Picture credit — LangChain

Understanding LangGraph — Note-2

@Anil's Notes Follow 5 min read · Mar 24, 2024

15

In this second note, we're picking up where we left off in "Understanding LangGraph — Note-1.". we'll dive into LangGraph by piecing together a basic RAG (Retrieval Augmented Generation) pipeline. To make sure it's easy for you to follow along, we'll be using the Google Serper API and an OpenAI key that you need as prerequisites to try.

Understanding LangGraph — Note-1

LangChain is an open-source framework designed to facilitate the creation of AI applications based on large language…

anilktalla.medium.com

Basic RAG sample (using LangGraph)

```
# install modules
!pip install langchain_openai python-dotenv langgraph

#setup environment variables - prerequisites
os.environ["OPENAI_API_KEY"] = "REPLACE_ME"
os.environ["SERPER_API_KEY"] = "REPLACE_ME"

# Sample-3 -  Basic RAG pipeline
from langgraph.graph import Graph
from langchain_openai import ChatOpenAI
from langchain.utilities import GoogleSerperAPIWrapper
from langchain.agents import Tool
import operator
from typing import Annotated, Sequence, TypedDict
from langchain_core.messages import BaseMessage
from langchain_openai import ChatOpenAI
from langgraph.graph import StateGraph, END
from langchain_core.messages import HumanMessage, SystemMessage
```

```
# Set the model as ChatOpenAI
model = ChatOpenAI(temperature=0)

#Define agent state
class AgentState(TypedDict):
    messages: Annotated[Sequence[BaseMessage], operator.add]

# Define the function that retrieves information from serper
def retrieve_results(state):
  messages = state["messages"]
  # Based on the continue condition
  # we know the last message involves a function call
  last_message = messages[-1]
  search_serper_api_wrapper = GoogleSerperAPIWrapper()
  search_tool = Tool(name="Search", func=search_serper_api_wrapper.run, descript
  response = search_tool.invoke(last_message)
  return {"messages": [response]}

# Define the function that calls the LLM to generate
def generate_answer(state):
    messages = state['messages']
    last_message = messages[-1]
    prompt_input_messages = [
      SystemMessage(
          content="You are a helpful assistant that can understand the given mes
      ),
      HumanMessage(
          content=last_message
      ),
    ]
    response = model.invoke(prompt_input_messages)
    # We return a list, because this will get added to the existing list
    return {"messages": [response]}


# Define a new graph
workflow = StateGraph(AgentState)

# Define the two nodes we will cycle between
workflow.add_node("retriever", retrieve_results)
workflow.add_node("generate", generate_answer)

workflow.add_edge("retriever", "generate")

# Set the entrypoint as `agent` where we start
workflow.set_entry_point("retriever")
workflow.set_finish_point("generate")

app = workflow.compile()
```

```
input = {"messages": ["What is the most expensive car Elon musk drives?"]}

for output in app.stream(input):
    for key, value in output.items():
        print(f"Output from node '{key}':")
        print("---")
        print(value)
    print("\n---\n")

final_response = app.invoke(input)["messages"][-1].content
print(final_response)
```

Output:

```
Output from node 'retriever':
---
{'messages': ['The most expensive car currently in Elon\'s collection, then, is

---

Output from node 'generate':
---
{'messages': [AIMessage(content="Elon Musk's most expensive car is a Lotus Espri

---

Output from node '__end__':
---
{'messages': ['What is the most expensive car Elon musk drives?', 'The most expe

---

Elon Musk's most expensive car is a Lotus Esprit, purchased for $920,000. He dri
```

Code step by step explanation

1. Importing Libraries and Modules

Importing essential python modules from LangGraph, LangChain, and other libraries like ChatOpenAI, GoogleSerperAPIWrapper, and operator to facilitate various functionalities within the workflow.

2. Setting Up the ChatOpenAI Model

"generator" using LLM model.

3. Defining Agent State and Functions

- **AgentState:** A typed dictionary is introduced to manage the agent's state, storing a sequence of messages crucial for processing.
- **retrieve_results:** This function retrieves information using GoogleSerperAPIWrapper based on the last message in the agent's state.
- **generate_answer**: The function generates an answer utilizing the ChatOpenAI model, leveraging the last message in the agent's state for context.

4. Creating a StateGraph Workflow

A StateGraph is instantiated with AgentState to oversee the agent's state transitions and data flow management throughout the execution.

5. Adding Nodes and Edges

Nodes "**retriever**" and "**generate**" are incorporated into the workflow to represent distinct functions that cyclically interact during processing. An edge is established from “retriever” to “generate” to signify the flow of data between these nodes.

6. Setting Entry and Finish Points

The entry point is designated as “retriever,” marking the starting point of the workflow, while “generate” serves as the finish point where the final response is generated.

7. Compiling and Executing the Workflow

The workflow undergoes compilation into an executable application ready for processing input data. An initial message regarding news on LangGraph is provided to initiate the workflow’s execution.

8. Displaying Outputs

During execution, outputs from each node are systematically printed to track the processing steps. The final response generated by the workflow encapsulates the essence of all preceding interactions, culminating in a comprehensive output for further analysis or action.

In the next note, we will explore expanding basic RAG to more advanced RAG pipeline.

Langgraph

Written by @Anil's Notes

176 followers · 257 following

Follow

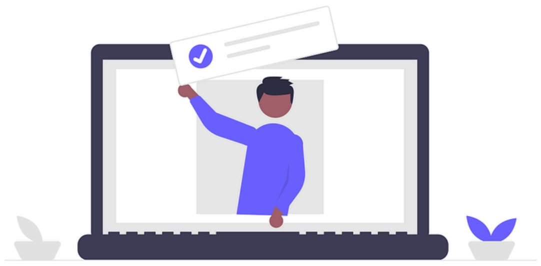
Thoughts I add here are my personal notes and learnings only

No responses yet

Rajendra Bichu

What are your thoughts?

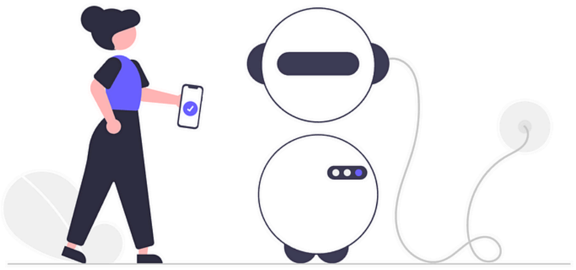
More from @Anil's Notes



@Anil's Notes

Prompt engineering -1— Shot prompting

Continuing from my previous article on LLM concepts, I am writing about prompt…



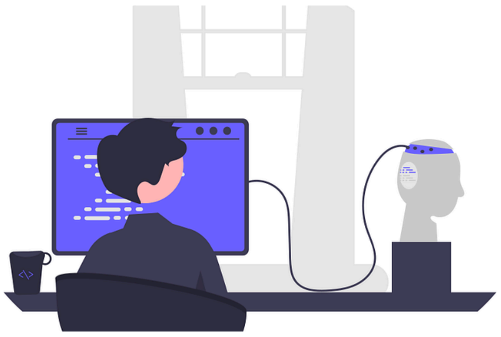
@Anil's Notes

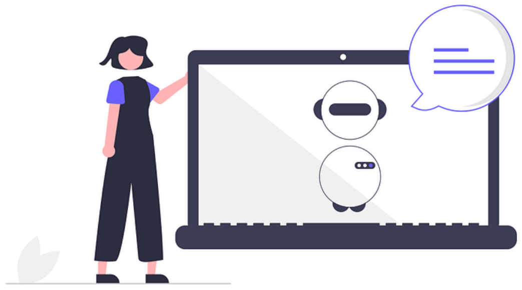
AI protocols: MCP, ACP, and A2A — My notes

If you've been following AI development recently, you've probably heard buzzwords…

Jul 24, 2023 46

May 21 11





@Anil's Notes

LLM Concepts—My notes

Last week one evening, my 9-year-old daughter came running towards me, excited…

May 28, 2023 4

@Anil's Notes

Semantic Kernel concepts—My notes (part-1)

Earlier, I was primarily focused on LangChain and related frameworks. However, my intere…

Jun 30, 2024 4

See all from @Anil's Notes

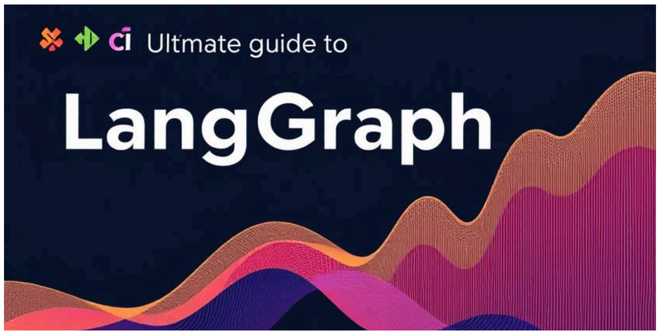
Recommended from Medium

Kaizin

How to build LLM Agent with LangGraph—StateGraph and…

LLM agent is LLM application that selects appropriated tools to solve tasks by itself…

Feb 1



In Dev Genius by Neural pAi

🚀 The Ultimate Guide to LangGraph: All Aspects Explained

LangGraph is an innovative framework designed to create, manage, and execute…

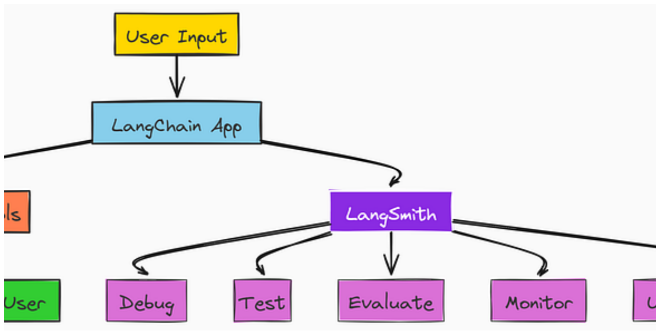
Feb 23 106

In The Pythonworld by Aashish Kumar

What One Python Interview Taught Me About Real-World Tech

This one interview made me realize what actually matters in production code and tea…

4d ago 54



In Level Up Coding by Fareed Khan

Building a Multi-Agent AI System with LangGraph and LangSmith

A step-by-step guide to creating smarter AI with sub-agents

6d ago 833 12

Ravi Sankar Uppala

LangGraph

You ask your AI assistant to calculate compound interest on your savings, and it…

Apr 29

Nidhi

LangGraph 101

Introduction

Apr 23

See more recommendations