## Overview[¶](https://langchain-ai.github.io/langgraph/#overview)

[LangGraph](https://langchain-ai.github.io/langgraph/) is a library for **building stateful, multi-actor applications with LLMs**, used to **create agent and multi-agent workflows.**   
Compared to other LLM frameworks, it offers these core benefits: **cycles, controllability, and persistence.**  
LangGraph allows you to define flows that involve cycles, essential for most agentic architectures, differentiating it from DAG-based solutions.   
As a very low-level framework, it provides fine-grained control over both the flow and state of your application, crucial for creating reliable agents. Additionally, LangGraph includes built-in persistence, enabling advanced human-in-the-loop and memory features.  
LangGraph是一个用于构建具有LLMs的**状态持久化、多参与者应用程序**的库，用于创建**代理和多方代理**工作流程。  
 与其他LLM框架相比，它提供了以下核心优势：**循环、可控性和持久性。** LangGraph允许您**定义涉及循环**的流程，这对于大多数代理架构来说都是必不可少的，这使得它与基于**DAG的解决方案**区别开来。 作为一个非常底层的框架，它为您应用程序的流程和状态提供了细粒度的控制，这对于创建可靠的代理至关重要。  
此外，LangGraph还包括**内置的持久性，支持高级的人工参与循环和记忆**功能。

LangGraph is inspired by [Pregel](https://research.google/pubs/pub37252/) and [Apache Beam](https://beam.apache.org/). The public interface draws inspiration from [NetworkX](https://networkx.org/documentation/latest/). LangGraph is built by LangChain Inc, the creators of LangChain, but can be used without LangChain.

[LangGraph Platform](https://langchain-ai.github.io/langgraph/concepts/langgraph_platform) is infrastructure for deploying LangGraph agents. It is a commercial solution for deploying agentic applications to production, built on the open-source LangGraph framework. The LangGraph Platform consists of several components that work together to support the development, deployment, debugging, and monitoring of LangGraph applications: [LangGraph Server](https://langchain-ai.github.io/langgraph/concepts/langgraph_server) (APIs), [LangGraph SDKs](https://langchain-ai.github.io/langgraph/concepts/sdk) (clients for the APIs), [LangGraph CLI](https://langchain-ai.github.io/langgraph/concepts/langgraph_cli) (command line tool for building the server), [LangGraph Studio](https://langchain-ai.github.io/langgraph/concepts/langgraph_studio) (UI/debugger),

To learn more about LangGraph, check out our first LangChain Academy course, Introduction to LangGraph, available for free [here](https://academy.langchain.com/courses/intro-to-langgraph).  
  
LangGraph受到Pregel和Apache Beam的启发。 公共接口借鉴了NetworkX的设计理念。 LangGraph由LangChain Inc.构建，该公司也是LangChain的创造者，但LangGraph可以不依赖LangChain使用。   
LangGraph平台是部署LangGraph代理的基础设施。它是一个商业解决方案，用于将代理应用程序部署到生产环境，基于开源的LangGraph框架构建。  
LangGraph平台由几个协同工作的组件组成，支持LangGraph应用程序的开发、部署、调试和监控：  
LangGraph服务器（APIs）、LangGraph SDKs（API的客户端）、LangGraph CLI（用于构建服务器的命令行工具）和LangGraph Studio（用户界面/调试器）。   
想要了解更多关于LangGraph的信息，请查看我们的第一个LangChain学院课程《LangGraph入门》，免费提供，点击此处访问。

### Key Features[¶](https://langchain-ai.github.io/langgraph/#key-features)

* **Cycles and Branching**: Implement loops and conditionals in your apps.
* **Persistence**: Automatically save state after each step in the graph. Pause and resume the graph execution at any point to support error recovery, human-in-the-loop workflows, time travel and more.
* **Human-in-the-Loop**: Interrupt graph execution to approve or edit next action planned by the agent.
* **Streaming Support**: Stream outputs as they are produced by each node (including token streaming).
* **Integration with LangChain**: LangGraph integrates seamlessly with [LangChain](https://github.com/langchain-ai/langchain/) and [LangSmith](https://docs.smith.langchain.com/) (but does not require them).

### LangGraph Platform[¶](https://langchain-ai.github.io/langgraph/#langgraph-platform)

LangGraph Platform is a commercial solution for deploying agentic applications to production, built on the open-source LangGraph framework. Here are some common issues that arise in complex deployments, which LangGraph Platform addresses:

* **Streaming support**: LangGraph Server provides [multiple streaming modes](https://langchain-ai.github.io/langgraph/concepts/streaming) optimized for various application needs
* **Background runs**: Runs agents asynchronously in the background
* **Support for long running agents**: Infrastructure that can handle long running processes
* [**Double texting**](https://langchain-ai.github.io/langgraph/concepts/double_texting): Handle the case where you get two messages from the user before the agent can respond
* **Handle burstiness**: Task queue for ensuring requests are handled consistently without loss, even under heavy loads

## Installation[¶](https://langchain-ai.github.io/langgraph/#installation)

pip install -U langgraph

## Example[¶](https://langchain-ai.github.io/langgraph/#example)

One of the central concepts of LangGraph is state. Each graph execution creates a state that is passed between nodes in the graph as they execute, and each node updates this internal state with its return value after it executes. The way that the graph updates its internal state is defined by either the type of graph chosen or a custom function.

Let's take a look at a simple example of an agent that can use a search tool.

pip install langchain-anthropic

export ANTHROPIC\_API\_KEY=sk-...

Optionally, we can set up [LangSmith](https://docs.smith.langchain.com/) for best-in-class observability.

export LANGSMITH\_TRACING=true

export LANGSMITH\_API\_KEY=lsv2\_sk\_...

from typing import Annotated, Literal, TypedDict

from langchain\_core.messages import HumanMessage

from langchain\_anthropic import ChatAnthropic

from langchain\_core.tools import tool

from langgraph.checkpoint.memory import MemorySaver

from langgraph.graph import END, START, StateGraph, MessagesState

from langgraph.prebuilt import ToolNode

# Define the tools for the agent to use

@tool

def search(query: str):

"""Call to surf the web."""

# This is a placeholder, but don't tell the LLM that...

if "sf" in query.lower() or "san francisco" in query.lower():

return "It's 60 degrees and foggy."

return "It's 90 degrees and sunny."

tools = [search]

tool\_node = ToolNode(tools)

model = ChatAnthropic(model="claude-3-5-sonnet-20240620", temperature=0).bind\_tools(tools)

# Define the function that determines whether to continue or not

def should\_continue(state: MessagesState) -> Literal["tools", END]:

messages = state['messages']

last\_message = messages[-1]

# If the LLM makes a tool call, then we route to the "tools" node

if last\_message.tool\_calls:

return "tools"

# Otherwise, we stop (reply to the user)

return END

# Define the function that calls the model

def call\_model(state: MessagesState):

messages = state['messages']

response = model.invoke(messages)

# We return a list, because this will get added to the existing list

return {"messages": [response]}

# Define a new graph

workflow = StateGraph(MessagesState)

# Define the two nodes we will cycle between

workflow.add\_node("agent", call\_model)

workflow.add\_node("tools", tool\_node)

# Set the entrypoint as `agent`

# This means that this node is the first one called

workflow.add\_edge(START, "agent")

# We now add a conditional edge

workflow.add\_conditional\_edges(

# First, we define the start node. We use `agent`.

# This means these are the edges taken after the `agent` node is called.

"agent",

# Next, we pass in the function that will determine which node is called next.

should\_continue,

)

# We now add a normal edge from `tools` to `agent`.

# This means that after `tools` is called, `agent` node is called next.

workflow.add\_edge("tools", 'agent')

# Initialize memory to persist state between graph runs

checkpointer = MemorySaver()

# Finally, we compile it!

# This compiles it into a LangChain Runnable,

# meaning you can use it as you would any other runnable.

# Note that we're (optionally) passing the memory when compiling the graph

app = workflow.compile(checkpointer=checkpointer)

# Use the Runnable

final\_state = app.invoke(

{"messages": [HumanMessage(content="what is the weather in sf")]},

config={"configurable": {"thread\_id": 42}}

)

final\_state["messages"][-1].content

"Based on the search results, I can tell you that the current weather in San Francisco is:\n\nTemperature: 60 degrees Fahrenheit\nConditions: Foggy\n\nSan Francisco is known for its microclimates and frequent fog, especially during the summer months. The temperature of 60°F (about 15.5°C) is quite typical for the city, which tends to have mild temperatures year-round. The fog, often referred to as "Karl the Fog" by locals, is a characteristic feature of San Francisco\'s weather, particularly in the mornings and evenings.\n\nIs there anything else you\'d like to know about the weather in San Francisco or any other location?"

Now when we pass the same "thread\_id", the conversation context is retained via the saved state (i.e. stored list of messages)

final\_state = app.invoke(

{"messages": [HumanMessage(content="what about ny")]},

config={"configurable": {"thread\_id": 42}}

)

final\_state["messages"][-1].content

"Based on the search results, I can tell you that the current weather in New York City is:\n\nTemperature: 90 degrees Fahrenheit (approximately 32.2 degrees Celsius)\nConditions: Sunny\n\nThis weather is quite different from what we just saw in San Francisco. New York is experiencing much warmer temperatures right now. Here are a few points to note:\n\n1. The temperature of 90°F is quite hot, typical of summer weather in New York City.\n2. The sunny conditions suggest clear skies, which is great for outdoor activities but also means it might feel even hotter due to direct sunlight.\n3. This kind of weather in New York often comes with high humidity, which can make it feel even warmer than the actual temperature suggests.\n\nIt's interesting to see the stark contrast between San Francisco's mild, foggy weather and New York's hot, sunny conditions. This difference illustrates how varied weather can be across different parts of the United States, even on the same day.\n\nIs there anything else you'd like to know about the weather in New York or any other location?"

### Step-by-step Breakdown[¶](https://langchain-ai.github.io/langgraph/#step-by-step-breakdown)

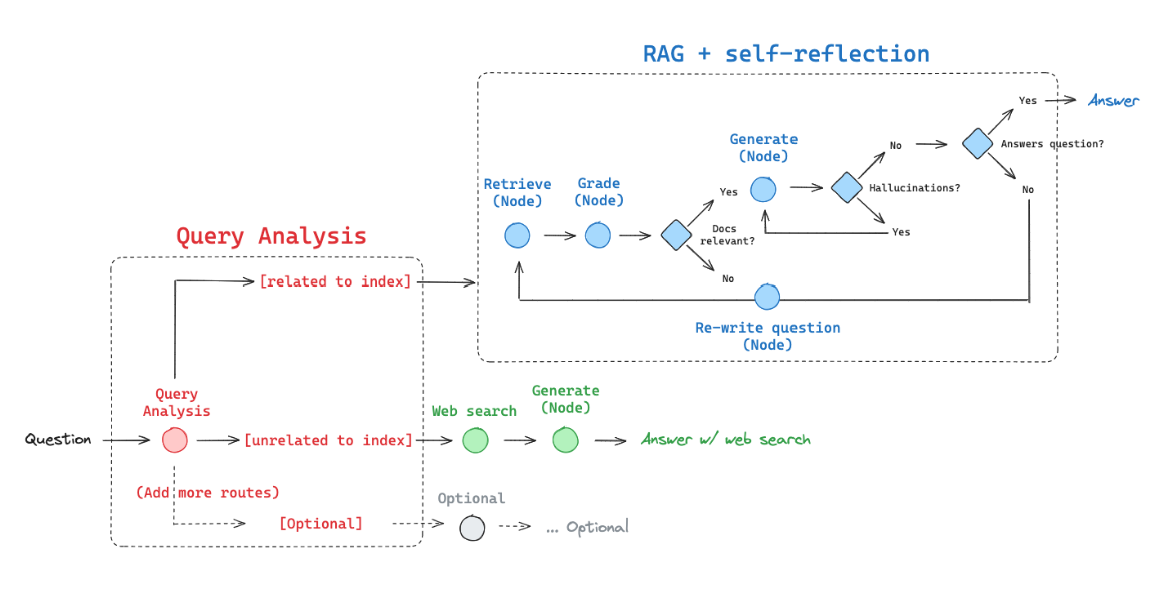
1. Initialize the model and tools.
2. Initialize graph with state.
3. Define graph nodes.
4. Define entry point and graph edges.
5. Compile the graph.
6. Execute the graph.

## Documentation[¶](https://langchain-ai.github.io/langgraph/#documentation)

* [Tutorials](https://langchain-ai.github.io/langgraph/tutorials/): Learn to build with LangGraph through guided examples.
* [How-to Guides](https://langchain-ai.github.io/langgraph/how-tos/): Accomplish specific things within LangGraph, from streaming, to adding memory & persistence, to common design patterns (branching, subgraphs, etc.), these are the place to go if you want to copy and run a specific code snippet.
* [Conceptual Guides](https://langchain-ai.github.io/langgraph/concepts/high_level/): In-depth explanations of the key concepts and principles behind LangGraph, such as nodes, edges, state and more.
* [API Reference](https://langchain-ai.github.io/langgraph/reference/graphs/): Review important classes and methods, simple examples of how to use the graph and checkpointing APIs, higher-level prebuilt components and more.
* [LangGraph Platform](https://langchain-ai.github.io/langgraph/concepts/#langgraph-platform): LangGraph Platform is a commercial solution for deploying agentic applications in production, built on the open-source LangGraph framework.

## Contributing[¶](https://langchain-ai.github.io/langgraph/#contributing)

For more information on how to contribute, see [here](https://github.com/langchain-ai/langgraph/blob/main/CONTRIBUTING.md).



# Q&A:

LangGraph 常见问题解答 原文： No. LangGraph is an orchestration framework for **complex agentic systems and is more low-level and controllable than LangChain agents.** On the other hand, LangChain provides a standard interface to interact with models and other components, useful for **straight-forward chains and retrieval flows.**

No. LangGraph 是一个复杂的代理系统编排框架，比 LangChain 代理更底层、更可控。另一方面，LangChain 提供了一个与模型和其他组件交互的标准接口，适用于直接的链式调用和数据检索流程。

Other agentic frameworks can work for simple, generic tasks but fall short for complex tasks bespoke to a company’s needs. LangGraph provides a more expressive framework to handle companies’ unique tasks without restricting users to a single black-box cognitive architecture.

其他代理框架可能适用于简单、通用的任务，但在处理特定公司需求的复杂任务时显得力不从心。  
LangGraph 提供了一个更具表现力的框架，可以处理公司独特的任务，而不会将用户限制在单一的黑盒认知架构中。

LangGraph will not add any overhead to your code and is specifically designed with streaming workflows in mind.

LangGraph 不会给您的代码增加任何负担，并且是**专门为流式工作流程设计的。**

Yes. LangGraph is an MIT-licensed open-source library and is free to use.

是的。LangGraph 是一个遵循 MIT 许可的开源库，可以免费使用。

No. LangGraph Cloud is proprietary software that will eventually be a paid service for certain tiers of usage. We will always give ample notice before charging for a service and reward our early adopters with preferential pricing.

不。LangGraph Cloud 是专有软件，最终会对某些使用层级的服务收费。我们在开始收费服务之前会提前充分通知，并给予我们的早期采用者优先定价优惠。

关键特性 **循环和分支：在应用中实现循环和条件逻辑。**  
 **持久性：**在图的每一步之后自动保存状态。在任何点暂停和恢复图的执行，以支持错误恢复、人机交互循环工作流、时间旅行等功能。   
**人机交互循环：**中断图的执行，以批准或编辑代理计划的下一个行动。   
**流式处理支持：**流式传输每个节点生成的输出（包括令牌流）。   
与LangChain集成：LangGraph与LangChain和LangSmith（但不需要它们）无缝集成。

Adaptive RAG是一种RAG（Retrieval-Augmented Generation）策略，它结合了查询分析和主动/自我纠正的RAG。在论文中，作者报告了通过查询分析来决定使用以下哪种RAG：

1. **无检索（No Retrieval）**：对于最简单的查询，不进行检索直接回答。
2. **单次RAG（Single-shot RAG）**：对于简单查询，只需要一轮检索就可以让LLM完成回答。
3. **迭代RAG（Iterative RAG）**：对于复杂的需要多跳推理的查询，则需要多轮检索才能让LLM完成回答。

基于此，我们可以使用LangGraph来构建一个代理，该代理可以：

1. **网络搜索（Web search）**：用于与最近事件相关的问题。
2. **自我纠正RAG（Self-corrective RAG）**：用于与我们索引相关的问题。

