter the agent loop (until it returns something).
 while self._should_continue():
    next_step_output = take_next_step(name_to_tool_map, .., inputs, intermediate_steps)
    iterations += 1
    output = self.agent.return_stopped_response(intermediate_steps, **inputs)
    return self._return(output, intermediate_steps)
```

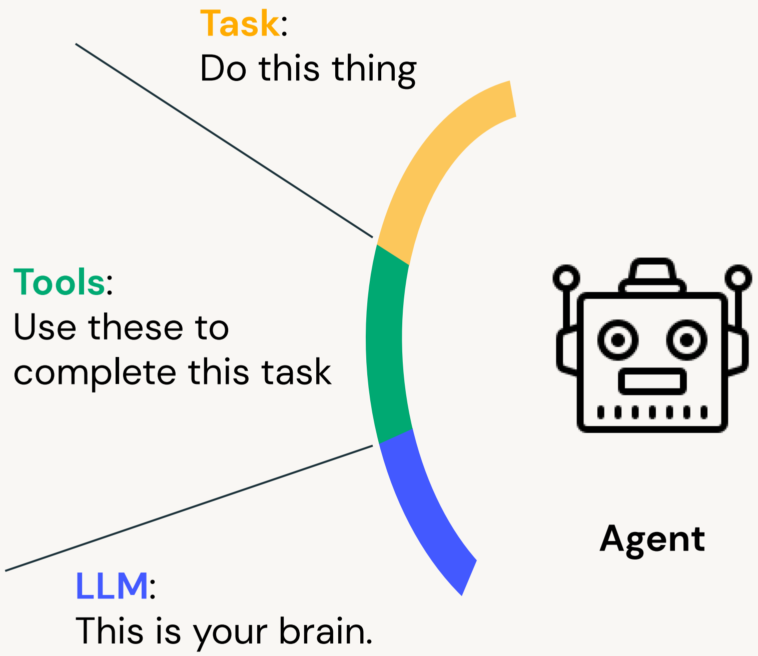
LLM Agents

Building reasoning loops with LLMs

To solve the task assigned, agents make use of two key components:

An LLM as the reasoning/decision making entity.

A set of tools that the LLM will select and execute to perform steps to achieve the task.



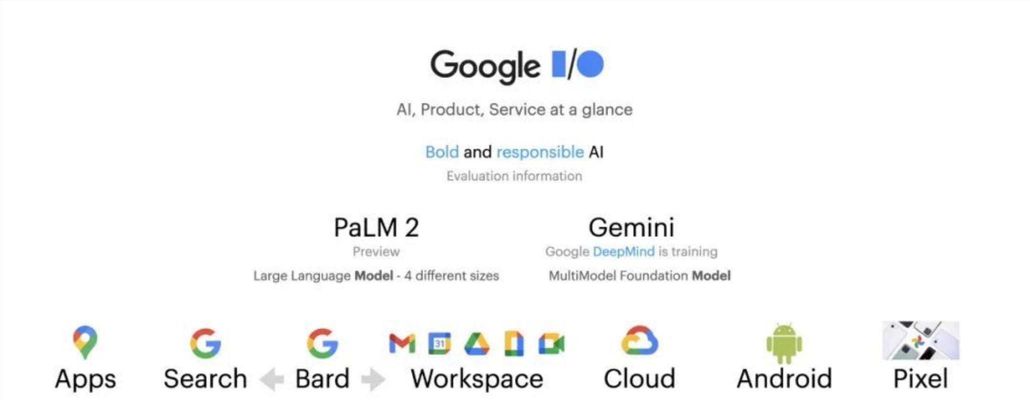

```
tools = load_tools([Google Search,Python Interpreter])
agent = initialize_agent(tools, llm)
agent.run("In what year was Isaac Newton born? What is
that year raised to the power of 0.3141?"))
```

[Simplified code from the LangChain Agent]

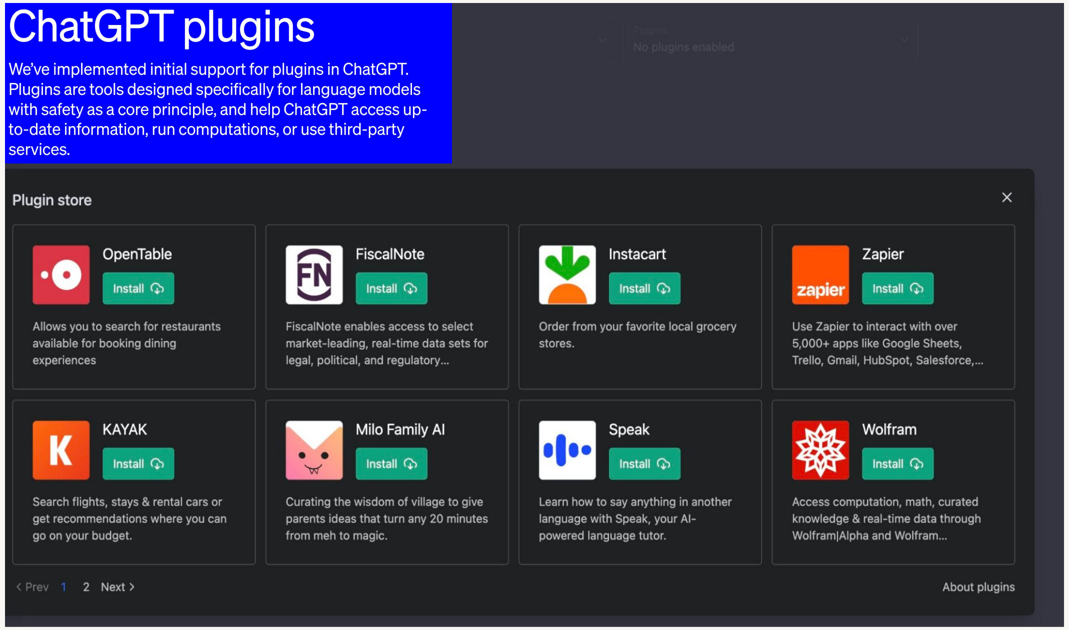
LLM Plugins are coming

LangChain was first to show LLMs+tools. But companies are catching up!

Source: Twitter.com



Source: csdn.net



Source: arstechnica.com

OpenAI and ChatGPT Plugins

OpenAI acknowledged the open-sourced community moving in similar directions

March 23, 2023

Authors
OpenAI ↓

Announcements, Product

LangChain

In line with our iterative deployment philosophy, we are gradually rolling out plugins in ChatGPT so we can study their real-world use, impact, and safety and alignment challenges—all of which we'll have to get right in order to achieve our mission.

Users have been asking for plugins since we launched ChatGPT (and many developers are experimenting with similar ideas) because they unlock a vast range of possible use cases. We're starting with a small set of users and are planning to gradually roll out larger-scale access as we learn more (for plugin developers, ChatGPT users, and after an alpha period, API users who would like to integrate plugins into their products). We're excited to build a community shaping the future of the human–AI interaction paradigm.

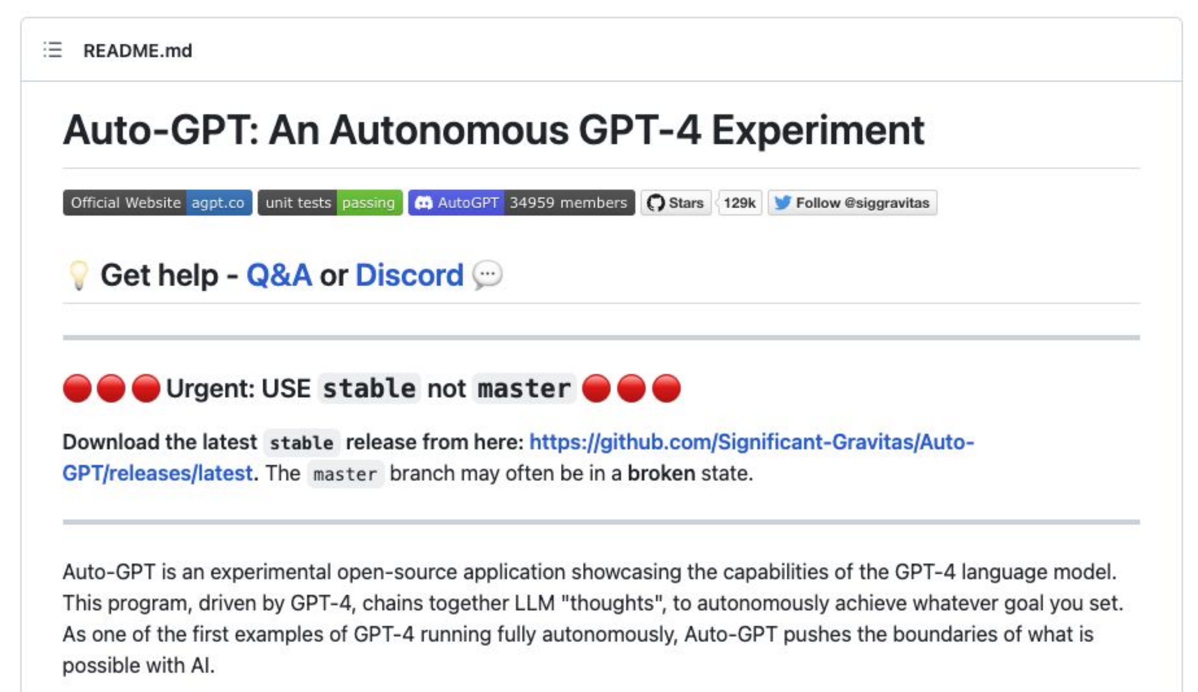
Plugin developers who have been invited off our waitlist can use our documentation to build a plugin for ChatGPT, which then lists the enabled plugins in the prompt shown to the language model as well as documentation to instruct the model how to use each. The first plugins have been created by Expedia, FiscalNote, Instacart, KAYAK, Klarna, Milo, OpenTable, Shopify, Slack, Speak, Wolfram, and Zapier.

Image source: openai.com

Automating plugins: self-directing agents

AutoGPT (early 2023) gains notoriety for using GPT-4 to create copies of itself

- Used self-directed format
- Created copies to perform any tasks needed to respond to prompts


README.md

Auto-GPT: An Autonomous GPT-4 Experiment

Official Website agpt.co | unit tests passing | AutoGPT 34959 members | Stars 129k | Follow @siggravitas

Get help - Q&A or Discord

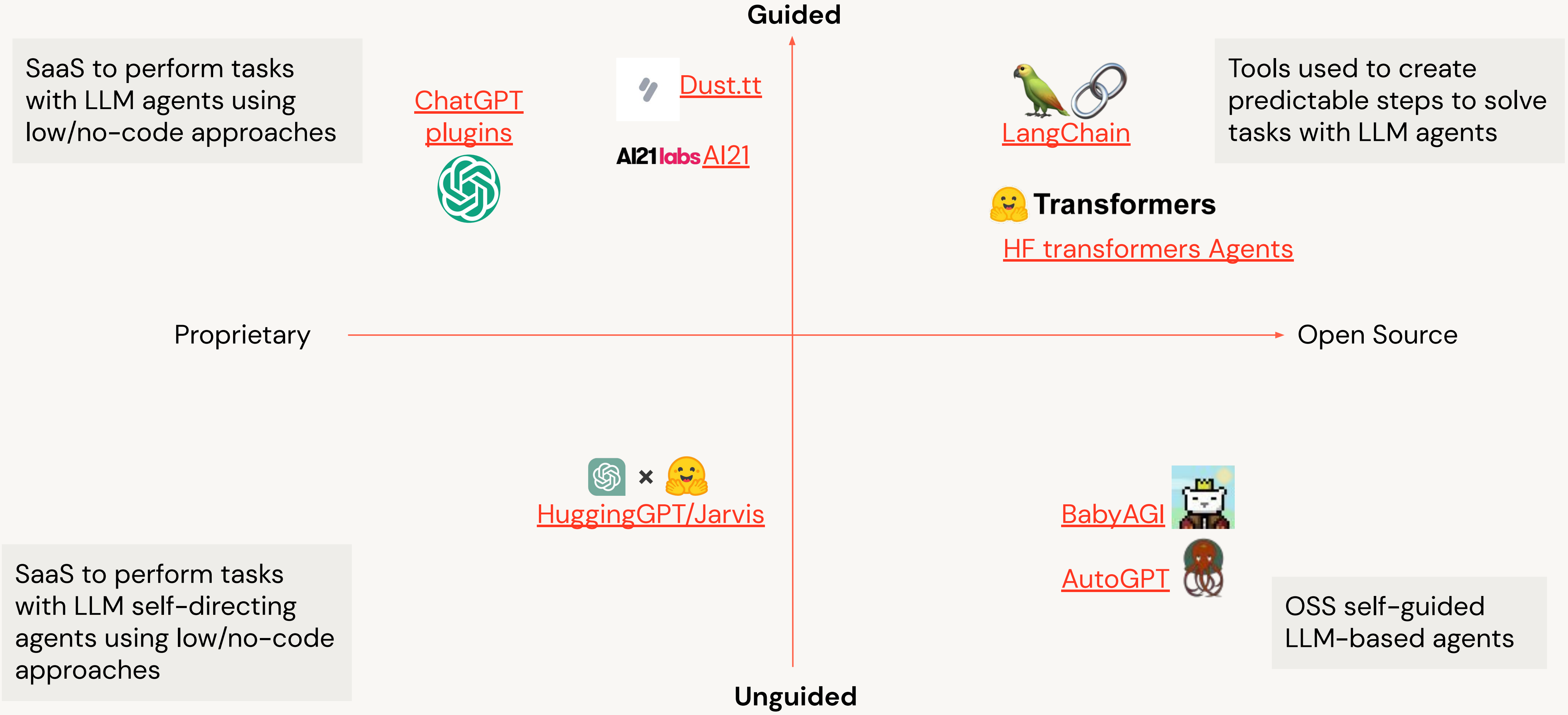
Urgent: USE `stable` not `master`

Download the latest `stable` release from here: https://github.com/Significant-Gravitas/Auto-GPT/releases/latest. The `master` branch may often be in a **broken** state.

Auto-GPT is an experimental open-source application showcasing the capabilities of the GPT-4 language model. This program, driven by GPT-4, chains together LLM "thoughts", to autonomously achieve whatever goal you set. As one of the first examples of GPT-4 running fully autonomously, Auto-GPT pushes the boundaries of what is possible with AI.

Image source: GitHub

Multi-stage Reasoning Landscape



Module Summary

Multi-stage Reasoning - What have we learned?

- LLM Chains help incorporate LLMs into larger workflows, by connecting prompts, LLMs, and other components.
- LangChain provides a wrapper to connect LLMs and add tools from different providers.
- LLM agents help solve problems by using models to plan and execute tasks.
- Agents can help LLMs communicate and delegate tasks.

Time for some code!