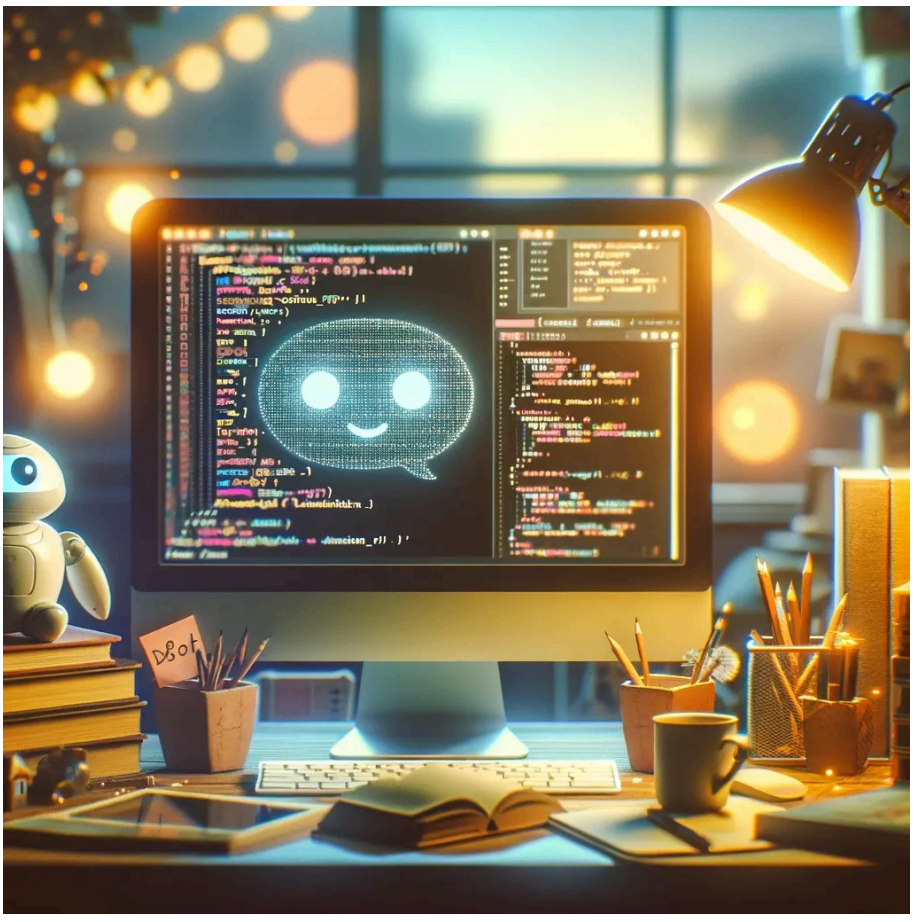
Introduction to LangGraph: A Beginner's Guide

CPlog Follow 3 min read · Feb 14, 2024



LangGraph is a powerful tool for building stateful, multi-actor applications with Large Language Models (LLMs). It extends the LangChain library, allowing you to coordinate multiple chains (or actors) across multiple steps of computation in a cyclic manner. In this article, we'll introduce LangGraph, walk you through its basic concepts, and share some insights and common points of confusion for beginners.

What is LangGraph?

LangGraph is a library built on top of LangChain, designed to add cyclic computational capabilities to your LLM applications. While LangChain allows you to define chains of computation (Directed Acyclic Graphs or DAGs), LangGraph introduces the ability to add cycles, enabling more complex, agent-like behaviors where you can call an LLM in a loop, asking it what action to take next.

Key Concepts

- **Stateful Graph:** LangGraph revolves around the concept of a stateful graph, where each node in the graph represents a step in your computation, and the graph maintains a state that is passed around and updated as the computation progresses.
- **Nodes:** Nodes are the building blocks of your LangGraph. Each node represents a function or a computation step. You define nodes to perform specific tasks, such as processing input, making decisions, or interacting with external APIs.
- **Edges:** Edges connect the nodes in your graph, defining the flow of computation. LangGraph supports conditional edges, allowing you to dynamically determine the next node to execute based on the current state of the graph.

A Simple Example

Let's walk through a simple example where we use LangGraph to classify user input as either a "greeting" or a "search" query and respond accordingly.

Step 1: Define the Graph State

First, we define the state structure for our graph. In this example, our state includes the user's question, the classification of the question, and a response.

```
from typing import Dict, TypedDict, Optional
class GraphState(TypedDict):
    question: Optional[str] = None
    classification: Optional[str] = None
    response: Optional[str] = None
```

Step 2: Create the Graph

Next, we create a new instance of `StateGraph` with our `GraphState` structure.

```
from langgraph.graph import StateGraph
```

```
workflow = StateGraph(GraphState)
```

Step 3: Define Nodes

We define nodes for classifying the input, handling greetings, and handling search queries.

```
def classify_input_node(state):
    question = state.get('question', '').strip()
    classification = classify(question)  # Assume a function that classifies the
    return {"classification": classification}
def handle_greeting_node(state):
    return {"response": "Hello! How can I help you today?"}
def handle_search_node(state):
    question = state.get('question', '').strip()
    search_result = f"Search result for '{question}'"
    return {"response": search_result}
```

Step 4: Add Nodes to the Graph

We add our nodes to the graph and define the flow using edges and conditional edges.

```
workflow.add_node("classify_input", classify_input_node)
workflow.add_node("handle_greeting", handle_greeting_node)
workflow.add_node("handle_search", handle_search_node)

def decide_next_node(state):
    return "handle_greeting" if state.get('classification') == "greeting" else "
workflow.add_conditional_edges(
    "classify_input",
    decide_next_node,
    {
        "handle_greeting": "handle_greeting",
        "handle_search": "handle_search"
    }
)
```

Step 5: Set Entry and End Points

We set the entry point for our graph and define the end points.

```
workflow.set_entry_point("classify_input")
workflow.add_edge('handle_greeting', END)
workflow.add_edge('handle_search', END)
```

Step 6: Compile and Run the Graph

Finally, we compile our graph and run it with some input.

```
app = workflow.compile()
inputs = {"question": "Hello, how are you?"}
result = app.invoke(inputs)
print(result)
```

Common Confusions

- **State Management:** Understanding how the state is passed around and updated in the graph can be tricky. Remember that each node receives the current state, can modify it, and passes it on to the next node.
- **Conditional Edges:** Setting up conditional edges requires careful consideration of the conditions and the mapping of outcomes to the next nodes. Ensure that the keys returned by the condition function match the keys in the conditional edge mapping.
- **Dead-End Nodes:** Every node in the graph should have a path leading to another node or to the `END` node. If a node has no outgoing edges, it's considered a dead-end, and you'll need to add an edge to avoid errors.

Conclusion

LangGraph is a versatile tool for building complex, stateful applications with LLMs. By understanding its core concepts and working through simple examples, beginners can start to leverage its power for their projects. Remember to pay attention to state management, conditional edges, and ensuring there are no dead-end nodes in your graph. Happy coding!

Langchain | Llm | AI | Chatbots | Python

Written by CPlog

77 Followers · 58 Following

Follow

AI enthusiast & Principal Data Scientist at LFX & Li & Fung. I innovate with LLMs, LangChain, Stable Diffusion & ComfyUI to revolutionize supply chains.

Responses (10)

Thomas Modern

What are your thoughts?

Kiss Tibor (Cloud Mentor) he/him
Sep 20, 2024

Great introduction. Code does not work

11 2 replies Reply

TELIKICHERLA R.K.KAMESWARA SARMA
Jul 22, 2024

what does classify() do in this ? it functionality missing in the article. How do we classify it as Search/greeting? Please help on classify(question)

20 3 replies Reply

Abubakar
Nov 22, 2024

Executable code:

from typing import Dict, TypedDict, Optional

from langgraph.graph import StateGraph, END

class GraphState(TypedDict, total=False):

question: str

classification: str

response: str

def classify(question: str) -> str:

return "greeting" if... more

32 Reply

See all responses

More from the list: "LLM"

Curated by Thomas Modern

Edwin Lisowski
What Every AI Engineer Should Know About A2...
Apr 23

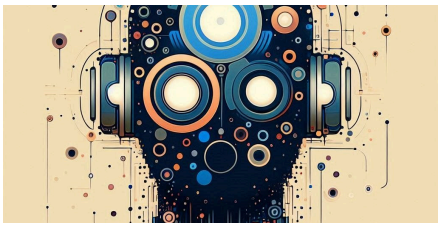
In Google Cloud... by Heik...
Getting Started with Google A2A: A Hands-o...
Apr 15

Minyang Chen
Google's A2A Protocol: Seamless Agent...
· Apr 18

AI Clou...
Bu Ap
Apr

View list

More from CPlog



CPlog

Introduction to AI Agent with LangChain and LangGraph: A...

Master Tool-Calling AI with LangGraph: A Comprehensive Guide to Building...

Aug 13, 2024 86 1

CPlog

Faker Data Generation

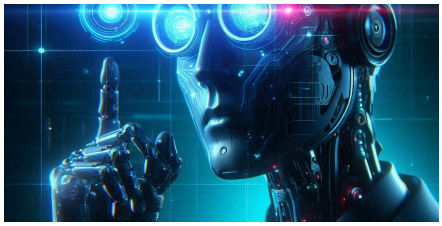
Feb 17, 2024 145

CPlog

Exposing Your Local Service to the Internet with Cloudflare Tunnel: A...

This tutorial will show you how to securely make a locally hosted service—such as an...

Apr 13



CPlog

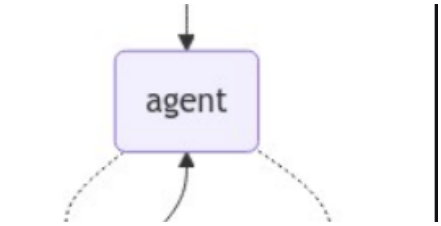
HOST YOUR WEBSITE AS GITHUB.IO

Hugo + Github = Your own static website (blog)

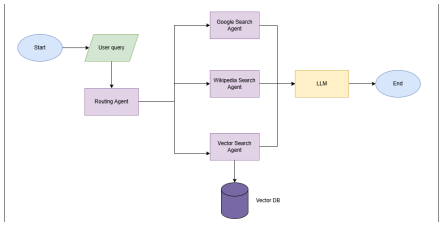
Jun 6, 2018 155

See all from CPlog

Recommended from Medium



Story_Teller



Janindu Munasinghe

Introduction to Tool Use with LangGraph's ToolNode

Modern AI applications often require seamless integration of external tools to...

Dec 14, 2024 11 4

Building a Simple Multi-Agent Platform Using Llama and...

In today's interconnected digital world, multi-agent systems are becoming increasingly...

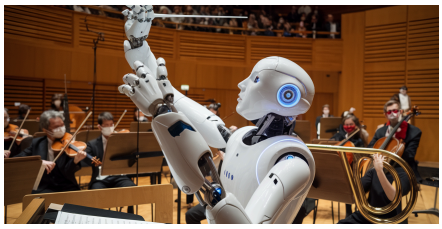
Jan 5 181

Lovelyn David

LangGraph Series-2-Creating a Conversational Bot with Memory...

In the previous article, we explored how to construct a graph using LangGraph. In this...

Apr 11 16 1

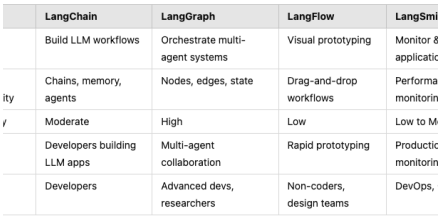


Wendell Rodrigues, Ph.D.

Building an Agentic RAG with LangGraph: A Step-by-Step Guide

Leverage LangGraph to orchestrate a powerful Retrieval-Augmented Generation...

Jan 14 15 2



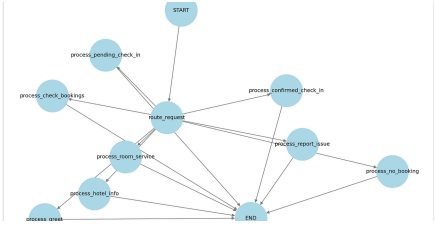
	LangChain	LangGraph	LangFlow	LangSmi
	Build LLM workflows	Orchestrate multi-agent systems	Visual prototyping	Monitor & applicatic
ity	Chains, memory, agents	Nodes, edges, state	Drag-and-drop workflows	Performa monitorin
/	Moderate	High	Low	Low to M
	Developers building LLM apps	Multi-agent collaboration	Rapid prototyping	Productic monitorin
	Developers	Advanced devs, researchers	Non-coders, design teams	DevOps,

Anshuman

LangChain vs LangGraph vs LangFlow vs LangSmith: A Detaile...

In the rapidly evolving world of AI, building applications powered by advanced languag...

Jan 17 55



WS

LangGraph Simplified: Understanding Conditional edge...

In the evolving landscape of AI and machine learning, LangGraph has emerged as a...

Feb 25 2

See more recommendations