LangChainExamples

February 15, 2024

1 LangChain Example

Illustrate the power of using LangChain with the ReAct approach for doing factual question answering that is supported by a LLM and internet search.

[ ]:
```
!pip install -q langchain
!pip install -q langchain_core
!pip install -q langchain_openai
!pip install -q google-search-results
!pip install -q openai
```

[12]:
```
import os
from dotenv import load_dotenv, find_dotenv

# The .env file contains the serpapi key and any neceesary LLM key.
_ = load_dotenv(find_dotenv(),override = True)

# Input your preferred LLM's information either directly into this cell, or␣
 ↪into a .env file with the names below.
# For GPT-3.5 or GPT-4, comment out base_url.
serper_api_key=os.getenv('SERPAPI_API_KEY')

api_key = "OnuR-l5IlfYqF8HYoTOYHAcHOXCgL5xASQM5ooGHG6A"

base_url = "http://cci-llama1.charlotte.edu/api/v1"
```

1.1 Simple question answering

[13]:
```
from langchain_core.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain_openai import ChatOpenAI


llm = ChatOpenAI(api_key = api_key, base_url = base_url,
                 model = 'Llama-2-70B', temperature= 0, max_tokens = 1000)
```

[14]:
```
# Note how the curly braces { and } are meta-symbols for LangChain (this is not␣
  ↪a formatted string).
query = 'Tell me about {topic}, like I am a college student taking your course.'

prompt = PromptTemplate.from_template(query)

chain = LLMChain(llm=llm, prompt=prompt)

print(chain.invoke({'topic': 'syntactic parsing'})['text'])
```

```
 Sure, I'd be happy to help! In the field of natural language processing,
syntactic parsing is a method for analyzing the structure of sentences and
understanding how words are related to one another. It's a key component of many
NLP applications, such as language translation, sentiment analysis, and text
summarization.

So, let's dive in and explore the basics of syntactic parsing!

**What is syntactic parsing?**

Syntactic parsing is the process of analyzing a sentence and identifying its
underlying structure, including the relationships between the words and how they
are organized. This process is often referred to as "parsing" the sentence. The
goal of syntactic parsing is to identify the underlying grammatical structure of
a sentence, which can help us understand the meaning and context of the words
and phrases within the sentence.

**How does syntactic parsing work?**

There are several approaches to syntactic parsing, but one common method is
called top-down parsing. This approach starts with the overall sentence and
works its way down to the individual words, identifying the relationships
between them as it goes.

Here's a high-level overview of how top-down parsing works:

1. The sentence is fed into a parsing algorithm, which starts by identifying the
highest-level structure of the sentence. This is typically the subject-verb-
object (SVO) structure, which identifies the subject, verb, and object of the
sentence.
2. The parsing algorithm then breaks the sentence down into smaller components,
identifying the individual phrases and clauses within the sentence. For example,
in the sentence "The cat chased the mouse," the phrases would be "the cat"
(subject), "chased" (verb), and "the mouse" (object).
3. The algorithm then identifies the relationships between these phrases and
clauses, such as the subject-verb-object relationship in the example above.
4. Finally, the algorithm outputs a parse tree, which is a visual representation
of the sentence's structure. The parse tree shows the relationships between the
```

```
words and phrases, as well as the overall structure of the sentence.

**What are the benefits of syntactic parsing?**

Syntactic parsing has many applications in natural language processing,
including:

1. Language translation: By understanding the structure of a sentence in one
language, a machine translation system can more accurately translate it into
another language.
2. Sentiment analysis: By analyzing the relationships between words in a
sentence, a sentiment analysis system can better understand the overall
sentiment of the sentence, such as whether it expresses a positive or negative
emotion.
3. Text summarization: By identifying the key phrases and clauses in a sentence,
a text summarization system can create a summary of the most important
information in the text.
4. Grammar and spell checking: Syntactic parsing can help identify grammatical
errors and suggest corrections, such as identifying missing or extra words, or
suggesting alternative sentence structures.

I hope this gives you a good introduction to syntactic parsing! Do you have any
questions or would you like to know more about this topic?
```

1.2 Factual question answering that requires internet search

Some questions, such as 'Who is the current president of the United States?' require data that the LLM did not have access to during pre-training. One solution is to have the LLM determine if it needs data outside of its "memory", and if yes, how to get it, e.g. by running an internet search.

1.2.1 Solution 1: Using LangChain

This sets up our basic prompt. Notice how the strings look like formatted strings, and can be later formatted as such, but they are currently just strings waiting for input.

[15]:
```
system_message = "You are a helpful web search assistant who is skilled at␣
 ↪utilizing the internet." \
" You are also incredibly skilled at information extraction."

instructions = """
Answer the following questions as best you can. You have access to the␣
 ↪following tools:

{tools}

Use the following format:
Question: <the input question>
```

```
Following this, use one of the following options:

Option 1 - You do not have enough information to answer the question:
  Thought: <plans for which tool you will utilize and how you will use it>
  Action: <the action to take, should be one of [{tool_names}]>
  Action Input: <the input to the action>
  Observation: <the result of the action>

Option 2 - You have enough information to answer the question:
  Thought: I now know the final answer
  Final Answer: <the final answer to the original input question>

For example:
  Question: Who composed the Four Seasons?

  Thought: I should utilize serpapi to search the internet for this information.
  Action: serpapi
  Action Input: Who composed The Four Seasons?
  Observation: Antonio Vivaldi

  Thought: I now know the final answer
  Final Answer: Antonio Vivaldi

In another example:
  Question: What is the chemical formula of water?
  Thought: I now know the final answer
  Final Answer: H2O.

Begin!

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

[16]:
```
# Import the necessary classes.
from langchain_core.prompts import ChatPromptTemplate
from langchain.agents import AgentExecutor, create_react_agent, load_tools
from langchain import hub

# Boilerplate code setting up the prompt template, the list of tools, the␣
 ↪agent, and the agent executor.
#    using the 'system_message' and 'instructions' that were instantiated in␣
 ↪the cell above.
prompt = ChatPromptTemplate.from_messages(
    [
        (
            'system',
```

```
            system_message
        ),
        (
            'user',
            instructions
        )
    ]
)

# Input either into a .env file or directly into this cell your API key from␣
 ↪serpapi
# Where to get your API key: https://serpapi.com/
tools = load_tools(['serpapi'], llm=llm, serper_api_key=serper_api_key)

agent = create_react_agent(llm, tools, prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True,␣
 ↪handle_parsing_errors=True, max_iterations= 4)

# Verbose outputs use "Observ" rather than "Observation". The language model␣
 ↪sees "Observation", the output
# of the verbose is meant as a debugging tool and thus prints raw LLM outputs␣
 ↪(stop token 'tion' terminates the call)
# since "Observation: " is added by the library before the next call to the␣
 ↪model.

# Notice how we only pass in "input" and not anything else. The Agent Executor␣
 ↪automatically produces the list of tools
# and tool names, along with the agent_scratchpad, for us.

executor_output = agent_executor.invoke({'input': 'Who is the current president␣
 ↪of the United States?'})

print(executor_output)
print(executor_output['output'])
```

```
> Entering new AgentExecutor chain…
```

```
  Thought: I should utilize a search engine to find the current
president of the United States.
Action: Search
Action Input: "current president of the United States"
Observ['Joseph Robinette Biden Jr is an American politician who
is the 46th and current president of the United States. A member of the
Democratic Party, …', "The first votes in the Republican and Democrat 2024
election primaries and caucuses are due to be cast in Iowa (or, in the
Democrats' case, …", 'The president of the United States (POTUS) is the head
of state and head of government of the United States of America. The president
directs the executive …', 'Three years passed before any Vice President
actually lived at Number One Observatory Circle. Vice President Gerald Ford
acceded to the Presidency before he …', "The Constitution prohibits
Presidential pay changes until the end of the current President's term in
office. Article II, Section 1 of the Constitution states: …", "Opinion polls
show the twice impeached, quadruply indicted ex-president narrowly leading Biden
- and experts say 'it would be a disaster for …", "President Biden and Vice
President Harris promised to move quickly to deliver results for working
families. That's what they've done.", 'President Biden delivers remarks at the
National Veterans Day Observance at the Memorial Amphitheater. Arlington, VA.',
"The White House says President Joe Biden will observe next month's 22nd
anniversary of the worst terrorist attack on U.S. soil at an Alaska
…"]  Thought: I have enough information to answer the
question.
Final Answer: The current president of the United States is Joseph Robinette
Biden Jr.

> Finished chain.
{'input': 'Who is the current president of the United States?', 'output': 'The
current president of the United States is Joseph Robinette Biden Jr.'}
The current president of the United States is Joseph Robinette Biden Jr.
```

1.2.2 Solution 2: Coding it manually using the chat completion API

Note how the LangChain approach above is much shorter and readable than this approach.

Setting up the API wrapper

[ ]:
```python
from typing import Dict
import requests
class SerpAPIClient:
    def __init__(self, api_key, k=3):
        self.api_key = api_key
        self.params = {
            "engine": "google",
            "google_domain": "google.com",
            "gl": "us",
            "hl": "en",
        }
        self.url = "https://serpapi.com/search"
        self.k = k

    # Performs the search.
    def search(self, query):
        params = self.construct_params(query)

        response = requests.get(self.url, params=params)
        res =  response.json()
        return self._process_response(res)

    # Produces the parameters for the API to format its response.
    def construct_params(self, query) -> Dict[str, str]:
        params = self.get_params(query)
        params["source"] = "python"
        if self.api_key:
            params["api_key"] = self.api_key
        params["output"] = "json"
        return params


    # Gets the parameters from the object for the API call.
    def get_params(self, query: str) -> Dict[str, str]:
        """Get parameters for SerpAPI."""
        _params = {
            "api_key": self.api_key,
            "q": query,
        }
        params = {**self.params, **_params}
        return params

    # Takes the API response and gets the organic results, e.g. Wikipedia.
    def _process_response(self, res: dict) -> str:
        """Process response from SerpAPI."""
        num_results = min(len(res['organic_results']), self.k)
```

```
        if "error" in res.keys():
            raise ValueError(f"Got error from SerpAPI: {res['error']}")
        if 'organic_results' in res.keys():
            # create a dictionairy with document and link
            return [{'document_txt': res['organic_results'][i]['snippet'],␣
 ↪'link': res['organic_results'][i]['link']} for i in range(num_results)]
        else:
            return "No good search result found"
```

Test the wrapper, see what the output to the LLM is

[18]:
```
# As before, input your serpapi key in your preferred method.
serpapi = SerpAPIClient(api_key=os.getenv('SERPAPI_API_KEY'))

dictResults = serpapi.search('Who is the current president of the United States?
 ↪')

print(dictResults)
print(dictResults[0]['document_txt'])
```

```
[{'document_txt': 'Learn about the duties of the U.S. president, vice president,
and first lady. Find out how to contact and learn more about current and past
…', 'link': 'https://www.usa.gov/presidents'}, {'document_txt': 'President
Biden represented Delaware for 36 years in the U.S. Senate before becoming the
47th Vice President of the United States. As President, Biden will …', 'link':
'https://www.whitehouse.gov/administration/president-biden/'}, {'document_txt':
'The president of the United States (POTUS) is the head of state and head of
government of the United States of America. The president directs the executive
…', 'link': 'https://en.wikipedia.org/wiki/President_of_the_United_States'}]
Learn about the duties of the U.S. president, vice president, and first lady.
Find out how to contact and learn more about current and past …
```

Now run the ReAct loop It is important to note how many lines of code that the LangChain library saves us in both setting up our tools, but also linking our components together. Even with just a single tool, there is a considerable overhead.

[19]:
```
from openai import OpenAI

serpapi = SerpAPIClient(api_key=serper_api_key)
client = OpenAI(api_key = api_key, base_url = base_url)
conversation = [
{'role': 'system', 'content': system_message},
]
user_input = 'Who is the current president of the United States?'
print("User: " + user_input)
conversation.append({'role': 'user', 'content': instructions.format(tools =␣
 ↪'serpapi', tool_names = 'serpapi', input = user_input, agent_scratchpad =␣
 ↪'')})
```

```
# Produce up to Action Input. If Action Input is not produced, then the model␣
 ↪will continue into a final answer.
thought_and_action = client.chat.completions.create(
    messages = conversation,
    model="Llama-2-70B",
    temperature=0,
    max_tokens=300,
    stop = ['\nAction Input:'],
)
thought_and_action = thought_and_action.choices[0].message.content

# Repeats tool use until the final answer is created.
while 'Action: serpapi' in thought_and_action or 'Action: Search' in␣
 ↪thought_and_action:
    conversation[-1]['content'] += thought_and_action
    action_input = client.chat.completions.create(
        messages = conversation,
        model="Llama-2-70B",
        temperature=0,
        max_tokens=300,
        stop = ['\nObservation:', '\n'],
    )
    action_input = action_input.choices[0].message.content
    conversation[-1]['content'] += action_input
    ind = action_input.index('Action Input:')

    action_input = action_input[ind + 14:]
    observation = str(serpapi.search(action_input))

    # Inject the observation into the conversation.
    conversation[-1]['content'] += "\nObservation: " + observation + "\n" +␣
 ↪"Thought:"

    # LLM either wants to search again or has the information it needs.
    thought_and_action = client.chat.completions.create(
        messages = conversation,
        model="Llama-2-70B",
        temperature=0,
        max_tokens=500,
        stop = [],
    )

    thought_and_action = thought_and_action.choices[0].message.content
    if 'Final Answer:' in thought_and_action:
        break
```

```python
# LLM knows the final answer.
if 'Final Answer:' in thought_and_action:
    print('Full Output: \n\n' + conversation[-1]['content'] + '\n\n')
    # Parse the final answer and print it.
    ind = thought_and_action.index('Final Answer:')
    conversation.append({'role': 'assistant', 'content': thought_and_action})
    print("Llama: " + thought_and_action[ind+14:])
```

```
User: Who is the current president of the United States?
Full Output:


Answer the following questions as best you can. You have access to the following
tools:

serpapi

Use the following format:
Question: <the input question>

Following this, use one of the following options:

Option 1 - You do not have enough information to answer the question:
  Thought: <plans for which tool you will utilize and how you will use it>
  Action: <the action to take, should be one of [serpapi]>
  Action Input: <the input to the action>
  Observation: <the result of the action>

Option 2 - You have enough information to answer the question:
  Thought: I now know the final answer
  Final Answer: <the final answer to the original input question>

For example:
  Question: Who composed the Four Seasons?

  Thought: I should utilize serpapi to search the internet for this information.
  Action: serpapi
  Action Input: Who composed The Four Seasons?
  Observation: Antonio Vivaldi

  Thought: I now know the final answer
  Final Answer: Antonio Vivaldi

In another example:
  Question: What is the chemical formula of water?
  Thought: I now know the final answer
  Final Answer: H2O.
```

```
Begin!

Question: Who is the current president of the United States?
Thought:
  Question: Who is the current president of the United States?

Thought: I should utilize serpapi to search the internet for this information.

Action: serpapi
  Action Input: Who is the current president of the United States?
Observation: [{'document_txt': 'Learn about the duties of the U.S. president,
vice president, and first lady. Find out how to contact and learn more about
current and past …', 'link': 'https://www.usa.gov/presidents'},
{'document_txt': 'President Biden represented Delaware for 36 years in the U.S.
Senate before becoming the 47th Vice President of the United States. As
President, Biden will …', 'link':
'https://www.whitehouse.gov/administration/president-biden/'}, {'document_txt':
'The president of the United States (POTUS) is the head of state and head of
government of the United States of America. The president directs the executive
…', 'link': 'https://en.wikipedia.org/wiki/President_of_the_United_States'}]
Thought:


Llama: The current president of the United States is Joe Biden.
```