Langchain, Agents, Vectors and RAG

Contenido

Introduction

Retrieval Augmented Generation (RAG) is a technique that combines large-scale pre-trained language models (LMs) with external knowledge sources, such as databases or documents, to generate natural language responses. RAG allows the LMs to access relevant information from the knowledge sources during the generation process, which can improve the quality, diversity, and factual consistency of the outputs.

Langchain

LLMChain is a framework that enables the development of LMs that can leverage multiple knowledge sources in a modular and scalable way. LLMChain consists of three main components: Agents, Tools, and Vector databases. Agents are the LMs that perform the generation task, such as question answering, summarization, or dialogue. Tools are the modules that provide the interface between the Agents and the knowledge sources, such as retrieval, ranking, or filtering. Vector databases are the storage and indexing systems that store the knowledge sources and allow efficient and flexible retrieval of relevant information.

Prompt

Notebook 1.1

A prompt for a language model is a set of instructions or input provided by a user to guide the model's response, helping it understand the context and generate relevant and coherent language-based output, such as answering questions, completing sentences, or engaging in a conversation.

There are different types of prompts, we will see three of them.

We will start by importing the class *PromptTemplate* from langchain

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

You have an already existing text information, that is a little text from Wikipedia about Elon Musk.

```
information = """
Elon Reeve Musk (/ˈiːlɒn/ EE-lon; born June 28, 1971) is a businessman and investor. He is the founder, chairman, CEO, and CTO of SpaceX; angel investor
A member of the wealthy South African Musk family, Musk was born in Pretoria and briefly attended the University of Pretoria before immigrating to Canad
In 2004, Musk became an early investor in electric vehicle manufacturer Tesla Motors, Inc. (later Tesla, Inc.). He became the company's chairman and pro
Musk has expressed views that have made him a polarizing figure.[5] He has been criticized for making unscientific and misleading statements, including
"""
```

Then you have this:

```
template = """{information}"""
prompt = PromptTemplate(input_variables=["information"], template=template)
print(prompt.format(information=information))
```

The variable *template* is the one that will contain the prompt text that you want to use. It can include variables, like the information one that is already written.

Then you instantiate the *PrompTemplate*, and as parameters you provide the *input_variables* (list of variables that you're using inside the template) and the *template*.

Finally, if you call format providing variables by name, it will generate the formatted text.

Exercise: modify the template to be a valid LLM prompt that provides the information and ask the LLM to execute 2 tasks: a summarization and provide 2 interesing facts.

```
template = """
given the information {information} I want you to create:
1. A short summary
2. two interesting facts
"""
prompt = PromptTemplate(input_variables=["information"], template=template)
print(prompt.format(information=information))
```

```
given the information
Elon Reeve Musk (/ˈiːlɒn/ EE-lon; born June 28, 1971) is a businessman and inv
CEO, product architect, and former chairman of Tesla, Inc.; owner, executive c
der of Neuralink and OpenAI; and president of the Musk Foundation. He is one c
s net worth to be $178 billion.[4]
A member of the wealthy South African Musk family, Musk was born in Pretoria a
at age 18, acquiring citizenship through his Canadian-born mother. Two years l
er transferred to the University of Pennsylvania and received bachelor's degre
ord University, but dropped out after two days and, with his brother Kimbal, c
by Compaq for $307 million in 1999. That same year, Musk co-founded X.com, a c
2002, eBay acquired PayPal for $1.5 billion. Using $100 million of the money h
company, in 2002.
In 2004, Musk became an early investor in electric vehicle manufacturer Tesla
uct architect, assuming the position of CEO in 2008. In 2006, Musk helped crea
became Tesla Energy. In 2013, he proposed a hyperloop high-speed vactrain tran
telligence research company. The following year, Musk co-founded Neuralink—a r
Company, a tunnel construction company. In 2018, the U.S. Securities and Excha
he had secured funding for a private takeover of Tesla. To settle the case, Mu
22, he acquired Twitter for $44 billion. He subsequently merged the company in
In March 2023, Musk founded xAI, an artificial intelligence company.
Musk has expressed views that have made him a polarizing figure.[5] He has bee
OVID-19 misinformation and antisemitic conspiracy theories.[5][6][7][8] His ow
s of large numbers of employees, an increase in hate speech and misinformation.
 I want you to create:
1. A short summary
2. two interesting facts
```

Few Shot Prompt

Notebook 1.2

Sometimes you want to provide several examples to the LLM, and then provide a partial example so the LLM completes it. Like:

Question: Who wrote the novel '1984'?
Answer: George Orwell

Question: What is the capital city of Australia?
Answer: Canberra

Question: What is the chemical symbol for Gold?
Answer: Au

Question: In what year was the first iPhone released?
Answer: 2007

Question: What are wrinkles?

To achieve that, we will use two classes: the *PromptTemplate* and the *FewShotPromptTemplate*

```
from langchain.prompts import PromptTemplate, FewShotPromptTemplate
```

We already have a *questions* variable with some examples:

```
questions = [
    {
        "question": "Who wrote the novel '1984'?",
        "answer": "George Orwell"
    },
    {
        "question": "What is the capital city of Australia?",
        "answer": "Canberra"
    },
    {
        "question": "What is the chemical symbol for Gold?",
        "answer": "Au"
    },
    {
        "question": "In what year was the first iPhone released?",
        "answer": "2007"
    }
]
```

The first step is to build a *PromptTemplate* able to build the text of an example given the question and answer

```
question_prompt = PromptTemplate(input_variables=["question", "answer"], template="Question: {question}\nAnswer: {answer}")
```

Then we will instantiate *FewShotPromptTemplate*

```
prompt = FewShotPromptTemplate(
  examples = ???,
  example_prompt=???,
  suffix=???,
  input_variables=???
)
print(prompt.format(input="What are wrinkles?"))
```

What means each parameter?

The parameter *examples* will receive the array of examples.

The parameter *example_prompt* will receive a *PromptTemplate* instance that, given an item from the examples, is able to build a prompt for this example.

The parameter *suffix* receives the template text that will generate the text after the examples.

The parameter *input_variables* is the array of variable names, in this case, for the suffix.

Our code should look like this:

```
prompt = FewShotPromptTemplate(
  examples = questions,
  example_prompt=question_prompt,
  suffix="Question: {input}",
  input_variables=["input"]
)
print(prompt.format(input="What are wrinkles?"))
```

```
Question: Who wrote the novel '1984'?
Answer: George Orwell

Question: What is the capital city of Australia?
Answer: Canberra

Question: What is the chemical symbol for Gold?
Answer: Au

Question: In what year was the first iPhone released?
Answer: 2007

Question: What are wrinkles?
```

Chat Prompt

Notebook 1.3

Sometimes the prompt that we are looking for represents a conversation between the user and the LLM.

First at all, we import the class *ChatPromptTemplate* from langchain.

```
from langchain.prompts import ChatPromptTemplate
```

We will need a list of tuples of the conversation

```
messages = [
    ("system", "You are a helpful AI bot. Your name is {name}."),
    ("human", "Hello, how are you doing?"),
    ("ai", "I'm doing well, thanks!"),
    ("human", "{user_input}"),
]
```

As you can see, we have three roles:

- Human: a message sent from the perspective of the human
- AI: a message sent from the perspective of the AI
- System: a message setting the objectives the AI should follow

We create the *ChatPromptTemplate* instance.

```
prompt = ChatPromptTemplate.from_messages(messages)
```

Finally, we use it in a similar way to *PrompTemplate*, but we can skip the *input_variables* array, and provide the variables directly.

```
messages = prompt.format_messages(name=???, user_input=???)
print(messages)
```

```
messages = prompt.format_messages(name="Bob", user_input="What is your name?")
print(messages)
```

```
[SystemMessage(content='You are a helpful AI bot. Your name is Bob.'), HumanMessage(content='Hello, how are you doing?'), AIMessage(content="I'm doing well, thanks!"), HumanMessage(content='What is your name?')]
```

Wikipedia

Notebook 1.4

We will use the PromptTemplate combined with the result from Wikipedia to build a prompt.

```
from langchain.prompts import PromptTemplate
from langchain_community.tools import WikipediaQueryRun
from langchain_community.utilities import WikipediaAPIWrapper
```

We create our PromptTemplate

```
template = """
given the information {information} I want you to create:
1. A short summary
2. two interesting facts
"""
prompt = PromptTemplate(input_variables=["information"], template=template)
```

We call Wikipedia with a topic that we choose.

```
topic = ???
wikipedia = WikipediaQueryRun(api_wrapper=WikipediaAPIWrapper())
information = wikipedia.run(topic)
```

Example: if we run that with Elon Musk as topic, we will obtain something similar to what we got at notebook 1.1. but without hardcoding the information

```
topic = "Elon Musk"
wikipedia = WikipediaQueryRun(api_wrapper=WikipediaAPIWrapper())
information = wikipedia.run(topic)
```

```
print(prompt.format(information=information))
```

```
given the information Page: Elon Musk
Summary: Elon Reeve Musk ( EE-lon; born June 28, 1971) is a businessma
r, CEO, product architect, and former chairman of Tesla, Inc.; owner,
ounder of Neuralink and OpenAI; and president of the Musk Foundation.
his net worth to be $178 billion.
A member of the wealthy South African Musk family, Musk was born in Pr
at age 18, acquiring citizenship through his Canadian-born mother. Two
er transferred to the University of Pennsylvania and received bachelor
ord University, but dropped out after two days and, with his brother K
by Compaq for $307 million in 1999. That same year, Musk co-founded X.
2002, eBay acquired PayPal for $1.5 billion. Using $100 million of the
company, in 2002.
In 2004, Musk became an early investor in electric vehicle manufacture
uct architect, assuming the position of CEO in 2008. In 2006, Musk hel
became Tesla Energy. In 2013, he proposed a hyperloop high-speed vactr
telligence research company. The following year, Musk co-founded Neura
```

Chat OpenAI

Notebook 1.5

We start by importing the class *ChatOpenAI* from langchain.

We also import *load_dotenv* and *os* as we want to have the API Keys from the environment or in a .env file.

```
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv
import os
```

We load the .env file and get the OPENAI_API_KEY from the environment.

```
load_dotenv(override=True)
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
```

We create our LLM connection to OpenAI

```
llm = ChatOpenAI(
  model="gpt-3.5-turbo",
  temperature=0,
  api_key=OPENAI_API_KEY
)
```

We are using GPT-3.5-turbo.

Are you able to build a translator from English to Japanese? Use the structure of code below:

```
messages = [
    ("system", ???),
    ("human", ???),
]
res = llm.invoke(messages)
print(res)
```

```python
messages = [
    ("system", "You are a helpful assistant that translates English to Japanese."),
    ("human", "Translate this sentence from English to Japanese. I love programming."),
]
res = llm.invoke(messages)
print(res)
```

```
content='私はプログラミングが大好きです。' response_metadata={'token_usage': {'completion_tokens'
e': 'gpt-3.5-turbo', 'system_fingerprint': 'fp_3b956da36b', 'finish_reason': 'stop', 'logprobs
```

Runnables

Notebook 1.6

We will put everything together using the topics of the previous lessons.

We will have a prompt template that gets information from the Wikipedia and an LLM.

```
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv
import os
from langchain.prompts import PromptTemplate
from langchain_community.tools import WikipediaQueryRun
from langchain_community.utilities import WikipediaAPIWrapper
```

```
load_dotenv(override=True)
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
```

```
template = """
given the information {information} I want you to create:
1. A short summary
2. two interesting facts
"""
prompt = PromptTemplate(input_variables=["information"], template=template)
```

```
topic = "Elon Musk"
wikipedia = WikipediaQueryRun(api_wrapper=WikipediaAPIWrapper())
information = wikipedia.run(topic)
```

```
llm = ChatOpenAI(
  model="gpt-3.5-turbo",
  temperature=0,
  api_key=OPENAI_API_KEY
)
```

Finally we will create a runnable: a pipeline of tasks to be executed one after the other.

```
chain = prompt | llm
res = chain.invoke(input = { "information": information })
```

```
print(res.content)
```

```
Summary: Elon Musk acquired Twitter in October 2022 and served
Twitter underwent significant changes, including rebranding to
act-checking system.

Two interesting facts:
1. Musk's acquisition of Twitter for $44 billion in 2022 marked
al media industries.
2. The rebranding of Twitter to X in July 2023 signaled a new d
ration, and verification processes.
```

Formatting output

Notebook 1.7

Imagine that we want to generate the output in a given format, and we want the LLM to format it given a model for the output that we want. To achieve that, we need to somehow get the formatting instructions for the LLM from the model.

Let's use Pydantic for that.

First we import the libraries, the OPENAI_API_KEY and we create the LLM.

```
from typing import List
from dotenv import load_dotenv
import os
from langchain.prompts import PromptTemplate
from langchain_openai import ChatOpenAI
from langchain.pydantic_v1 import BaseModel, Field, validator
from langchain.output_parsers import PydanticOutputParser
```

```
load_dotenv(override=True)
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
```

```
llm = ChatOpenAI(
  model="gpt-3.5-turbo",
  temperature=0,
  api_key=OPENAI_API_KEY
)
```

Imagine that what we want to achieve is get a random Actor and some film examples, and store it in an instance of the model. We can create the model like:

```
class ActorModel(BaseModel):
    name: str = Field(description="name of an actor")
    film_names: List[str] = Field(description="list of names of films they starred in")
```

Then we can automatically get the format instructions for the LLM:

```
parser = PydanticOutputParser(pydantic_object=ActorModel)
format_instructions = parser.get_format_instructions()
print(format_instructions)
```

```
The output should be formatted as a JSON instance that conforms to the JSON schema below.

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a list of strings", "type": "array", "items": {"type": "string"}}},
"required": ["foo"]}
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The object {"properties": {"foo": ["bar", "baz"]}} is not well-formatted.

Here is the output schema:
```
{"properties": {"name": {"title": "Name", "description": "name of an actor", "type": "string"}, "film_names": {"title": "Film Names", "description": "lis
t of names of films they starred in", "type": "array", "items": {"type": "string"}}}, "required": ["name", "film_names"]}
```
```

Then we create the Runnable chaining the prompt, the LLM and the parser to generate the output:

```
prompt = PromptTemplate(
    template = "Answer the user query.\n {format_instructions}\n {query}\n",
    input_variables=["query", "format_instructions"]
)
```

```
chain = prompt | llm | parser
res = chain.invoke({ "query": "Generate the filmography for a random actor", "format_instructions": format_instructions })
print(res)
```

```
name='Tom Hanks' film_names=['Forrest Gump', 'Cast Away', 'Saving Private Ryan', 'Toy Story', 'The Green Mile']
```

The output is an instance of the ActorModel

Agents

In the previous block we learned about Runnables, basically chains of methods, usually a prompt, a LLM and an output parser.

But what happens when we want to have more complex tasks for our LLM, or to expand it with new skills?

This is the objective of the Agents: to provide Tools that will complement the LLM, and the Agent will use the LLM and Tools to build the proper answer, even if it has to decide between which tool to use or even to create chains of tools to generate the proper answer.

But first, lest talk about Tavily.

Tavily

Notebook 2.1

When we want to search some information in internet, and we want to get it formatted, Tavily is a very good option, compatible with LLMs. This will be an example of usage.

We import TavilySearchAPIRetriever, and the TAVILY_API_KEY

```
from dotenv import load_dotenv
import os
from langchain_community.retrievers import TavilySearchAPIRetriever
```

```
load_dotenv(override=True)
TAVILY_API_KEY = os.getenv("TAVILY_API_KEY")
```

Then we invoke the search of one topic:

```
retriever = TavilySearchAPIRetriever(api_key=TAVILY_API_KEY, k=3)

retriever.invoke("what year was breath of the wild released?")
```

```
[Document(page_content='[]\nReferences\nThe Legend of Zelda \xa0·\nThe Adventure of Link \xa0
ng (DX; Nintendo Switch) \xa0·\nOcarina of Time (Master Quest; 3D) \xa0·\nMajora\'s Mask (3D)
Swords (Anniversary Edition) \xa0·\nThe Wind Waker (HD) \xa0·\nFour Swords Adventures \xa0·\n
Hourglass \xa0·\nSpirit Tracks \xa0·\nSkyward Sword (HD) \xa0·\nA Link Between Worlds \xa0·\n
he Kingdom\nZelda (Game & Watch) \xa0·\nThe Legend of Zelda Game Watch \xa0·\nLink\'s Crossbo
0·\nCadence of Hyrule \xa0·\nGame & Watch: The Legend of Zelda\nCD-i Games\n Listings[]\nChar
s[]\nTranslations[]\nCredits[]\nReception[]\nSales[]\nEiji Aonuma and Hidemaro Fujibayashi ac
at The Game Awards 2017\nBreath of the Wild was estimated to have sold approximately 1.3 mill
owners were estimated to have also purchased the game.[52] Sales of the game have remained st
7.14 million copies worldwide while the Wii U version has sold 1.69 million copies worldwide
cumulative total of 28.83 million copies sold.\n It also earned a Metacritic score of 97 from
ames of all time.[59][60] Notably, the game received the most perfect review scores for any g
ath of the Wild was chosen as the best Legend of Zelda game of all time in their "Top 10 Best
cond" best Zelda game in their new revamped version of their "Top 10 Best Zelda Games" list i
on ranks Breath of the Wild as one of the best video games of all time.[74] Metacritic ranked
```

If will give us also some metadata

```
nt relics and the need of the ancient technology, so the Hylians discovered the ancient relics that the Sheikah Tribe had constructed ten thousand years
prior: the Guardians and the four Divine Beasts, which were constructed to protect Hyrule. Link partners with Yunobo by sneaking their way up to avoid th
e spotlights from the Guardian Sentries and fires the young Goron at Vah Rudania until it's forced into the crator where Link hops on and activates five
terminals and the Main Control Unit, and manages to kill Fireblight Ganon, which Daruk's spirit takes it to the top of the mountain to take aim at the ca
stle.", metadata={'title': 'The Legend of Zelda: Breath of the Wild | Nintendo | Fandom', 'source': 'https://nintendo.fandom.com/wiki/The_Legend_of_Zeld
a:_Breath_of_the_Wild', 'score': 0.94834, 'images': None})]
```

Maths with LLMMathChain

Notebook 2.2

If we want to perform mathematical operations with LLMs is not so simple, but we have the class LLMMathChain to help us.

We will create our LLM instance as usual:

```
from dotenv import load_dotenv
import os
from langchain import LLMMathChain
from langchain_openai import ChatOpenAI
```

```
load_dotenv(override=True)
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
```

```
llm = ChatOpenAI(
  model="gpt-3.5-turbo",
  temperature=0,
  api_key=OPENAI_API_KEY
)
```

Now we can do some maths using natural language

```
llm_math = LLMMathChain.from_llm(llm=llm, verbose=True)
```

```
llm_math.invoke("What is 13 raised to the .3432 power?")
```

```
> Entering new LLMMathChain chain...
What is 13 raised to the .3432 power?```text
13**0.3432
```
...numexpr.evaluate("13**0.3432")...

Answer: 2.4116004626599237
> Finished chain.
{'question': 'What is 13 raised to the .3432 power?',
 'answer': 'Answer: 2.4116004626599237'}
```
```

Tools

Notebook 2.3

Tools are the skills that we provide to our agents, with some instructions about how to use them.

First we will create our llm, our tavily retriever and our llm_math of previous lessons

```
from dotenv import load_dotenv
import os
from langchain import LLMMathChain
from langchain_openai import ChatOpenAI
from langchain_community.retrievers import TavilySearchAPIRetriever
from langchain.agents import initialize_agent, Tool, AgentType
```

```
load_dotenv(override=True)
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
TAVILY_API_KEY = os.getenv("TAVILY_API_KEY")
```

```
llm = ChatOpenAI(
  model="gpt-3.5-turbo",
  temperature=0,
  api_key=OPENAI_API_KEY
)
```

```
retriever = TavilySearchAPIRetriever(api_key=TAVILY_API_KEY, k=3)
```

```
llm_math = LLMMathChain.from_llm(llm=llm,verbose=True)
```

Now we can define an array of tools.

```
tools = [
    Tool(
        name = "Search",
        func=retriever.invoke,
        description="useful for when you need to answer questions about current events. You should ask targeted questions"
    ),
    Tool(
        name="Calculator",
        func=llm_math.invoke,
        description="useful for when you need to answer questions about math"
    )
]
```

For each tool we need the name of the tool, the function to be invoked and a description of the tool. This description will be used by the LLM to take decisions and create chains in real time.

We will do that using initialize_agent. This function is deprecated, but we will see the alternatives later as we will need more information.

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

```
C:\Users\jseij\AppData\Local\Programs\Python\Python312\Lib\site-packages\langchain_core
n `initialize_agent` was deprecated in LangChain 0.1.0 and will be removed in 0.2.0. Us
e_json_agent, create_structured_chat_agent, etc. instead.
  warn_deprecated(
```

```
query = "multiply the current temperature in Tokyo by two and raise to the cube"
agent.invoke(query)
```

```
> Entering new AgentExecutor chain...

Invoking: `Search` with `current temperature in Tokyo`


[Document(page_content='Tokyo Weather Forecasts. Weather Underground provides local & long-range weather forecasts, weatherreports, maps & tropical weath
er conditions for the Tokyo area.', metadata={'title': 'Tokyo, Japan Weather Conditions | Weather Underground', 'source': 'https://www.wunderground.com/w
eather/jp/tokyo', 'score': 0.96276, 'images': None}), Document(page_content='Current weather in Tokyo, Tokyo, Japan. Check current conditions in Tokyo, T
okyo, Japan with radar, hourly, and more.', metadata={'title': 'Tokyo, Tokyo, Japan Current Weather | AccuWeather', 'source': 'https://www.accuweather.co
m/en/jp/tokyo/226396/current-weather/226396', 'score': 0.92737, 'images': None}), Document(page_content='Current weather in Tokyo and forecast for today,
tomorrow, and next 14 days', metadata={'title': 'Weather for Tokyo, Japan - timeanddate.com', 'source': 'https://www.timeanddate.com/weather/japan/toky
o', 'score': 0.90993, 'images': None})]
Invoking: `Search` with `current weather in Tokyo`


[Document(page_content="{'location': {'name': 'Tokyo', 'region': 'Tokyo', 'country': 'Japan', 'lat': 35.69, 'lon': 139.69, 'tz_id': 'Asia/Tokyo', 'localt
ime_epoch': 1715086400, 'localtime': '2024-05-07 21:53'}, 'current': {'last_updated_epoch': 1715085900, 'last_updated': '2024-05-07 21:45', 'temp_c': 19.
2, 'temp_f': 66.6, 'is_day': 0, 'condition': {'text': 'Overcast', 'icon': '//cdn.weatherapi.com/weather/64x64/night/122.png', 'code': 1009}, 'wind_mph':
4.3, 'wind_kph': 6.8, 'wind_degree': 40, 'wind_dir': 'NE', 'pressure_mb': 1006.0, 'pressure_in': 29.72, 'precip_mm': 0.07, 'precip_in': 0.0, 'humidity':
79, 'cloud': 100, 'feelslike_c': 19.2, 'feelslike_f': 66.6, 'vis_km': 14.0, 'vis_miles': 8.0, 'uv': 1.0, 'gust_mph': 11.0, 'gust_kph': 17.7}}", metadata=
{'title': 'Weather in Tokyo', 'source': 'https://www.weatherapi.com/', 'score': 0.98577, 'images': None}), Document(page_content='Current weather in Toky
o, Tokyo, Japan. Check current conditions in Tokyo, Tokyo, Japan with radar, hourly, and more.', metadata={'title': 'Tokyo, Tokyo, Japan Current Weather
| AccuWeather', 'source': 'https://www.accuweather.com/en/jp/tokyo/226396/current-weather/226396', 'score': 0.96515, 'images': None}), Document(page_cont
ent='Get the monthly weather forecast for Tokyo, Tokyo, Japan, including daily high/low, historical averages, to help you plan ahead.', metadata={'titl
e': 'Tokyo, Tokyo, Japan Monthly Weather | AccuWeather', 'source': 'https://www.accuweather.com/en/jp/tokyo/226396/may-weather/226396', 'score': 0.95676,
'images': None})]
Invoking: `Calculator` with `(19.2 * 2) ** 3`
responded: The current temperature in Tokyo is 19.2°C. Let me calculate the result for you.


> Entering new LLMMathChain chain...
(19.2 * 2) ** 3```text
(19.2 * 2) ** 3
```
...numexpr.evaluate("(19.2 * 2) ** 3")...

Answer: 56623.10399999999
> Finished chain.
{'question': '(19.2 * 2) ** 3', 'answer': 'Answer: 56623.10399999999'}The result of multiplying the current temperature in Tokyo by two and raising it to
the cube is approximately 56623.104.

> Finished chain.
```

```
{'input': 'multiply the current temperature in Tokyo by two and raise to the cube',
 'output': 'The result of multiplying the current temperature in Tokyo by two and raising it to the cube is approximately 56623.104.'}
```

It calls Search tool twice, one to get the url of the service for the temperature, and other to really get the temperature from the service.

Later, it calls Calculate tool to perform the maths.
```

ReAct

Notebook 2.4

You have the ReAct paper at the data folder.

Published as a conference paper at ICLR 2023

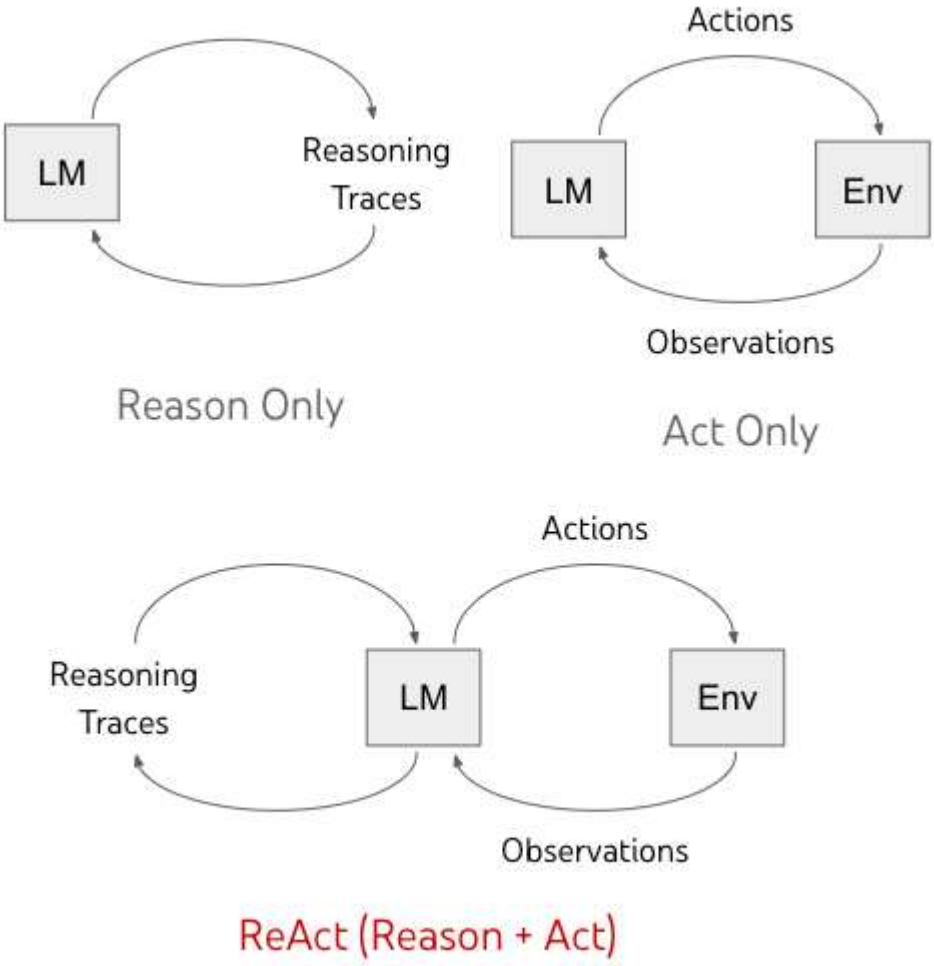
REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao*[,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

ABSTRACT

While large language models (LLMs) have demonstrated impressive performance across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate its effectiveness over state-of-the-art baselines in addition to improved human interpretability and trustworthiness. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generating human-like task-solving trajectories that are more interpretable than baselines without reasoning traces. Furthermore, on two interactive decision making benchmarks (ALFWorld and WebShop), ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively, while being prompted with only one or two in-context examples.

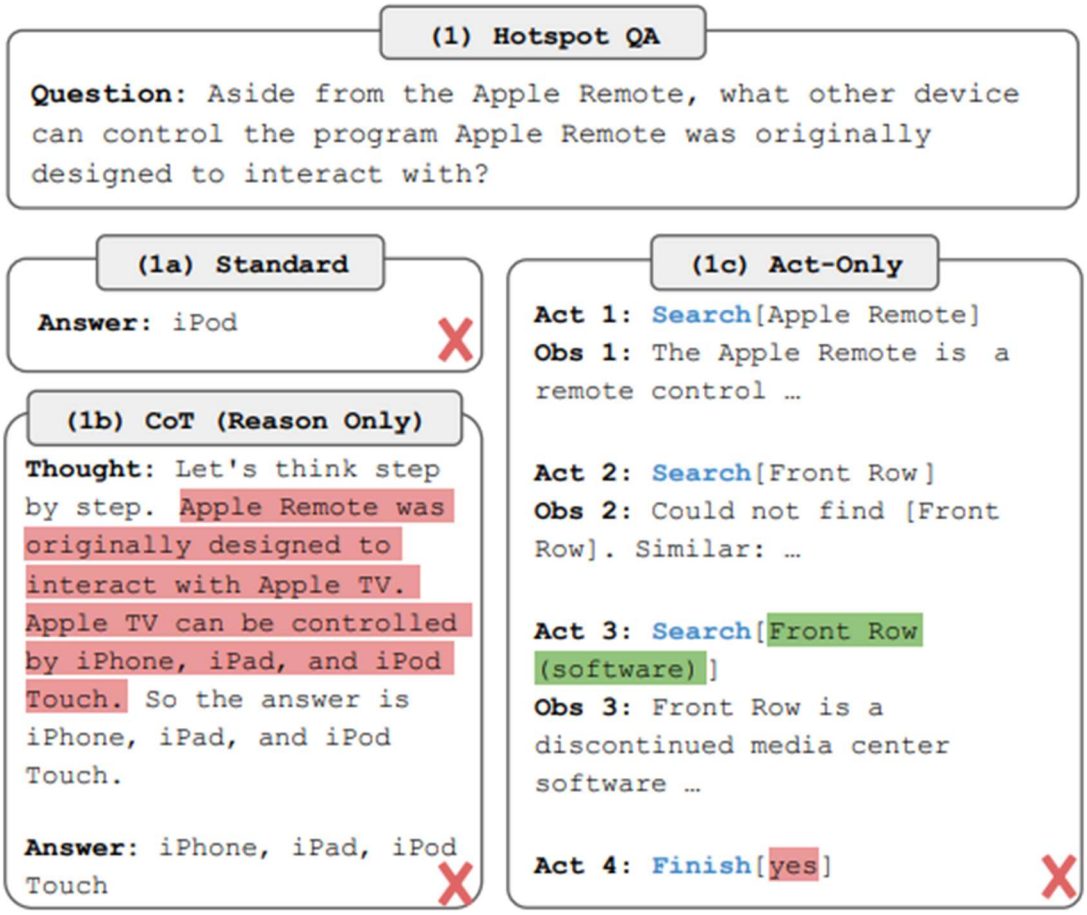


LLMs can be used for reasoning and for acting. ReAct propose that there are synergies between reasoning and acting, and their capabilities can be combined.

A ReAct prompt consists of few-shot task-solving trajectories, with human-written text reasoning traces and actions, as well as environment observations in response to actions.

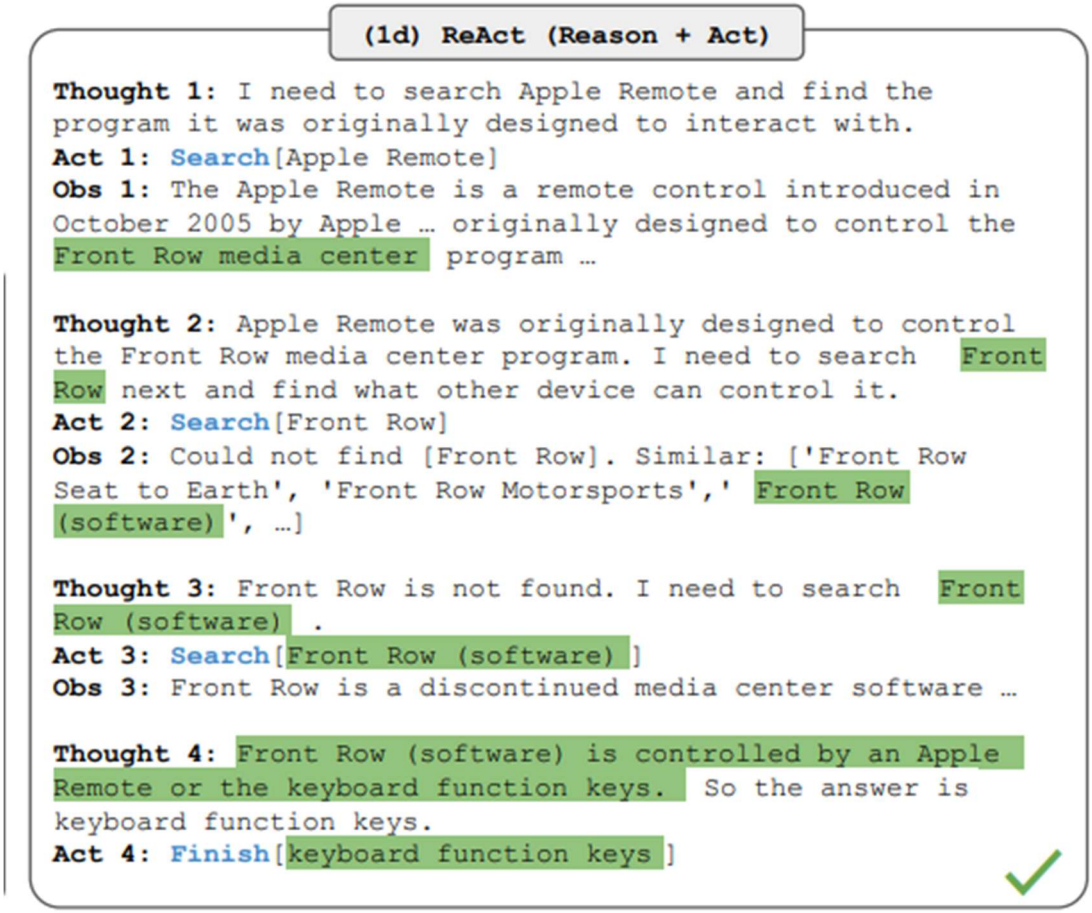
Reasoning only: the LLM uses it's internal representation to generate thoughts, without information from the external world. Models in reasoning only lacks integration with the real world.

Act only: capacity of the system to execute actions that impacts the environment.



Reasoning Only: This approach takes a step-by-step reasoning process, concluding that Apple TV, is controlled by iPhone, iPad, and iPod Touch. The answer includes multiple devices and is more comprehensive but is incorrect.

Act Only: This part details a sequence of actions without explicit reasoning for each step. The actions involve searching for information about the Apple Remote and related software or services (like Front Row), leading to the conclusion that no more steps are required (Finish[yes]), leading to an incorrect answer.



The system starts by engaging in a reasoning process. It understands the task at hand, and makes decisions on how to proceed.

Based on reasoning, the system then takes a specific action. This action could involve using a tool or executing a command that affects the system's environment. For example, querying a database, interacting with a user, or manipulating some digital content.

After taking action, the system gathers feedback from the environment.

The feedback is fed back into the system, where it is used to refine and improve the initial reasoning

So now, we will refactor our last code to use ReAct.

This will be very similar, but changing some of the imports:

```
from dotenv import load_dotenv
import os
from langchain import LLMMathChain
from langchain_openai import ChatOpenAI
from langchain_community.retrievers import TavilySearchAPIRetriever
from langchain.agents import create_react_agent, Tool, AgentExecutor
from langchain.prompts import PromptTemplate
```

```
load_dotenv(override=True)
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
TAVILY_API_KEY = os.getenv("TAVILY_API_KEY")
```

```
llm = ChatOpenAI(
  model="gpt-3.5-turbo",
  temperature=0,
  api_key=OPENAI_API_KEY
)
```

```
retriever = TavilySearchAPIRetriever(api_key=TAVILY_API_KEY, k=3)
```

```
llm_math = LLMMathChain.from_llm(llm=llm,verbose=True)
```

```
tools = [
    Tool(
        name = "Search",
        func=retriever.invoke,
        description="useful for when you need to answer questions about current events. You should ask targeted questions"
    ),
    Tool(
        name="Calculator",
        func=llm_math.invoke,
        description="useful for when you need to answer questions about math"
    )
]
```

We need the ReAct prompt:

```
template="""
Answer the following questions as best you can. You have access to the following tools: {tools}
Use the following format:
Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!
Question: {input}
Thought:{agent_scratchpad}
"""
```

```
prompt = PromptTemplate(input_variables=["tools", "tool_names", "input", "agent_scratchpad"], template=template)
```

Now we will have the agent and one agent executor:

```
agent = create_react_agent(llm=llm, tools=tools, prompt=prompt)
```

```
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
```

```
query = "multiply the current temperature in Tokyo by two and raise to the cube"
agent_executor.invoke({"input": query})
```

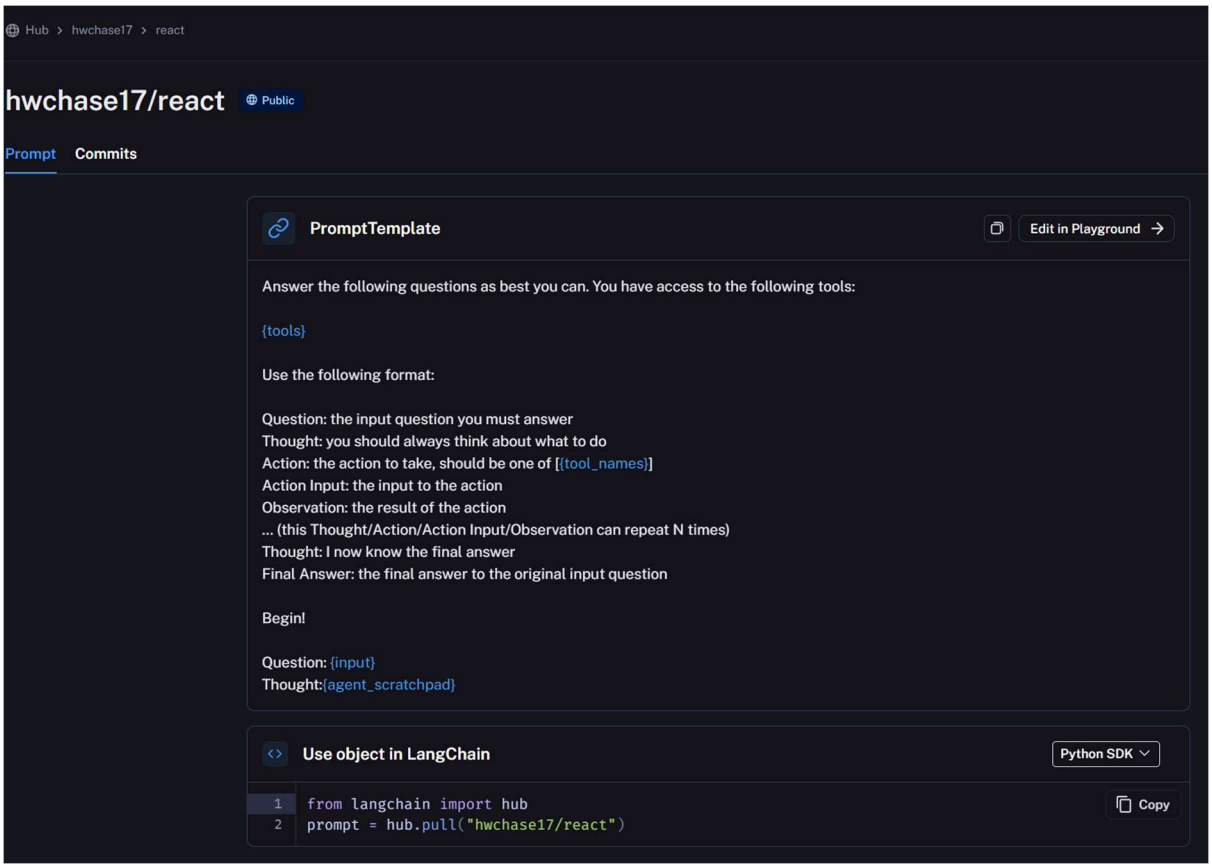
It will take more steps that before, because now is reasoning with smaller steps, representing the reasoning.

Prompt Hub

Notebook 2.5

Langchain has a prompt hub, is like github but for prompts.

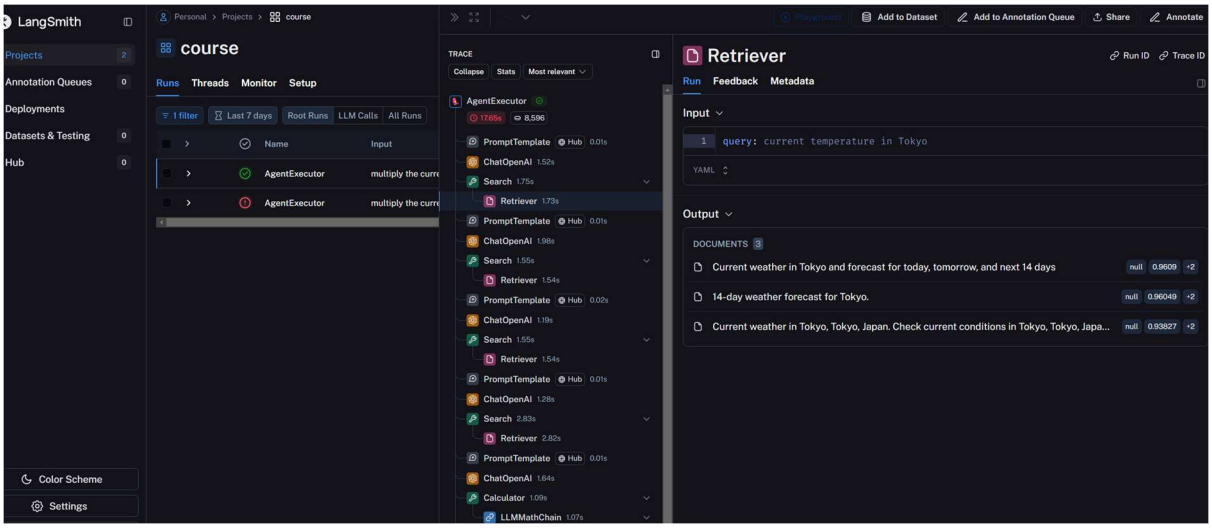
In this prompt hub, the ReAct prompt already exists:



So we can replace the text of our template and the prompt instantiation by this:

```
prompt = hub.pull("hwchase17/react")
```

Langsmith


LangSmith
Projects 2
Annotation Queues 0
Deployments
Datasets & Testing 0
Hub 0
Color Scheme
Settings
Personal › Projects › course
course
Runs Threads Monitor Setup
1 filter
Last 7 days
Root Runs LLM Calls All Runs
Name
Input
AgentExecutor
multiply the curre
AgentExecutor
multiply the curre
Add to Dataset
Add to Annotation Queue
Share
Annotate
TRACE
Collapse
Stats
Most relevant
AgentExecutor
17.65s
8,596
PromptTemplate Hub 0.01s
ChatOpenAI 1.52s
Search 1.75s
Retriever 1.73s
PromptTemplate Hub 0.01s
ChatOpenAI 1.98s
Search 1.55s
Retriever 1.54s
PromptTemplate Hub 0.02s
ChatOpenAI 1.19s
Search 1.55s
Retriever 1.54s
PromptTemplate Hub 0.01s
ChatOpenAI 1.28s
Search 2.83s
Retriever 2.82s
PromptTemplate Hub 0.01s
ChatOpenAI 1.64s
Calculator 1.09s
LLMMathChain 1.07s
Retriever
Run ID
Trace ID
Run
Feedback
Metadata
Input
1 query: current temperature in Tokyo
YAML
Output
DOCUMENTS 3
Current weather in Tokyo and forecast for today, tomorrow, and next 14 days
null 0.9609 +2
14-day weather forecast for Tokyo.
null 0.96049 +2
Current weather in Tokyo, Tokyo, Japan. Check current conditions in Tokyo, Tokyo, Japa...
null 0.93827 +2

Inside ReAct Agent

Notebook 2.6

We can create tools, one of the easiest way is to define a function and use the tool decorator.

```
@tool
def get_text_length(text: str) -> int:
    """return the length of a text"""
    return len(text.strip("\n").strip('"').strip("'"))
```

```
tools = [get_text_length]
```

We create the prompt and provide the tools and tool names.

```
template="""
Answer the following questions as best you can. You have access to the following tools: {tools}
Use the following format:
Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!
Question: {input}
Thought: {agent_scratchpad}
"""
```

```
prompt = PromptTemplate.from_template(template=template).partial(tools=render_text_description(tools), tool_names=", ".join([t.name for t in tools]))
```

Then we will create the agent_scrathpad to store information of each step, and the agent, that will an input an agent_scratchpad, piped with the prompt, the llm and finally the ReAct output parser that will convert the output to objects. We are interested in objects of class AgentAction (agent must execute an action invoking a tool) and AgentFinish.

```
agent_scratchpad = []
```

```
agent = (
    {
        "input": lambda x: x["input"],
        "agent_scratchpad": lambda x: format_log_to_str(x["agent_scratchpad"]),
    }
    | prompt
    | llm
    | ReActSingleInputOutputParser()
)
```

We create a function to find a tool given the tools array and the name of the tool.

```python
def find_tool(tools, name):
    for tool in tools:
        if tool.name == name:
            return tool
    return None
```

Finally the main loop, where we iterate while we don't find an AgentFinish (we can implement something to avoid infinite loops) and when there is an AgentAction we retrieve the tool, invoke the tool and store the observation in the agent_scratchpad

```python
step = None
while not isinstance(step, AgentFinish):
    step = agent.invoke(
        {
            "input": "What is the length of the word: pistacho",
            "agent_scratchpad": agent_scratchpad,
        }
    )
    print(step)
    if isinstance(step, AgentAction):
        tool_name = step.tool
        tool = find_tool(tools, step.tool)
        observation = tool.func(str(step.tool_input))
        print(f"{observation=}")
        agent_scratchpad.append((step, str(observation)))
```

```
tool='get_text_length' tool_input='pistacho' log='\nThought: The word "pistacho" is
ction: get_text_length\nAction Input: "pistacho"'
observation=8
return_values={'output': 'The length of the word "pistacho" is 8.'} log='Final Answ
```

```python
if isinstance(step, AgentFinish):
    print(step.return_values)
```

```
{'output': 'The length of the word "pistacho" is 8.'}
```

RAG

Introduction

“Garbage in, Garbage out”: even if the LLM input size is big enough, take into account that the longer is the input provided, more money (and/or time) will cost every call to the AI, but also that the more information is in the input, the more garbage you’ll get on the output.

TextLoader

Notebook 3.1

OpenAI Embeddings

Notebook 3.2

HuggingFace Embeddings

Notebook 3.3

FAISS

Notebook 3.4

FAISS Filtering

Notebook 3.5

PDF Ingestion

Notebook 3.6

RetrievalQA

Notebook 3.7

Understanding images

Notebook 3.8