Desarrollo de aplicaciones impulsadas por modelo de lenguaje (LLMs)

LangChain ❤️ LLMs.

QR Presentación

Jose Alberto Arango Sánchez

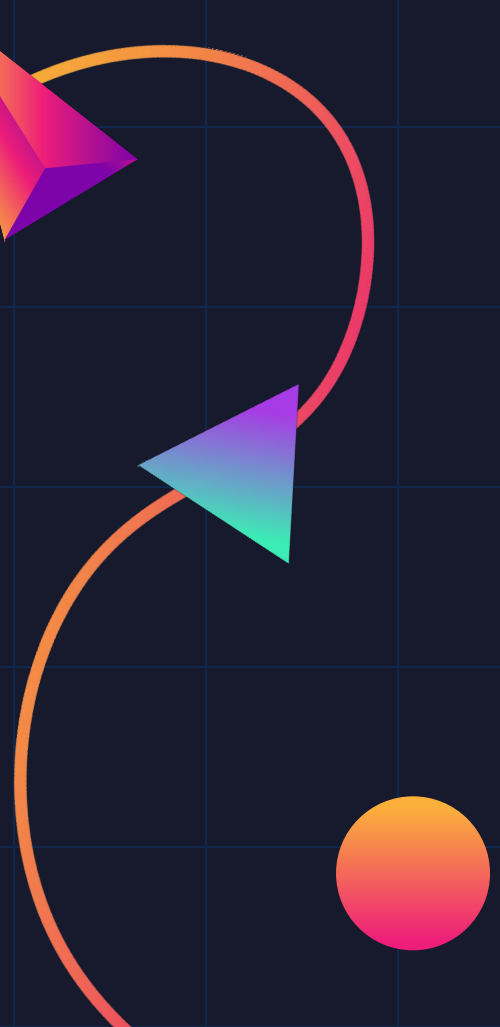
Ingeniero de sistemas - UdeA
Machine Learning Engineer - konecta

@josearangos

Tabla de contenido

01

Demo



02

¿Qué es LangChain?

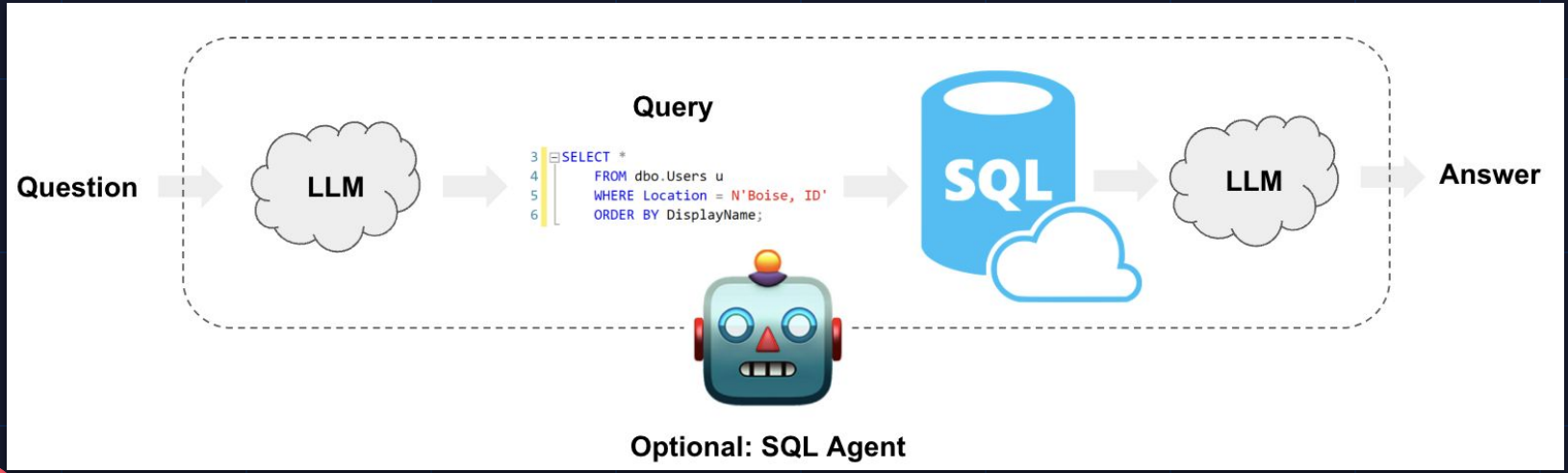
LangChain (LC)

Framework que nos permite crear de aplicaciones usando el poder de los LLMs.

Lang: Lenguaje
Chain: Cadena (Connotación de conectar cosas).

Casos de uso

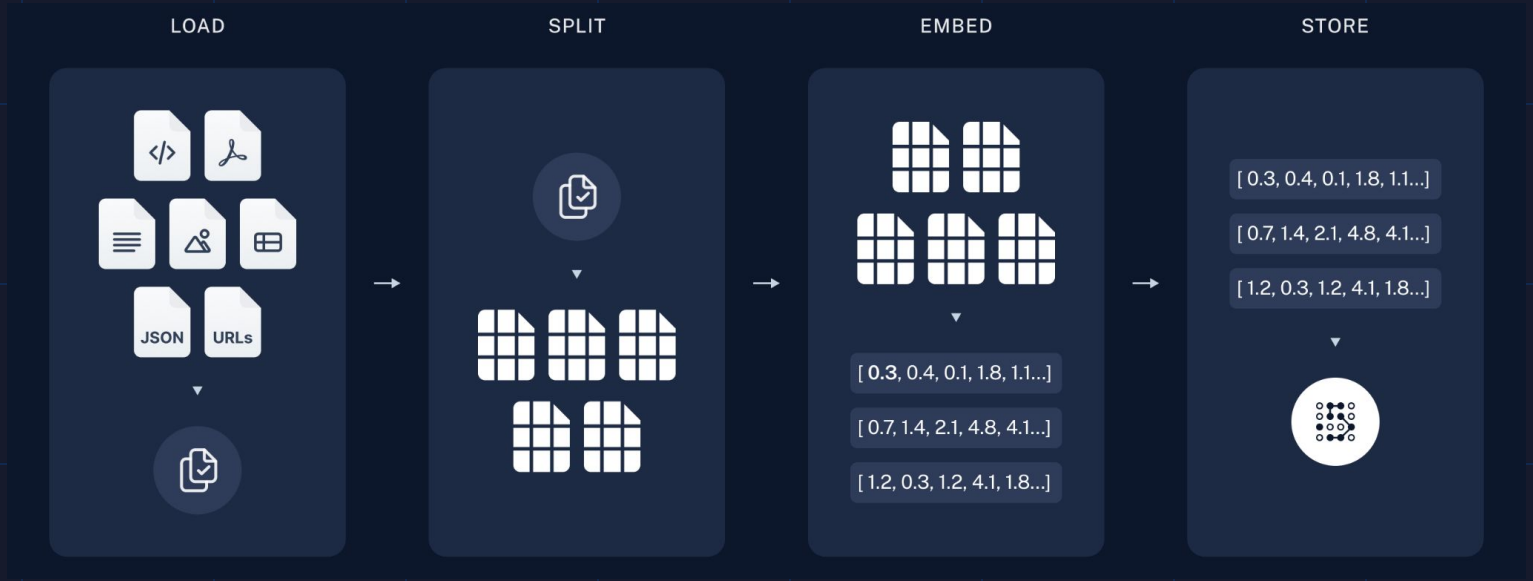
QA over structured data



Casos de uso

Retrieval-augmented generation (RAG)

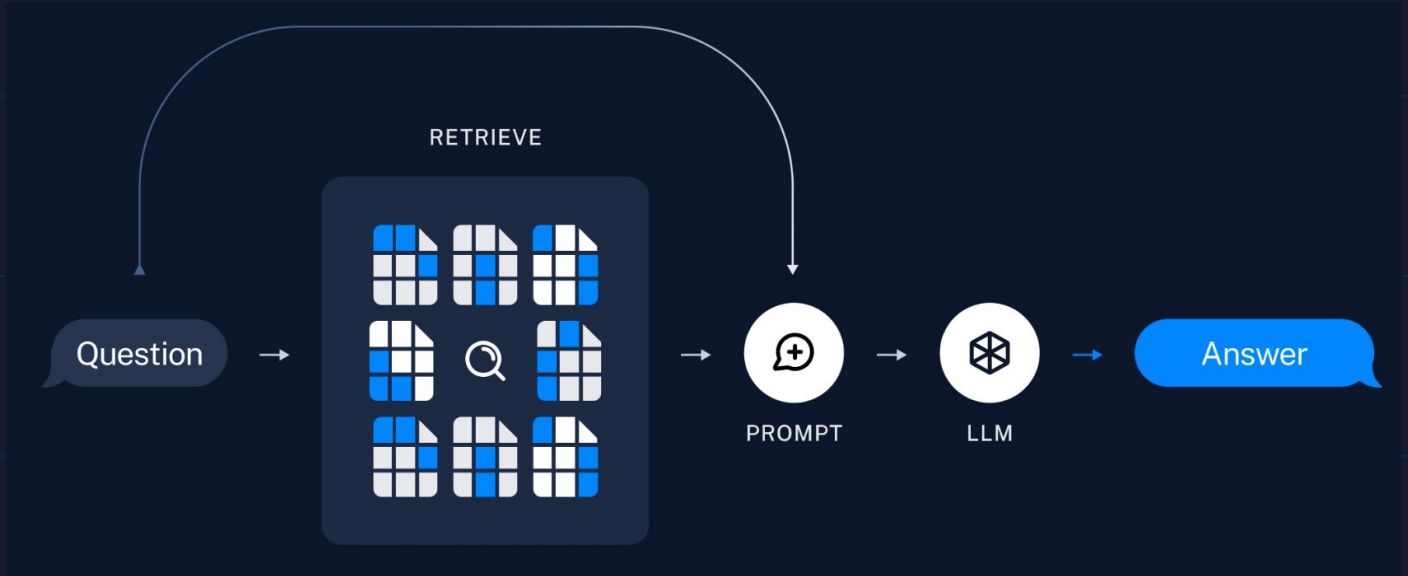
- Indexing



Casos de uso

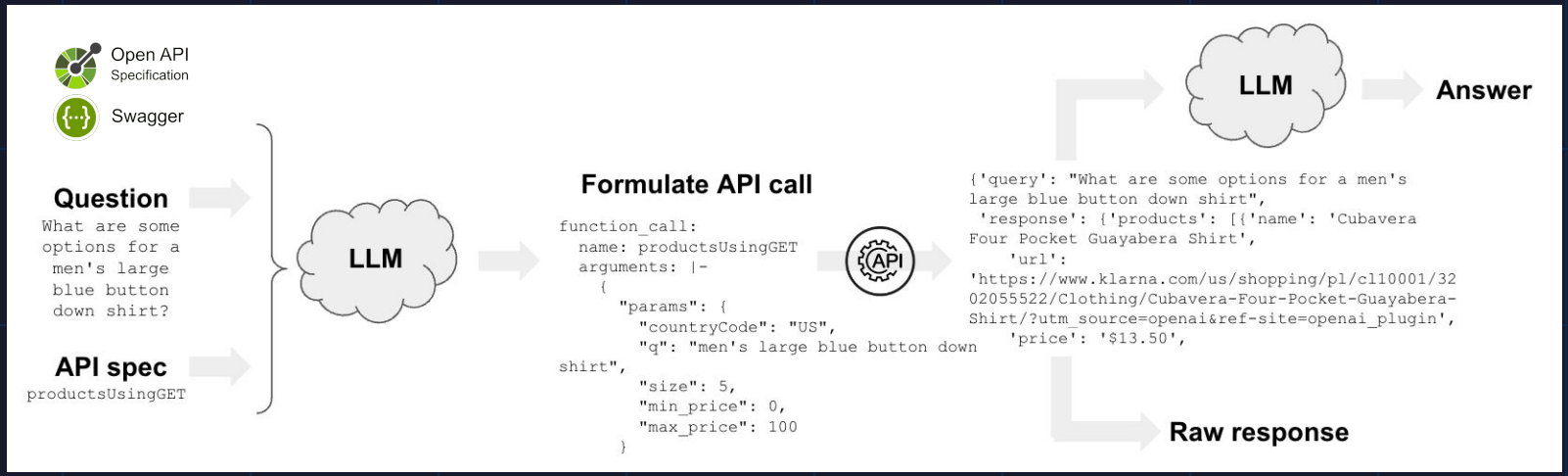
Retrieval-augmented generation (RAG)

- Retrieval and generation



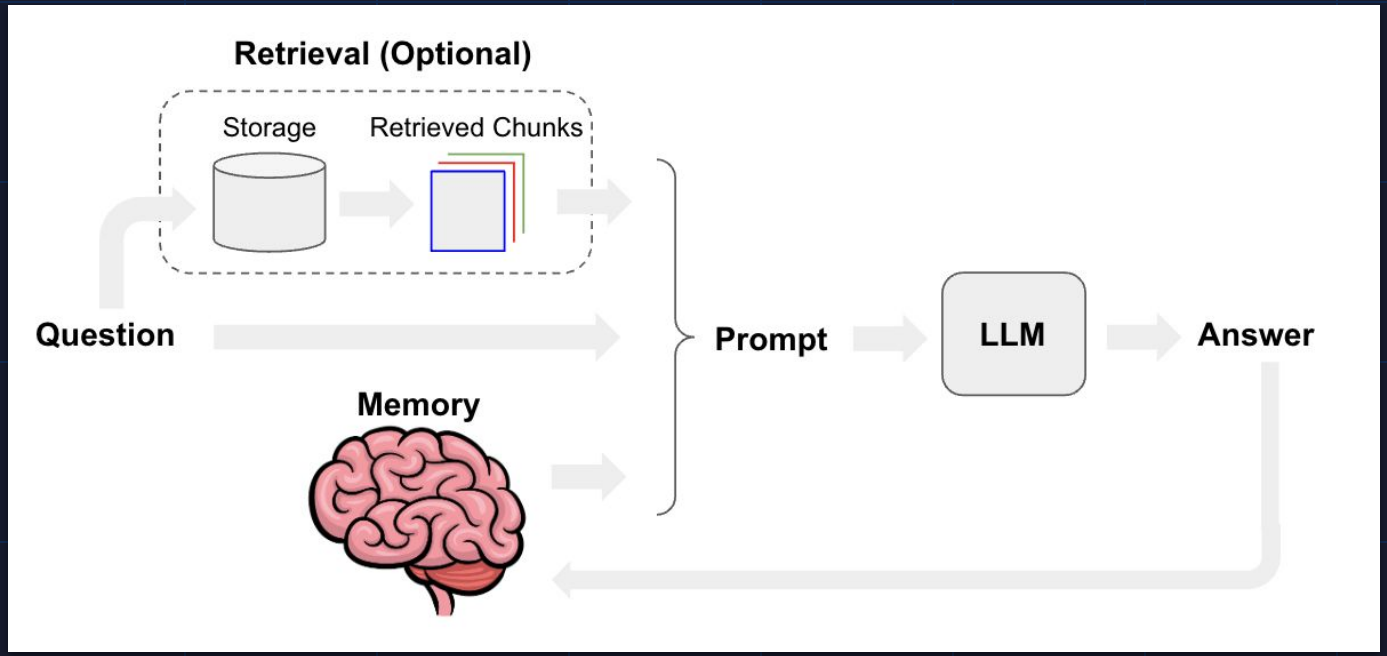
Casos de uso

Interacting with APIs



Casos de uso

ChatBot



Casos de uso

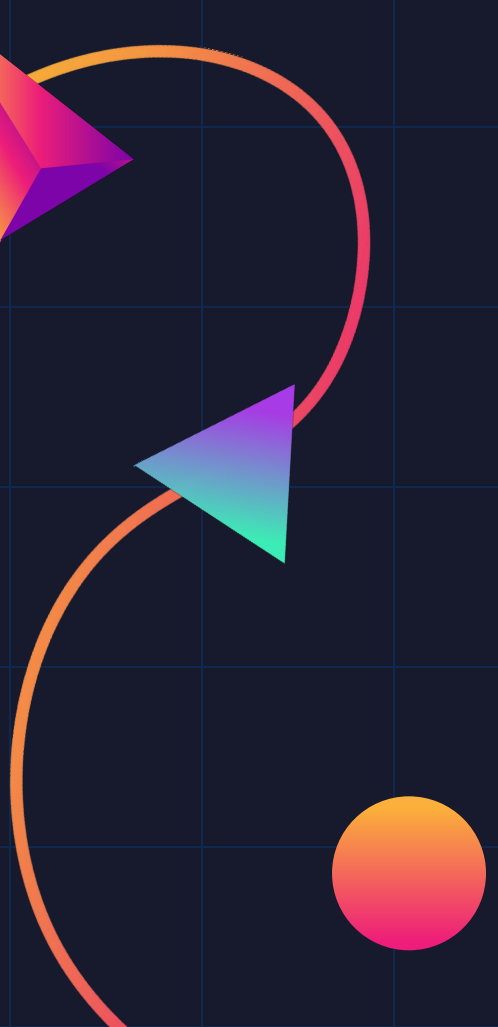
- Extraction
- Summarization
- Tagging
- Etc.

```
#Extraction
{
"name":"Alex",
"height":5,
"hair_color":"blonde"
}

#Tagging
{
"sentiment":"Positive",
"language":"Spanish"
}
```

03

Conceptos y componentes de LangChain



Componentes básicos de LangChain

Models

Prompts

Chains

Memory

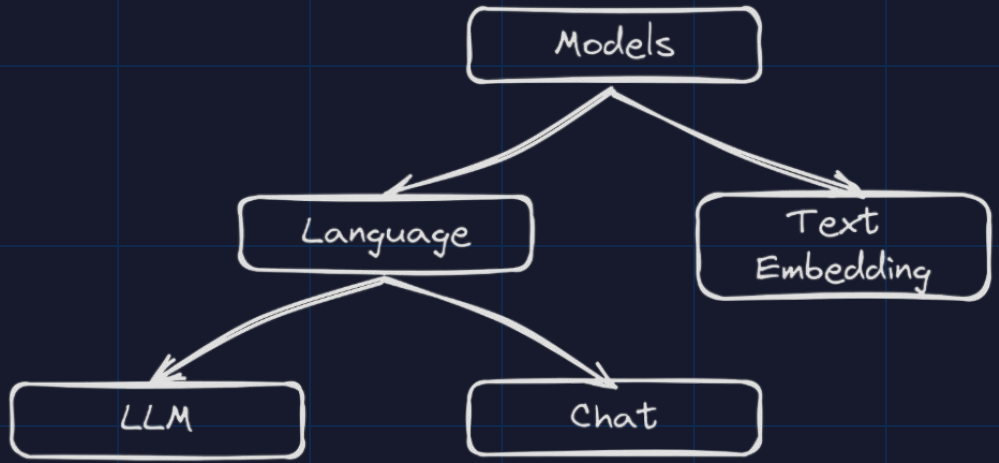
Indexes

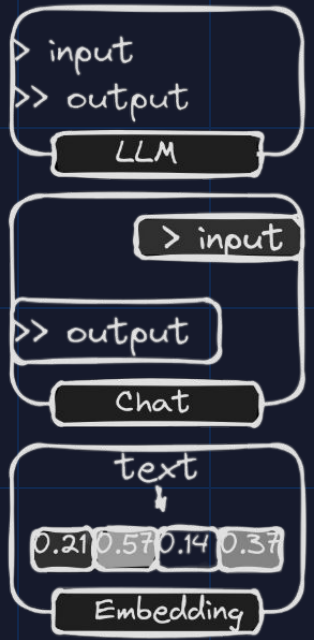
Agents and Tools

Models

Large Language Models (LLMs) son el core de LangChain.
Entrenados en **enormes cantidades** de conjuntos de datos masivos de texto y código.

Ejemplos: GPT-x, Bloom, Flan T5, Alpaca, LLama, etc.





LLMs

```python
from langchain.llms import OpenAI, AzureOpenAI

llm = OpenAI(model_name="text-davinci-003")

text = "What would be a good company name for a company that makes colorful socks?:"
llm.predict(text)
"""
> Rainbow Toes Socks
"""
```

Chat Models

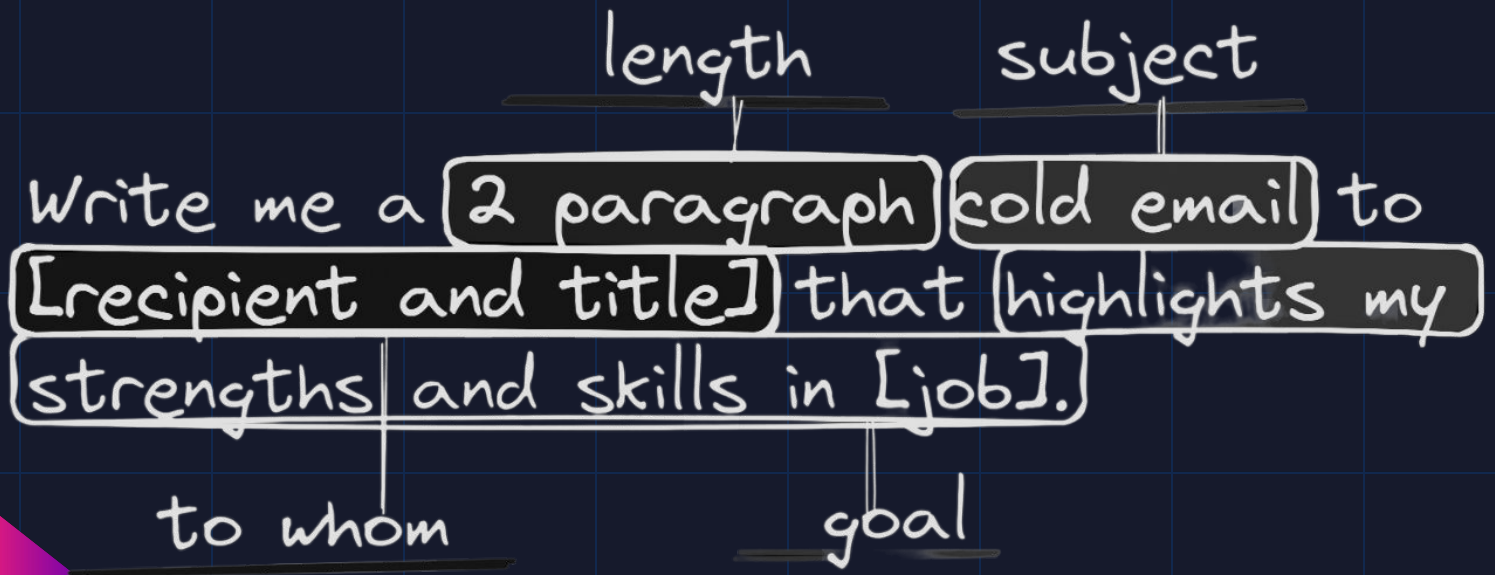
```python
from langchain.chat_models import ChatOpenAI, AzureChatOpenAI
from langchain.schema.messages import HumanMessage, SystemMessage

text = "What would be a good company name for a company that makes colorful socks?"
messages = [
    SystemMessage(content="You're a helpful assistant"),
    HumanMessage(content = text),
]
chat = ChatOpenAI(model_name="gpt-3.5-turbo-1106")
print(chat.predict(messages))
"""
> Rainbow Toe Co.
"""
```

Mayor información: https://python.langchain.com/docs/integrations/llms/

Prompts

Fragmentos de texto que guían al LLM para generar el resultado deseado, se pueden utilizar para generar texto, traducir idiomas, responder preguntas, etc.
En LangChain, los prompts juegan un papel vital en el **control** de la **salida** de los LLM.



Prompt Template

```
from langchain.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(
    "Tell me the names of the most typical meals from: {country}"
)
meals_prompt = prompt_template.format(country="Colombia")
"""
> Tell me the names of the most typical meals from
"""
llm.predict(meals_prompt)
"""
> 1. Bandeja Paisa, 2. Arepas, 3. Sancocho, 4. Ajiaco, 5. Tamales...
"""
```

Chat Prompt Template

```python
from langchain.prompts.chat import ChatPromptTemplate

template = "You are a helpful assistant that translates {input_language} to {output_language}."
human_template = "{text}"

chat_prompt = ChatPromptTemplate.from_messages([
    ("system", template),
    ("human", human_template),
])

messages = chat_prompt.format_messages(input_language="English",
                           output_language="Spanish",
                          text= "Langchain is awesome")
"""
> [
   SystemMessage(content='You are a helpful assistant that translates English to Spanish.'),
   HumanMessage(content='Langchain is awesome')
  ]
"""
chat.predict_messages(messages)
"""
> AIMessage(content='Langchain es increíble.')
"""
```

Algunos ejemplos

Algunos ejemplos de cómo se pueden utilizar los prompts:

- Especifique el formato de salida deseado:
  - Example: Translate the input to Arabic
- Proporcionar contexto:
  - Example: Explain the answer step-by-step like a school teacher
- Restringir la salida:
  - Example: Generate a tweet post using less than 140 words

Chains

Las **cadenas** son **secuencias** de **instrucciones** usadas para realizar una tarea.

- **Simplifica** el uso de los LLMs para tareas específicas
- Permite combinar la **potencia** de los LLMs con otras **técnicas** de **programación**.

Ejemplo 1:

```python
from langchain.prompts.chat import ChatPromptTemplate
from langchain.chat_models import ChatOpenAI

template = "You are a helpful assistant that translates {input_language} to {output_language}."
human_template = "{text}"

translation_prompt = ChatPromptTemplate.from_messages([
    ("system", template),
    ("human", human_template),
])
llm = ChatOpenAI(model_name="gpt-3.5-turbo-1106")
#Chain
translation_chain = translation_prompt | llm

#Run chain
translation_params = {"input_language":"English",
                      "output_language":"Spanish",
                      "text":"Langchain is awesome "}
translation_chain.invoke(translation_params)
```

Ejemplo 2:

```python
from langchain.schema import BaseOutputParser
class CommaSeparatedListOutputParser(BaseOutputParser):
    """Parse the output of an LLM call to a comma-separated list."""
    def parse(self, text: str):
        """Parse the output of an LLM call."""
        return text.strip().split(", ")
```

```python
from langchain.prompts.chat import ChatPromptTemplate
from langchain.chat_models import ChatOpenAI

template = """You are a helpful assistant who generates comma separated lists.
A user will pass in a country, and you should generate 5 food of that country in a comma separated list.
ONLY return a comma separated list, and nothing more."""

human_template = "{country}"
meals_prompt = ChatPromptTemplate.from_messages([
    ("system", template),
    ("human", human_template),
])

llm = ChatOpenAI(model_name="gpt-3.5-turbo-1106")
meals_chain = meals_prompt | llm | CommaSeparatedListOutputParser()

chain.invoke({"country":"Colombia"})
# >['Arepas', 'Bandeja Paisa', 'Empanadas', 'Ajiaco', 'Lechona']
```

LangChain Expression Language (LCEL).

Forma declarativa para componer cadenas fácilmente, utilizando el símbolo |, similar a los pipelines de los sistemas Unix:

Razones para usarlo:

1. Streaming support
2. Async support
3. Optimized parallel execution
4. Retries and fallbacks

LCEL Interface - Sync

```
#CHAIN
meals_chain = meals_prompt | llm

#Sync
-----------------------------------------------------------------
# Invoke
-----------------------------------------------------------------
meals_chain.invoke({"country":"colombia"})
-----------------------------------------------------------------
# Stream
for s in meals_chain.stream({"country": "Colombia"}):
    print(s.content, end="", flush=True)
-----------------------------------------------------------------
# Batch
meals_chain.batch([{"country" :"Colombia"},
            {"country" :"Argentina"},
            {"country" :"Brasil"}])
```

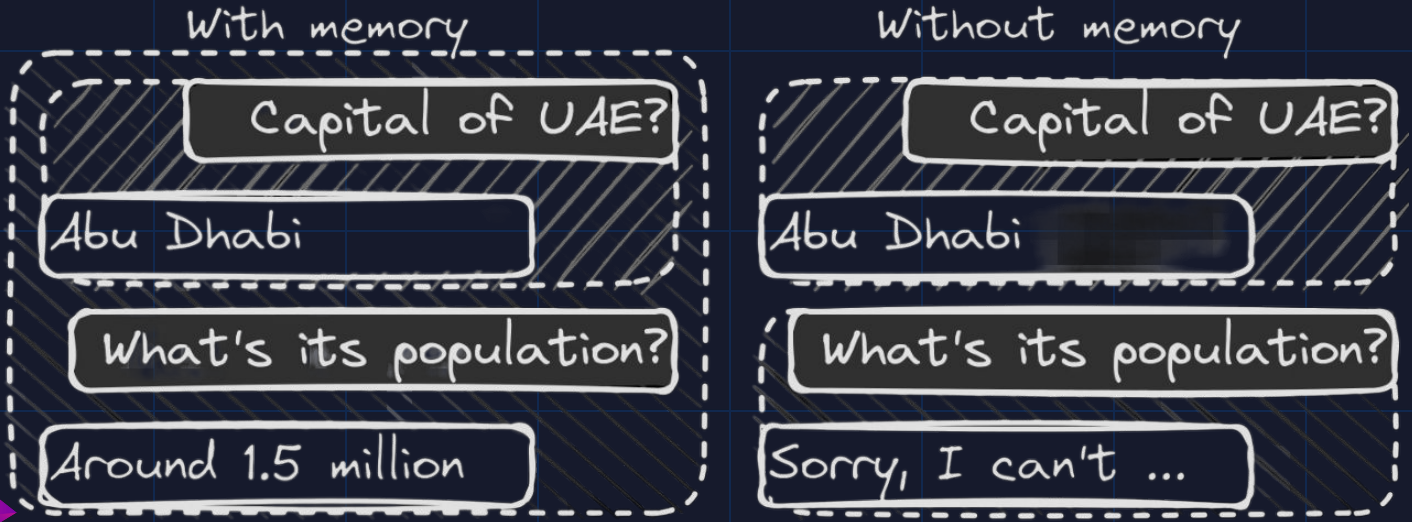
LCEL Interface - Async

```
#CHAIN
meals_chain = meals_prompt | llm

#Async
-----------------------------------------------------------------
await meals_chain.ainvoke({"country":"colombia"})
-----------------------------------------------------------------
# AStream
async for s in meals_chain.astream({"country": "Colombia"}):
    print(s.content, end="", flush=True)
-----------------------------------------------------------------
# ABatch
await meals_chain.abatch([{"country" :"Colombia"},
            {"country" :"Argentina"},
            {"country" :"Brasil"}])
```

Memory

- La memoria es el concepto de almacenar y recuperar datos en el proceso de una conversación.
- Permitiendo al LLM aprender de interacciones previas y construir una **base de conocimiento.**



Agents and Tools

- **Agentes:** Los agentes son componentes reutilizables que pueden realizar tareas específicas como la generación de textos, la traducción de idiomas y la respuesta a preguntas.
- **Tools:** Las herramientas son bibliotecas de funciones que pueden utilizarse como ayuda para desarrollar distintos agentes.

Agents and Tools

La idea central de los agentes es utilizar un **LLM** para elegir una **secuencia** de **acciones a realizar**.

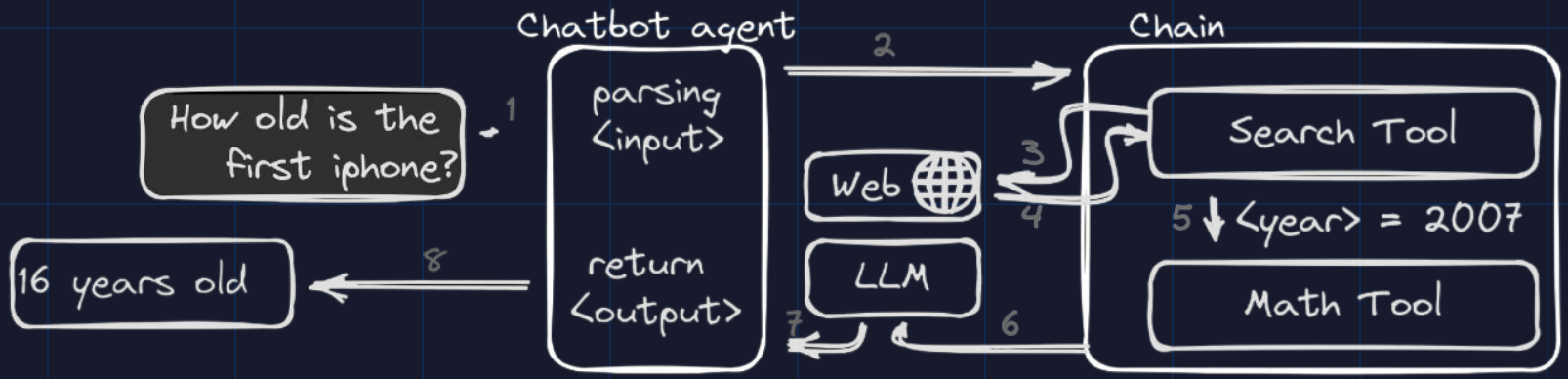
En los agentes, se utiliza un modelo de lenguaje como **motor** de "**razonamiento"** para determinar **qué acciones** realizar y en qué orden.
Ejemplos de agentes:

1. create_csv_agent
2. create_pandas_dataframe_agent
3. create_python_agent
4. create_sql_agent
5. create_spark_dataframe_agent

etc.

Agents and Tools

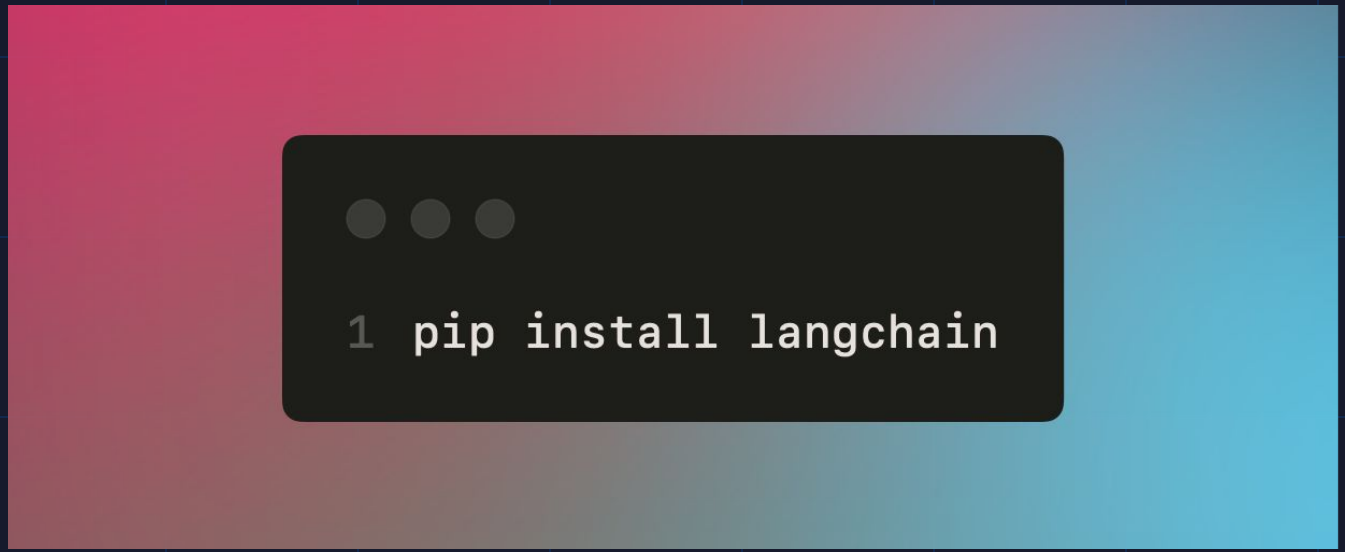
"agente" tiene acceso a un conjunto de herramientas. Dependiendo de la entrada del usuario, el agente puede decidir a cuál de estas herramientas llamar.



04

¿Cómo construir una aplicación con LangChain?

Instalación



Importamos módulos

```
1 from langchain import OpenAI
2 from langchain.chat_models import ChatOpenAI
3 from langchain.agents import initialize_agent
4 from langchain.sql_database import SQLDatabase
5 from langchain.agents.agent_types import AgentType
6 from langchain.agents.agent_toolkits import SQLDatabaseToolkit
7 from langchain.agents.agent_toolkits import GmailToolkit
```

DB Toolkit

```python
db = SQLDatabase.from_uri("sqlite:///resources/users.db")
db_toolkit = SQLDatabaseToolkit(db=db, llm=OpenAI(temperature=0))
print(db_toolkit.get_tools())
```

```
[QuerySQLDataBaseTool(description="Input to this tool is a detailed and correct SQL query, output is a result from the database. I
 InfoSQLDatabaseTool(description="Input to this tool is a comma-separated list of tables, output is the schema and sample rows for
 ListSQLDatabaseTool(db=<langchain.utilities.sql_database.SQLDatabase object at 0x17ca36f70>),
 QuerySQLCheckerTool(description='Use this tool to double check if your query is correct before executing it. Always use this tool
```

Gmail Toolkit

```
gmail_toolkit = GmailToolkit()
print(gmail_toolkit.get_tools())
# OUTPUT
[GmailCreateDraft(api_resource=<googleapiclient.discovery.Resource object at 0x120b6b3d0>),
 GmailSendMessage(api_resource=<googleapiclient.discovery.Resource object at 0x120b6b3d0>),
 GmailSearch(api_resource=<googleapiclient.discovery.Resource object at 0x120b6b3d0>),
 GmailGetMessage(api_resource=<googleapiclient.discovery.Resource object at 0x120b6b3d0>),
 GmailGetThread(api_resource=<googleapiclient.discovery.Resource object at 0x120b6b3d0>)]
```

Agent

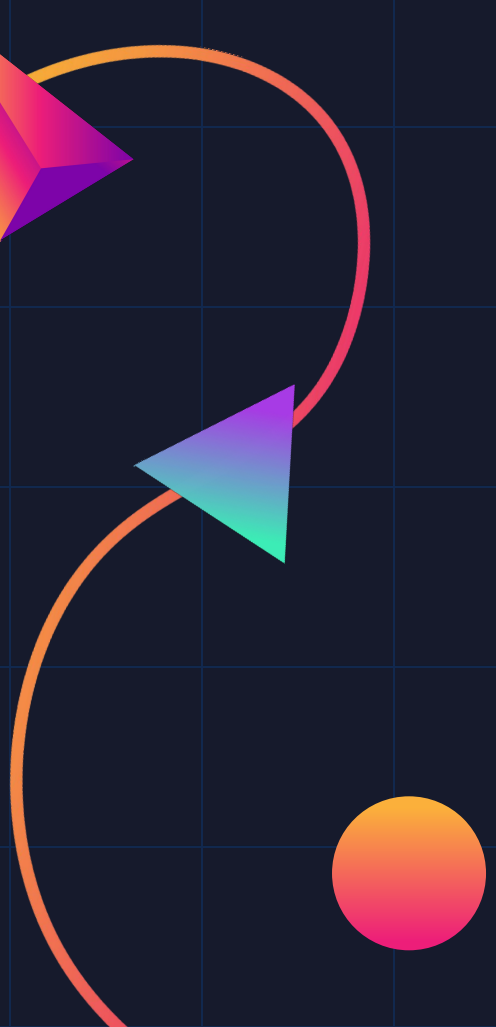
```
llm=ChatOpenAI(temperature=0,model_name="gpt-4")

agent = initialize_agent(
    llm=llm,
    tools=gmail_toolkit.get_tools()+db_toolkit.get_tools(),
    verbose=True,
    agent=AgentType.STRUCTURED_CHAT_ZERO_SHOT_REACT_DESCRIPTION,
)

agent.run('Busca los 2 usuarios que más han comprado y envía el siguiente correo:'
          '\nAsunto: Código de descuento por compras'
          '\nContenido: "Hola, gracias por tus compras. Tienes un bono de descuento en tu siguiente compra.'
          '\nUtiliza el código: #LLMConEsteroides y obtén un 20% de descuento."')
```

05

Documentación



Recursos aprender Langchain

1. Documentación oficial
2. LangChain for LLM Application Development
3. LangChain: Chat with Your Data

DeepLearning.AI

LangChain



06

Recursos de interés

Recursos de interés

Deploy langchain app

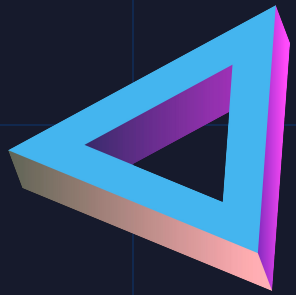
- https://github.com/msoedov/langcorn
- https://github.com/langchain-ai/langserve

UI for LangChain

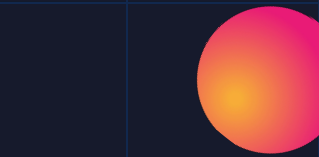
- https://github.com/logspace-ai/langflow
- https://flowiseai.com

LLMOps

- https://github.com/BerriAI/litellm
- https://github.com/pezzolabs/pezzo
- https://github.com/tensorchord/Awesome-LLMOps
- https://github.com/KennethanCeyer/awesome-llmops






Muchas Gracias !!!

Alguna pregunta ?