Functions, Tools and Agents with LangChain

Function Calling

OpenAI has fine-tuned the *gpt-3.5-turbo-0613* and *gpt-4-0613* models to:

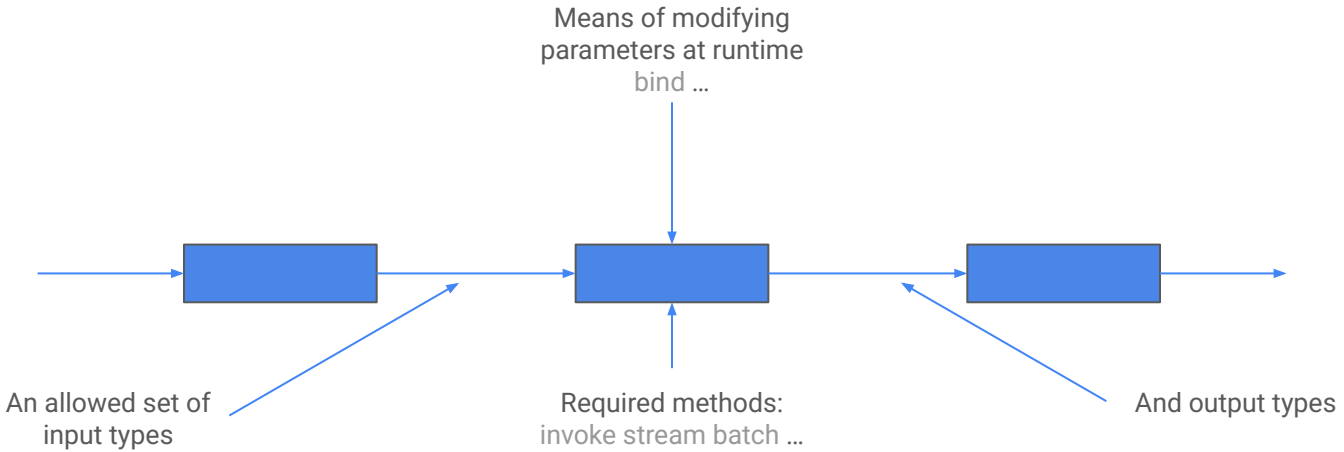
1. Accept additional arguments through which users can pass in descriptions of functions

2. If it is relevant, return the name of the function to use, along with a JSON object with the appropriate input parameters

LangChain Expression Language (LCEL)

LangChain composes chains of components

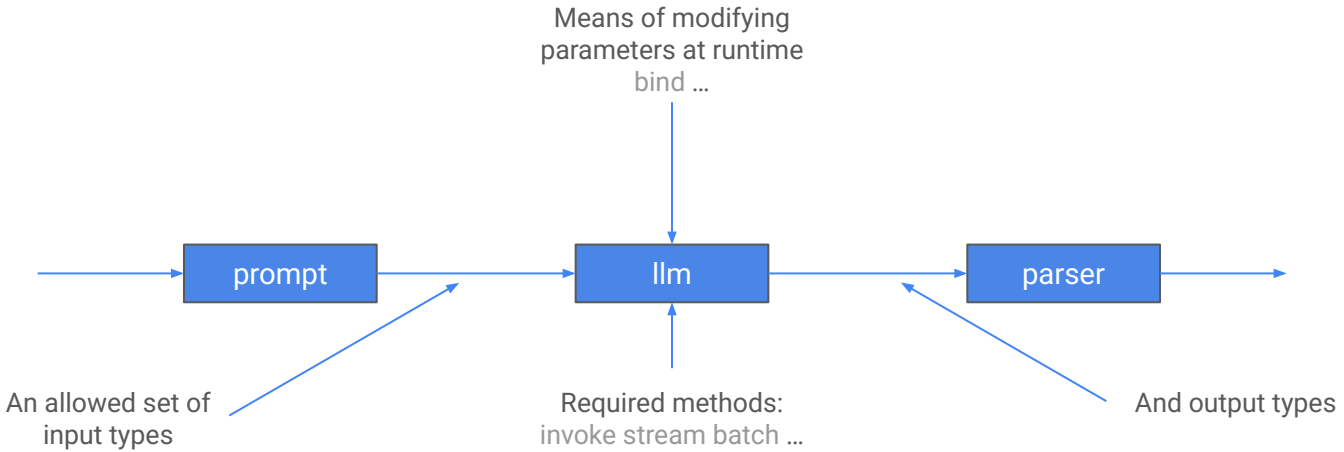
LCEL and the runnable protocol define:



LangChain Expression Language (LCEL)

LangChain composes chains of components

LCEL and the runnable protocol define:



Composition can now use the Linux pipe syntax

```
Chain = prompt | llm | OutputParser
```

Interface

- Components implements “Runnable” protocol
- Common methods include:
  - invoke
  - stream
  - batch
- Common properties
  - input_schema
  - output_schema
- Common I/O

Component	Input Type	Output Type
Prompt	Dictionary	Prompt Value
Retriever	Single String	List of Documents
LLM	String, list of messages or Prompt Value	String
ChatModel	String, list of messages or Prompt Value	ChatMessage
Tool	String / Dictionary	Tool dependent
Output Parser	Output of LLM / ChatModel	Parser dependent

LangChain Expression Language (LCEL)

- Runnables support
  - Async, Batch and Streaming Support
  - Fallbacks
  - Parallelism
    - LLM calls can be time consuming
    - Any component that can be run in parallel are!
  - Logging is built in

Pydantic

Pydantic is a ‘data validation library’ for python.

- Works with python type annotations. But rather than static type checking, they are actively used at runtime for data validation and conversion.
- Provides built-in methods to serialize/deserialize models to/from JSON, dictionaries, etc.
- LangChain leverages Pydantic to create JSON schema describing functions

Pydantic

In normal python you would create a class like this:

```
class User:
    def __init__ (self, name:str, age:int, email:str):
        self.name = name
        self.age = age
        self.email = email
```

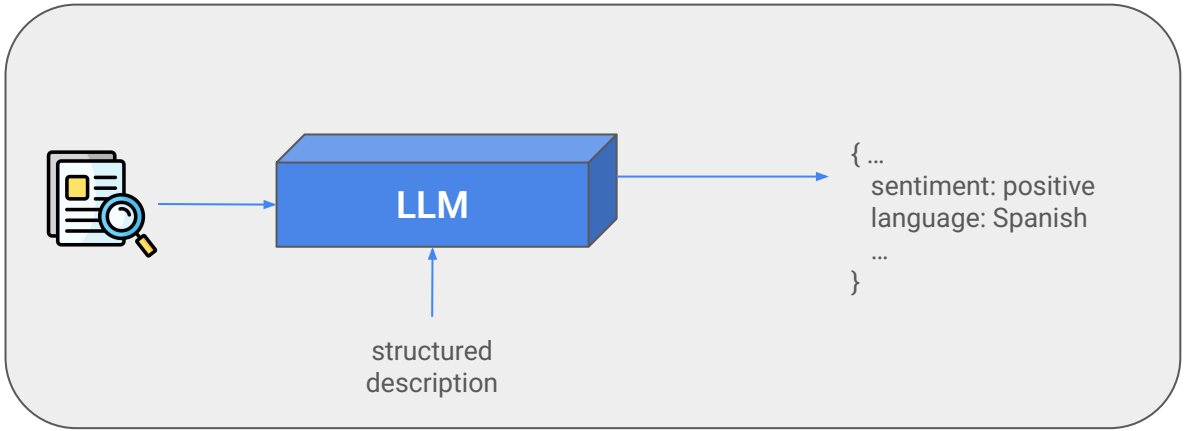
Using Pydantic, this is

```
from pydantic import BaseModel

class User(BaseModel):
    name: str
    age: int
    email: str
```

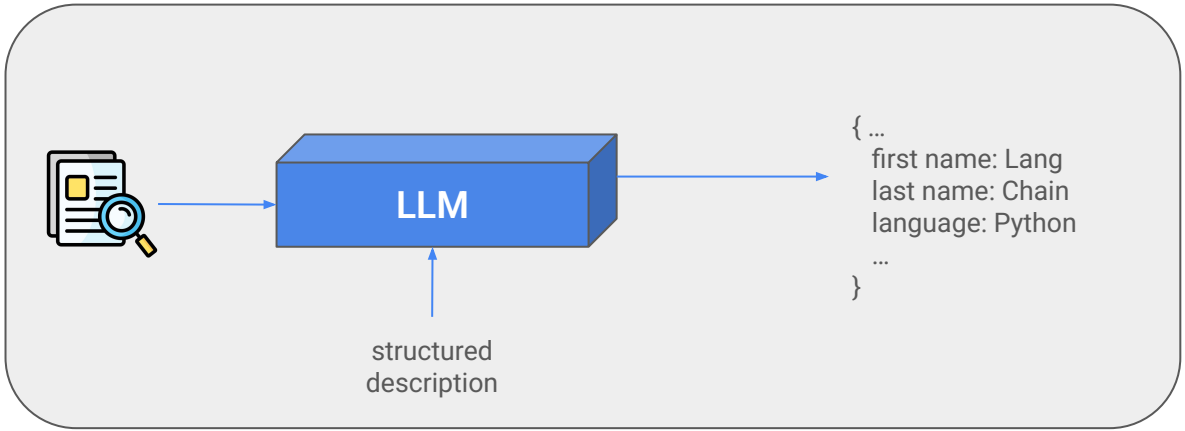
Tagging

- We have seen the LLM, given a function description, select arguments from the input text generate a structured output forming a function call
- More generally, the LLM can evaluate the input text and generate structured output



Extraction

- When given an input JSON schema, the LLM has been fine tuned to find and fill in the parameters of that schema
- The capability is not limited to function schema
- This can be used for general purpose extraction



Tools

- Functions and services an LLM can utilize to extend its capabilities are named “tools” in LangChain

- LangChain has many tools available
  - Search tools
  - Math tools
  - SQL tools
  - Many more

Agent Basics

- Agents
  - Are a combination of LLMs and code
  - LLMs reason about what steps to take and call for actions

- Agent Loop
  - Choose a tool to use
  - Observe the output of the tool
  - Repeat until a stopping condition is met

- Stopping conditions can be
  - LLM determined
  - Hardcoded rules

THANK YOU