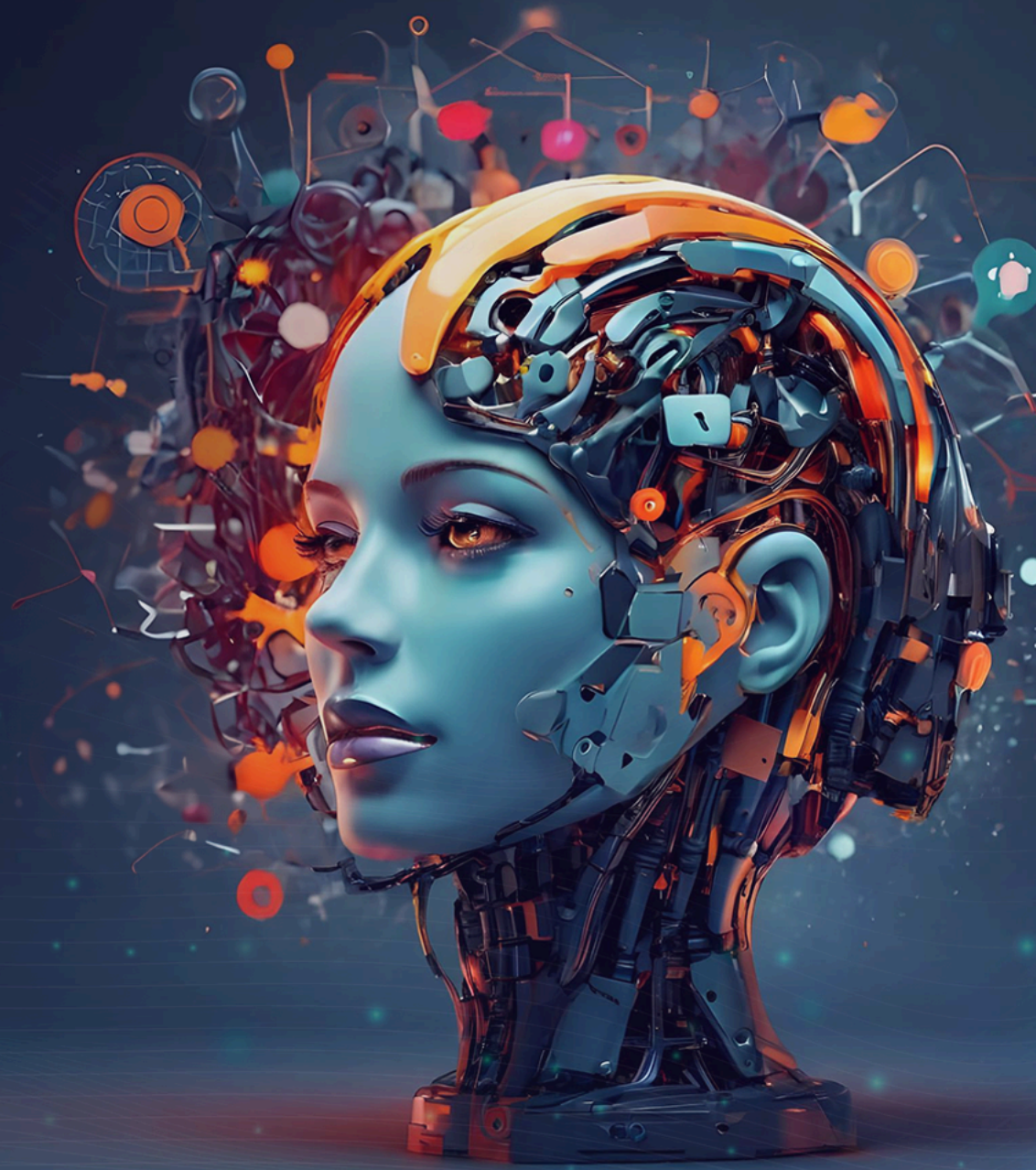
Analytics Vidhya

Future Outlook of LangChain



Instructor

Dipanjan Sarkar

Head of Community & Principal AI Scientist at Analytics Vidhya

Google Developer Expert - ML & Cloud Champion Innovator

Published Author

Outline

- LangChain’s Recent Event Timeline
- Future Outlook for LangChain
- LCEL vs. Regular Implementation - LLM Invoke
- LCEL vs. Regular Implementation - LLM Streaming
- LCEL vs. Regular Implementation - LLM Batching
- LCEL vs. Regular Implementation - LLM Async Calls
- LCEL vs. Regular Implementation - Calls to other LLMs

LangChain's Recent Event Timeline

Start

August, 2023

Introduction of LangChain Expression Language (LCEL), new features like MultiVectorRetriever and caching Embeddings

September, 2023

Routing capabilities in LCEL and LangChain rewrote all major AI agent types using LCEL

October, 2023

Introduction of app templates to be used with LangServe

December, 2023

Major changes in the LangChain library, we see a split of the larger langchain library into:

langchain-core: Core LangChain abstractions as well as LangChain Expression Language

langchain-community: Third party integrations into various components like vector DBs

langchain: Key components to build apps like chains, agents, and retrieval methods

Analytics Vidhya

LangChain’s Recent Event Timeline

January, 2024

Release of LangGraph and LangChain v0.1

Feb-March, 2024

Improvements and new features in LangGraph and LangChain

April, 2024

Standardization of the Tool calling interface, improvements to LangGraph and LangSmith

Analytics Vidhya

Future Outlook for LangChain

- Legacy Chains are slowly being deprecated
- LangChain officially is migrating most complex workflows into LCEL
- LangChain is marketing itself as a whole ecosystem of tools including:
  - LangChain
  - LangGraph
  - LangSmith
  - LangServe
- Expect more changes in the future given we are still in early stages of Generative AI itself

LCEL vs. Regular Implementation - LLM Invoke

Without LCEL

```
from typing import List

import openai


prompt_template = "Tell me a short joke about {topic}"
client = openai.OpenAI()

def call_chat_model(messages: List[dict]) -> str:
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
    )
    return response.choices[0].message.content

def invoke_chain(topic: str) -> str:
    prompt_value = prompt_template.format(topic=topic)
    messages = [{"role": "user", "content": prompt_value}]
    return call_chat_model(messages)

invoke_chain("ice cream")
```

LCEL

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough


prompt = ChatPromptTemplate.from_template(
    "Tell me a short joke about {topic}"
)
output_parser = StrOutputParser()
model = ChatOpenAI(model="gpt-3.5-turbo")
chain = (
    {"topic": RunnablePassthrough()}
    | prompt
    | model
    | output_parser
)

chain.invoke("ice cream")
```

Simplest example of passing a prompt to an LLM

LCEL vs. Regular Implementation - LLM Streaming

Without LCEL

```
from typing import Iterator


def stream_chat_model(messages: List[dict]) -> Iterator[str]:
    stream = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        stream=True,
    )
    for response in stream:
        content = response.choices[0].delta.content
        if content is not None:
            yield content

def stream_chain(topic: str) -> Iterator[str]:
    prompt_value = prompt.format(topic=topic)
    return stream_chat_model([{"role": "user", "content":
prompt_value}])


for chunk in stream_chain("ice cream"):
    print(chunk, end="", flush=True)
```

LCEL

```
for chunk in chain.stream("ice cream"):
    print(chunk, end="", flush=True)
```

Streaming results in LCEL just needs to call another function on the same chain

LCEL vs. Regular Implementation - LLM Batching

Without LCEL

```python
from concurrent.futures import ThreadPoolExecutor


def batch_chain(topics: list) -> list:
    with ThreadPoolExecutor(max_workers=5) as executor:
        return list(executor.map(invoke_chain, topics))

batch_chain(["ice cream", "spaghetti", "dumplings"])
```

LCEL

```python
chain.batch(["ice cream", "spaghetti", "dumplings"])
```

Batching inputs in parallel in LCEL has built-in support instead of implementing new logic

LCEL vs. Regular Implementation - LLM Async Calls

Without LCEL

```
async_client = openai.AsyncOpenAI()

async def acall_chat_model(messages: List[dict]) -> str:
    response = await async_client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
    )
    return response.choices[0].message.content

async def ainvoke_chain(topic: str) -> str:
    prompt_value = prompt_template.format(topic=topic)
    messages = [{"role": "user", "content": prompt_value}]
    return await acall_chat_model(messages)


await ainvoke_chain("ice cream")
```

LCEL

```
await chain.ainvoke("ice cream")
```

Making Asynchronous calls in LCEL is easy with built-in support instead of implementing new logic

LCEL vs. Regular Implementation - Calls to other LLMs

Without LCEL

```
import anthropic

anthropic_template =
f"Human:\n\n{prompt_template}\n\nAssistant:"
anthropic_client = anthropic.Anthropic()

def call_anthropic(prompt_value: str) -> str:
    response = anthropic_client.completions.create(
        model="claude-2",
        prompt=prompt_value,
        max_tokens_to_sample=256,
    )
    return response.completion

def invoke_anthropic_chain(topic: str) -> str:
    prompt_value = anthropic_template.format(topic=topic)
    return call_anthropic(prompt_value)

invoke_anthropic_chain("ice cream")
```

LCEL

```
from langchain_anthropic import ChatAnthropic

anthropic = ChatAnthropic(model="claude-2")
anthropic_chain = (
    {"topic": RunnablePassthrough()}
    | prompt
    | anthropic
    | output_parser
)

anthropic_chain.invoke("ice cream")
```

LCEL enables you to use the same chain and just switch out the LLM in the LLM step

Thank You

Analytics Vidhya