What is LangGraph?

Sahitya Arya

Last Updated : 04 Sep, 2024

Introduction

Artificial intelligence (AI) is a rapidly developing field. As a result, language models have advanced to a point where AI agents are able to perform complex tasks and make complex decisions. However, as these agents' skills have grown, the infrastructure that supports them has found it difficult to keep up. Presenting LangGraph, a revolutionary library that aims to revolutionize AI agent building and runtime execution.

Overview

- LangGraph is a library built on top of Langchain that is designed to facilitate the creation of cyclic graphs for large language model (LLM) – based AI agents.
- It views agent Objective Points about LangGraph and workflows as cyclic graph topologies, allowing for more variable and nuanced agent behaviors than linear execution models

- The library supports multi-agent coordination, allowing each agent to have its prompt, LLM, tools, and custom code within a single graph structure.
- Langgraph introduces a chat agent executor that represents the agent state as a list of messages, which is particularly useful for newer, chat-based models.

The Pre-LangGraph Era

The agent executor class in the Langchain framework was the main tool for building and executing AI agents before LangGraph. This class relied on a straightforward but powerful idea: it used an agent in a loop, asking it to make decisions, carry them out, and log observations. This technique had its uses, but its adaptability and customization possibilities were intrinsically restricted.

Although functional, the agent executor class limited developers' ability to design more dynamic and flexible agent runtimes by imposing a particular pattern of tool calling and error handling. As AI agents became more sophisticated, the need for a more adaptable architecture emerged.

What is LangGraph?

In response to these constraints, LangGraph presents a novel paradigm for agent building and runtime construction. Large Language Models (LLMs) are the foundation for designing sophisticated AI agents, and LangGraph, built on top of Langchain, is intended to make the process of creating cyclic graphs easier.

LangGraph views agent workflows as cyclic graph topologies at their foundation. This method enables more variable and nuanced behaviors from agents, surpassing its predecessors' linear execution model. Using graph theory, LangGraph provides new avenues for developing intricate, networked agent systems.

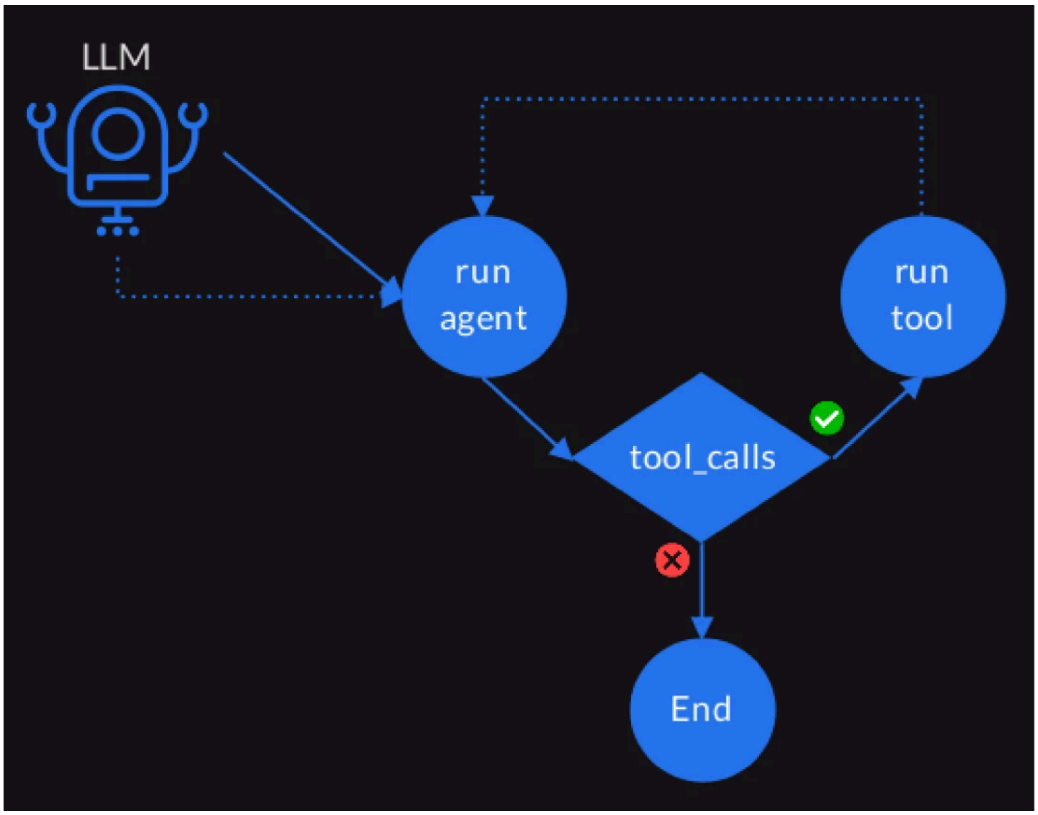
- **Flexibility:** As AI agents evolved, developers required more control over the agent runtime to enable personalized action plans and decision-making procedures.
- **The Cyclical Nature of AI Reasoning:** Many intricate LLM applications depend on cyclical execution when employing strategies like chain-of-thought reasoning. LangGraph offers a natural framework for modeling these cyclical processes.
- **Multi-Agent Systems:** As multi-agent workflows became more common, there was an increasing demand for a system that could efficiently manage and coordinate several autonomous agents.

State Management: As agents became more sophisticated, tracking and updating state data as the agent was being executed became necessary. LangGraph's stateful graph methodology satisfies this need.

How LangGraph Works?

The functionality of LangGraph is based on several essential elements:

- **Nodes:** These are functions or Langchain runnable items, like the agent's tools.
- **Edges:** Paths that define the direction of execution and data flow within the agent system, connecting nodes.
- **Stateful Graphs:** LangGraph allows for persistent data across execution



LangGraph Working

As shown in the image, the nodes include LLM, tools, etc. which are represented by circles or rhombus which. Flow of information between various nodes is represented by arrows.

The popular NetworkX library served as the model for the library's interface, which makes it user-friendly for developers with prior experience with graph-based programming.

LangGraph's approach to agent runtime differs significantly from that of its forerunners. Instead of a basic loop, it enables the construction of intricate, networked systems of nodes and edges. With this structure, developers can design more complex decision-making procedures and action sequences.

Tool Calling in LangGraph

Pre-requisites

Create an OpenAI API key to access the LLMs and Weather API key (get here) to access the weather information. Store these keys in a '.env' file:

Load and Import the keys as follows:

Copy Code

```
import os

from dotenv import load_dotenv

load_dotenv('/.env')

WEATHER_API_KEY = os.environ['WEATHER_API_KEY']

# Import the required libraries and methods

import json

import requests

import rich

from typing import List, Literal

from IPython.display import Image, display

from langchain_community.tools.tavily_search import TavilySearchResults

from langchain_core.tools import tool

from langchain_openai import ChatOpenAI
```

know the answer to the given query:

Copy Code

```
@tool
def get_weather(query: str) -> list:

    """Search weatherapi to get the current weather."""

    base_url = "http://api.weatherapi.com/v1/current.json"
    complete_url = f"{base_url}?key={WEATHER_API_KEY}&q={query}"
    response = requests.get(complete_url)
    data = response.json()

    if data.get("location"):
        return data
    else:
        return "Weather Data Not Found"

@tool
def search_web(query: str) -> list:

    """Search the web for a query."""

    tavily_search = TavilySearchResults(max_results=2, search_depth='advanced', m

    results = tavily_search.invoke(query)
    return results
```

To make these tools available for the LLM, we can bind these tools to the LLM as follows:

Copy Code

```
gpt = ChatOpenAI(model="gpt-4o-mini", temperature=0)

tools = [search_web, get_weather]

gpt_with_tools = gpt.bind_tools(tools)
```

```
        Do not change the query given for any search tasks

        1. What is the current weather in Greenland today

        2. Can you tell me about Greenland and its capital

        3. Why is the sky blue?
    """

results = gpt_with_tools.invoke(prompt)

results.tool_calls
```

The results will be the following:

```
[74]: results.tool_calls

[74]: [{'name': 'get_weather',
        'args': {'query': 'Greenland'},
        'id': 'call_hjNLdx7tWelqAuBwgXB5Ci9N',
        'type': 'tool_call'},
       {'name': 'search_web',
        'args': {'query': 'who won the ICC worldcup in 2024?'},
        'id': 'call_qZxe4LBHVaadkATOfde1pRRq',
        'type': 'tool_call'},
       {'name': 'search_web',
        'args': {'query': 'Why is the sky blue?'},
        'id': 'call_x7LO3yHH64DwKHdCXsi0LXgX',
        'type': 'tool_call'}]

[16]: query = """who won the ICC worldcup in 2024?"""
      response = gpt.invoke(query)
      response.content

[16]: "As of my last update in October 2023, the ICC Men's Cricket World Cup 2024 had not yet taken pl
      ace, so I do not have information on the winner. The tournament is scheduled to be held in the W
      est Indies and the USA. For the latest updates, please check the most recent sources."
```

As we see, when we ask about the weather, get_weather tool is called.

The GPT model doesn't know the who won ICC worldcup in 2024, as it is updated with information upto October 2023 only. So, when we ask about this

LangGraph has pre-built react (reason and act) agent. Let’s see how it works:

Copy Code

```
from langgraph.prebuilt import create_react_agent

# system prompt is used to inform the tools available to when to use each

system_prompt = """Act as a helpful assistant.

             Use the tools at your disposal to perform tasks as needed.

               - get_weather: whenever user asks get the weather of a place.

               - search_web: whenever user asks for information on current events

             Use the tools only if you don't know the answer.

          """

# we can initialize the agent using the gpt model, tools, and system prompt.

agent = create_react_agent(model=gpt, tools=tools, state_modifier=system_prompt)

# We will discuss its working in the next section. Let’s query the agent to see t

def print_stream(stream):
    for s in stream:
        message = s["messages"][-1]
        if isinstance(message, tuple):
            print(message)
        else:
            message.pretty_print()

inputs = {"messages": [("user", "who won the ICC worldcup in 2024?")]}

print_stream(agent.stream(inputs, stream_mode="values"))
```

```
inputs = {"messages": [("user", "who won the ICC worldcup in 2024?")]}
print_stream(agent.stream(inputs, stream_mode="values"))
```

```
================================ Human Message =================================

who won the ICC worldcup in 2024?
================================== Ai Message ==================================
Tool Calls:
  search_web (call_4m07HqUBoZuYUpBUjwkuUjSm)
 Call ID: call_4m07HqUBoZuYUpBUjwkuUjSm
  Args:
    query: 2024 ICC World Cup winner
================================ Tool Message =================================
Name: search_web

[{"url": "https://en.wikipedia.org/wiki/2024_ICC_Men's_T20_World_Cup_final", "content": "The 202
4 ICC Men's T20 World Cup final was a Twenty20 International cricket match played at Kensington
Oval in Bridgetown, Barbados on 29 June 2024 to determine the winner of the 2024 ICC Men's T20 W
orld Cup. [1] [2] It was played between South Africa and India.[3]India defeated South Africa by
7 runs to win their second T20 World Cup title. [4] Virat Kohli was named Player of the Match fo
r ..."}, {"url": "https://www.espncricinfo.com/series/icc-men-s-t20-world-cup-2024-1411166/india
-vs-south-africa-final-1415755/live-cricket-score", "content": "IND 176/7 vs SA 169/8 - India wo
n by 7 runs in Bridgetown, T20 World Cup 2024, June 29, 2024"}]
================================== Ai Message ==================================

India won the 2024 ICC Men's T20 World Cup, defeating South Africa by 7 runs in the final held o
n June 29, 2024, at Kensington Oval in Bridgetown, Barbados. Virat Kohli was named Player of the
Match.
```

Pre-built Agent using LangGraph

As we can see from the output, the LLM called the search_web tool for the given query, and tool found a URL and returned content back to the LLM which contains result to the query. Then, LLM returned the answer.

Build an Agent

Now we build an agent using langGraph:

Copy Code

```
# import the required methods

from langgraph.prebuilt import ToolNode
```

```
tool_node = ToolNode(tools)

# define functions to call the LLM or the tools

def call_model(state: MessagesState):
    messages = state["messages"]
    response = gpt_with_tools.invoke(messages)
    return {"messages": [response]}


def call_tools(state: MessagesState) -> Literal["tools", END]:
    messages = state["messages"]
    last_message = messages[-1]
    if last_message.tool_calls:
        return "tools"
    return END
```

The call_model function takes “messages” from state as input. The “messages” can include query, prompt, or content form the tools. It returns the response.

The call_tools function also takes state messages as input. If the last message contains tool calls, as we have seen in tool_calling output, then it returns “tools” node. Otherwise it ends.

Now let’s build nodes and edges:

Copy Code

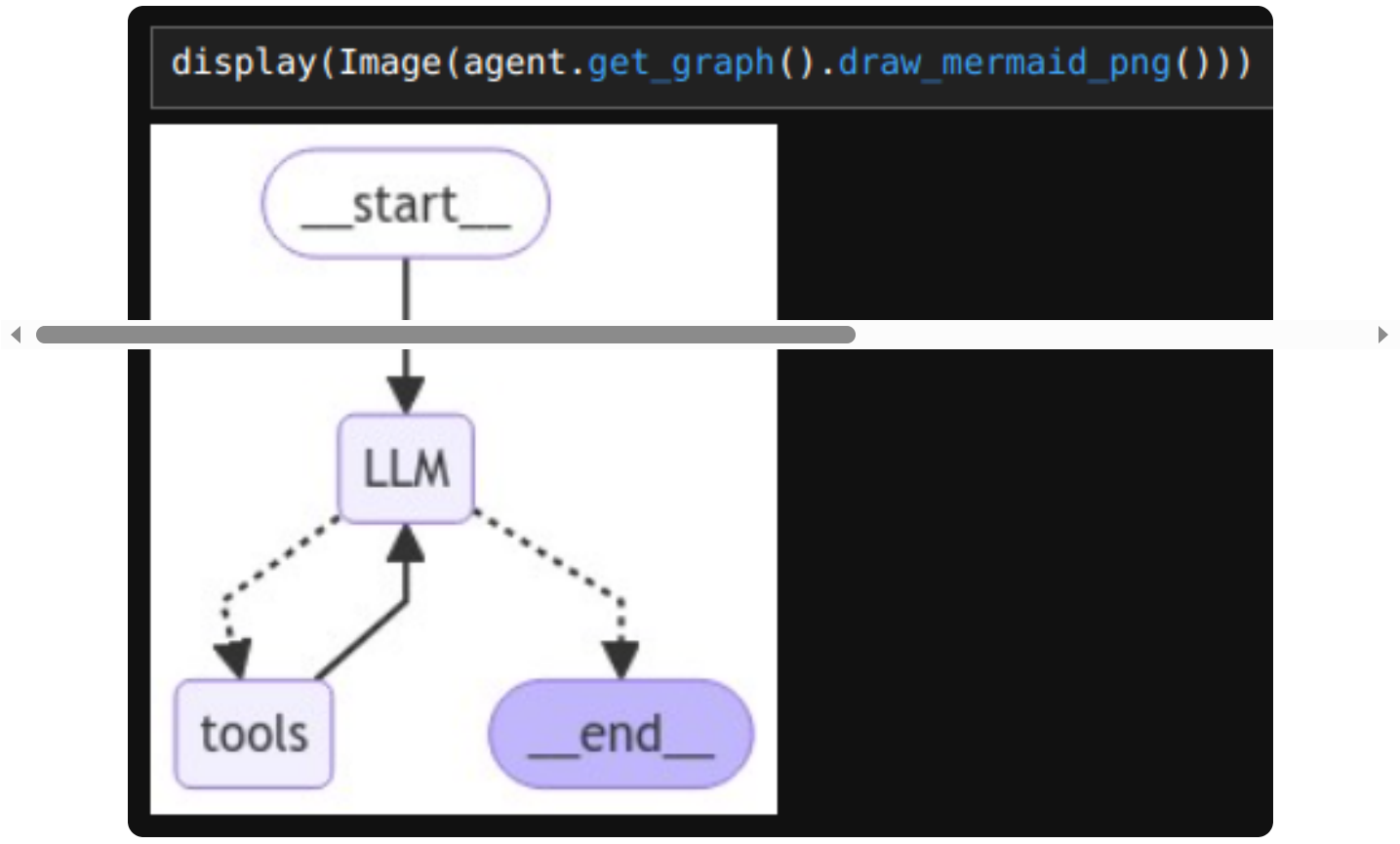
```
# initialize the workflow from StateGraph

workflow = StateGraph(MessagesState)

# add a node named ‘LLM’, with call_model function. This node uses an LLM to make

workflow.add_node("LLM", call_model)

# Our workflow starts with the ‘LLM’ node
```

```
# depending on the output of the LLM, it can go 'tools' node or end. So, we add a

workflow.add_conditional_edges("LLM", call_tools)

# 'tools' node sends the information back to the LLM

workflow.add_edge("tools", "LLM")
```

Now let's compile the workflow and display it.

Copy Code

```
agent = workflow.compile()

display(Image(agent.get_graph().draw_mermaid_png()))
```



As shown in the image, we start with the LLM. The LLM either calls the tools or

Now let's query the agent and see the result:

```
for chunk in agent.stream(
    {"messages": [("user", "Will it rain in Bengaluru today?")]},
    stream_mode="values",):
    chunk["messages"][-1].pretty_print()
```

Copy Code

Output:

```
for chunk in agent.stream(
    {"messages": [("user", "Will it rain in Bengaluru today?")]},
    stream_mode="values",
):
    chunk["messages"][-1].pretty_print()
================================ Human Message =================================

Will it rain in Bengaluru today?
================================== Ai Message ==================================
Tool Calls:
  get_weather (call_1UYlRf2fCY6oVrAnfKuuW6eu)
 Call ID: call_1UYlRf2fCY6oVrAnfKuuW6eu
  Args:
    query: Bengaluru
================================= Tool Message =================================
Name: get_weather

{"location": {"name": "Bengaluru", "region": "Karnataka", "country": "India", "lat": 12.98, "lon": 77.58, "tz_id": "Asia/Kolkata", "localtime_epoc
h": 1725432615, "localtime": "2024-09-04 12:20"}, "current": {"last_updated_epoch": 1725432300, "last_updated": "2024-09-04 12:15", "temp_c": 27.
2, "temp_f": 81.0, "is_day": 1, "condition": {"text": "Partly cloudy", "icon": "//cdn.weatherapi.com/weather/64x64/day/116.png", "code": 1003}, "w
ind_mph": 15.0, "wind_kph": 24.1, "wind_degree": 260, "wind_dir": "W", "pressure_mb": 1015.0, "pressure_in": 29.97, "precip_mm": 0.03, "precip_i
n": 0.0, "humidity": 74, "cloud": 75, "feelslike_c": 29.1, "feelslike_f": 84.4, "windchill_c": 25.0, "windchill_f": 77.0, "heatindex_c": 26.4, "he
atindex_f": 79.6, "dewpoint_c": 18.1, "dewpoint_f": 64.6, "vis_km": 8.0, "vis_miles": 4.0, "uv": 6.0, "gust_mph": 20.6, "gust_kph": 33.1}}
================================== Ai Message ==================================

Today in Bengaluru, the weather is partly cloudy with a temperature of 27.2°C. There is a very light amount of precipitation recorded (0.03 mm), b
ut it does not indicate significant rain. So, it is unlikely to rain today.
```

As we have asked about the weather, get_weather tool is called and it returned various weather related values. Based on those values, the LLM returned that it is unlikely to rain.

In this way, we can add different kinds of tools to the LLM so that we can get our queries answered even if LLM alone can't answer. Thus LLM agents will be far more useful in many scenarios

LangGraph offers a powerful toolset for building complex AI systems. It provides a framework for creating agentic systems that can reason, make decisions, and interact with multiple data sources. Key features include:

- **Modifiable Agent Runtimes:** With LangGraph, developers may create runtimes specifically suited to particular use cases and agent behaviors, overcoming the limitations of the conventional agent executor.
- **Support for Cyclic Execution:** When cyclic graphs are enabled, LangGraph makes applying sophisticated reasoning methods that require several LLM iterations easier.
- **Improved State Management:** Because LangGraph graphs are stateful, more intricate agent state tracking and updating can be done throughout the execution phase.
- **Multi-Agent Coordination:** Within a single graph structure, each agent can have its prompt, LLM, tools, and custom code. LangGraph is an expert in building and administering these kinds of systems.
- **Flexible Tool Integration:** LangGraph's node-based structure allows agents to easily incorporate various tools and functionalities into their repertoire.
- **Better Control Flow:** LangGraph's edge-based approach provides fine-grained control over the execution flow of an agent or multi-agent system.
- **Chat-Based Agent Support:** LangGraph introduces a chat agent executor, representing the agent state as a list of messages. This is particularly useful

LangGraph has many different real-world applications. It makes more complex decision-making processes possible in single-agent contexts by letting actors review and improve their arguments before acting. This is especially helpful in difficult problem-solving situations where linear execution might not be sufficient.

LangGraph excels in multi-agent systems. It permits the development of complicated agent ecosystems, wherein many specialized agents can work together to accomplish intricate tasks. LangGraph controls each agent's interactions and information sharing through its graph structure, which can be developed with specific capabilities.

For instance, a system with distinct agents for comprehending the initial query, retrieving knowledge, generating responses, and ensuring quality assurance may be developed in a customer service setting. LangGraph would oversee information flow management, enabling smooth and efficient consumer engagement among these workers.

The Future of AI Agents

Frameworks such as LangGraph are becoming increasingly important as AI develops. LangGraph is making the next generation of AI applications possible by offering a versatile and strong framework for developing and overseeing AI agents.

features, we may anticipate seeing more advanced AI agents that can do ever more complex jobs.

Conclusion

To sum up, LangGraph is a major advancement in the development of AI agents. It enables developers to push the limits of what's possible with AI agents by eliminating the shortcomings of earlier systems and offering a flexible, graph-based framework for agent construction and execution. LangGraph is positioned to influence the direction of artificial intelligence significantly in the future.

Unlock the potential of AI with LangGraph today! Start building your advanced AI agents and elevate your projects—explore our Generative AI Pinnacle Program now!

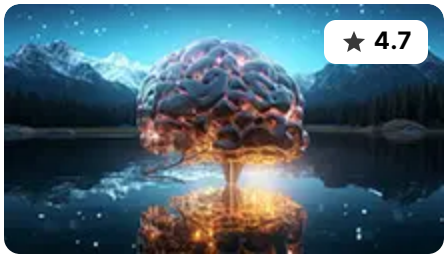
Also read: OpenAI's AI Agents to Automate Complex Tasks

Sahitya Arya

I'm Sahitya Arya, a seasoned Deep Learning Engineer with one year of hands-on experience in both Deep Learning and Machine Learning. Throughout my career, I've authored more than three research papers and have gained a profound understanding of Deep Learning techniques. Additionally, I possess expertise in Large Language Models (LLMs), contributing to my comprehensive skill set in cutting-edge technologies for artificial intelligence.

Advanced | Artificial Intelligence | Generative AI | Large Language Models | LLMs

Free Courses



Generative AI - A Way of Life

Explore Generative AI for beginners: create text and images, use top AI tools, learn practical skills, and ethics.

Getting Started with Large Language Models

Master Large Language Models (LLMs) with this course, offering clear guidance in NLP and model training made simple.

This free course guides you on building LLM apps, mastering prompt engineering, and developing chatbots with enterprise data.

Improving Real World RAG Systems: Key Challenges & Practical Solutions

Explore practical solutions, advanced retrieval strategies, and agentic RAG systems to improve context, relevance, and accuracy in AI-driven applications.

Microsoft Excel: Formulas & Functions

Master MS Excel for data analysis with key formulas, functions, and LookUp tools in this comprehensive course.

RECOMMENDED ARTICLES

What is LangChain?

Top 7 Frameworks for Building AI Agents in 2025

LangGraph Tutorial for Beginners

Understanding LangChain Agent Framework

Smolagents vs LangGraph: A Comprehensive Compar...

Building Smart AI Agents with LangChain

Agno Framework: A Lightweight Library for Build...

LangGraph vs CrewAI vs AutoGen to Build a Data ...

Responses From Readers

What are your thoughts?...

GenAI Pinnacle Program | GenAI Pinnacle Plus Program | AI/ML BlackBelt Program | Agentic AI Pioneer Program

Free Courses

Generative AI | DeepSeek | OpenAI Agent SDK | LLM Applications using Prompt Engineering | DeepSeek from Scratch | Stability.AI | SSM & MAMBA | RAG Systems using LlamaIndex | Building LLMs for Code | Python | Microsoft Excel | Machine Learning | Deep Learning | Mastering Multimodal RAG | Introduction to Transformer Model | Bagging & Boosting | Loan Prediction | Time Series Forecasting | Tableau | Business Analytics | Vibe Coding in Windsurf | Model Deployment using FastAPI | Building Data Analyst AI Agent | Getting started with OpenAI o3-mini | Introduction to Transformers and Attention Mechanisms

Popular Categories

AI Agents | Generative AI | Prompt Engineering | Generative AI Application | News | Technical Guides | AI Tools | Interview Preparation | Research Papers | Success Stories | Quiz | Use Cases | Listicles

Generative AI Tools and Techniques

GANs | VAEs | Transformers | StyleGAN | Pix2Pix | Autoencoders | GPT | BERT | Word2Vec | LSTM | Attention Mechanisms | Diffusion Models | LLMs | SLMs | Encoder Decoder Models | Prompt Engineering | LangChain | LlamaIndex | RAG | Fine-tuning | LangChain AI Agent | Multimodal Models | RNNs | DCGAN | ProGAN | Text-to-Image Models | DDPM | Document Question Answering | Imagen | T5 (Text-to-Text Transfer Transformer) | Seq2seq Models | WaveNet | Attention Is All You Need (Transformer Architecture) | WindSurf | Cursor

Popular GenAI Models

Llama 4 | Llama 3.1 | GPT 4.5 | GPT 4.1 | GPT 4o | o3-mini | Sora | DeepSeek-R1 |

AI Development Frameworks

n8n | LangChain | Agent SDK | A2A by Google | SmolAgents | LangGraph | CrewAI | Agno | LangFlow | AutoGen | LlamaIndex | Swarm | AutoGPT

Data Science Tools and Techniques

Python | R | SQL | Jupyter Notebooks | TensorFlow | Scikit-learn | PyTorch | Tableau | Apache Spark | Matplotlib | Seaborn | Pandas | Hadoop | Docker | Git | Keras | Apache Kafka | AWS | NLP | Random Forest | Computer Vision | Data Visualization | Data Exploration | Big Data | Common Machine Learning Algorithms | Machine Learning | Google Data Science Agent

Become a Mentor

Become an Instructor

Data Culture

AI Newsletter