

Hands-on LangGraph

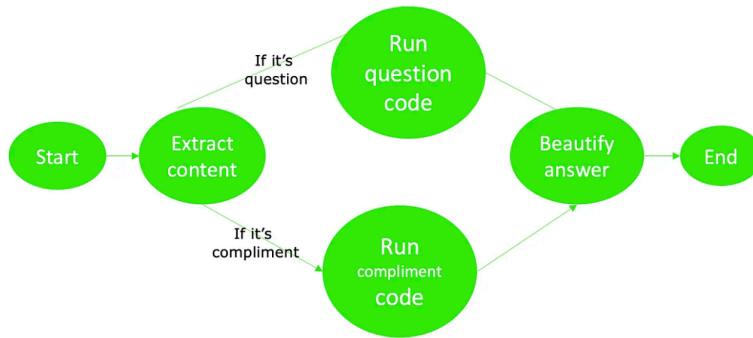
What is LangGraph?

This makes LangGraph especially useful for building: chatbots, multi-step reasoning pipelines, tool-using agents, multi-agent workflows, and robust control flows.

- Entry/Exit: start node (entry point) and end condition (finish point).
- Edges: transitions between nodes (linear or conditional).
- Nodes: Python functions that read/update the state (e.g., classify intent, call an LLM, format output).
- State: a shared dictionary/object that carries information across nodes (e.g., user question, classification label, final answer).

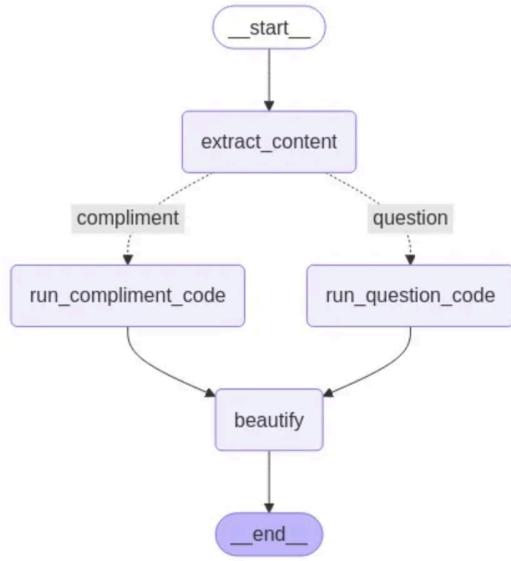
In LangGraph, your application typically consists of:

LangGraph is a graph-based framework for building LLM applications where your workflow is represented as a set of nodes (functions) and edges (transitions). Instead of writing a single linear chain, you explicitly define how data moves through the system, including loops, branching, and conditional routing.



What you will build

- A tiny graph that reads a social-media comment payload, extracts the customer remark, routes it as either a question or compliment, and then beautifies the response.
- You will run it end-to-end and optionally visualize the graph structure (Mermaid diagram).



Prerequisites

- Python 3.10+ (recommended 3.11).
- A terminal (macOS Terminal, Windows PowerShell, or Linux shell).
- Optional for graph image rendering: Jupyter / IPython, plus system tools for image display.

Step 1 – Create a project folder and virtual environment

Create a new folder and a virtual environment so installs do not affect your global Python.

macOS / Linux

```

mkdir LangGraph_Demo
cd LangGraph_Demo
python3 -m venv .venv
source .venv/bin/activate
python -m pip install --upgrade pip
  
```

Windows (PowerShell)

```

mkdir LangGraph_Demo
cd LangGraph_Demo
python -m venv .venv
.\.venv\Scripts\Activate.ps1
python -m pip install --upgrade pip
  
```

Step 2 – Install LangGraph and dependencies

Install LangGraph and LangChain (used only to talk to an LLM backend).

```
pip install -U langgraph langchain
```

If you plan to use Ollama (local LLM), install the Ollama integration:

```
pip install -U langchain-ollama
```

Step 3 – Choose and install an LLM backend

Local LLM via Ollama

Ollama runs open-source LLMs locally. This is great for hands-on labs because it avoids API keys and works offline once the model is downloaded.

Install Ollama:

- macOS: download the .dmg from the official Ollama download page and drag the app into Applications.
- Linux: use the install script from the official download page.
- Windows: download the installer from the official download page.

Verify and pull a small model (choose one that fits your machine). Example:

```
ollama --help
ollama pull llama3.2:1b
```

Quick test:

```
ollama run llama3.2:1b "Say hello in one sentence."
```

Step 4 – Create the LangGraph program

Create a file named langgraph_demo.py with the code below.

```
nano langgraph_demo.py
```

Below is the code and tutorial flow (extract -> route -> answer -> beautify).

```
from typing_extensions import TypedDict

from langgraph.graph import StateGraph, START, END

# Optional (only if you use Ollama)
from langchain_ollama import ChatOllama

class State(TypedDict):
```

```
text: str
answer: str
payload: list[dict]

# --- Node 1: extract text from payload ---
def extract_content(state: State):
    return {"text": state["payload"][0]["customer_remark"]}

# --- Router: decide which node to go to next ---
def route_question_or_compliment(state: State):
    # simple heuristic: '?' => question, otherwise compliment
    return "question" if "?" in state["text"] else "compliment"

# --- Node 2a: handle compliment ---
def run_compliment_code(state: State):
    return {"answer": "Thanks for the compliment."}

# --- Node 2b: handle question ---
def run_question_code(state: State):
    return {"answer": "Thanks for your question. We will look into it."}

# --- Node 3: beautify (optionally using an LLM) ---
def beautify(state: State):
    # If you want: uncomment Ollama call to rewrite the answer nicely.
    # llm = ChatOllama(model="llama3.2:1b", temperature=0)
    # prompt = f"Rewrite this politely in one sentence: {state['answer']}"
    # pretty = llm.invoke(prompt).content
    # return {"answer": pretty}

    return {"answer": state["answer"] + " (beautified)"}
```

```

def build_graph():

    graph_builder = StateGraph(State)

    graph_builder.add_node("extract_content", extract_content)
    graph_builder.add_node("run_question_code", run_question_code)
    graph_builder.add_node("run_compliment_code", run_compliment_code)
    graph_builder.add_node("beautify", beautify)

    graph_builder.add_edge(START, "extract_content")

    graph_builder.add_conditional_edges(
        "extract_content",
        route_question_or_compliment,
        {
            "compliment": "run_compliment_code",
            "question": "run_question_code",
        },
    )

    graph_builder.add_edge("run_question_code", "beautify")
    graph_builder.add_edge("run_compliment_code", "beautify")
    graph_builder.add_edge("beautify", END)

    return graph_builder.compile()

if __name__ == "__main__":
    graph = build_graph()
    # --- Save diagram as PNG (optional) ---
    png_bytes = graph.get_graph().draw_mermaid_png()
    with open("langgraph_diagram.png", "wb") as f:
        f.write(png_bytes)
    print("Diagram saved as: langgraph_diagram.png")

```

```

example_payload = [
    {
        "time_of_comment": "2025-01-20",
        "customer_remark": "Why has the packaging changed?",
        "social_media_channel": "facebook",
        "number_of_likes": 100,
    }
]

result = graph.invoke({"payload": example_payload})
print(" FINAL RESULT: ", result)
print(" STEP-BY-STEP STREAM: ")
for step in graph.stream({"payload": example_payload}):
    print(step)

```

Step 5 – Run it

From the same folder (with your virtual environment activated):

```
python3 langgraph_demo.py
```

You should see a final state dictionary (including text, answer, and the original payload), plus step-by-step node outputs from stream().

Optional – Visualize the graph (Mermaid diagram)

If you are in a Jupyter notebook:

```
pip install jupyter jupyterlab notebook jupyter_core ipykernel ( to install jupyter notebook)
```

```
python3 -m notebook ( open's juypter notebook on browser)
```

you can render a diagram:

```
from IPython.display import Image, display
```

```

graph = build_graph()

display(Image(graph.get_graph().draw_mermaid_png()))

```

Troubleshooting

- ImportError for ChatOllama: ensure you installed langchain-ollama.
- Ollama command not found: confirm Ollama is installed and the CLI is on your PATH; restart the terminal after installation.
- Model not found in Ollama: run "ollama pull <model>" first.
- Python version issues: LangGraph docs recommend Python 3.10+.

References

- 1) Dr. Varshita Sher. Gentle Introduction to LangGraph: A Step-by-Step Tutorial. Level Up Coding (Medium), Mar 20, 2025. <https://levelup.gitconnected.com/gentle-introduction-to-langgraph-a-step-by-step-tutorial-2b314c967d3c>
- 2) LangChain Docs (OSS). Install LangGraph.
<https://docs.langchain.com/oss/python/langgraph/install>
- 3) Ollama Docs. Download / Install. <https://ollama.com/download>
- 4) LangChain Docs (OSS). ChatOllama integration.
<https://docs.langchain.com/oss/python/integrations/chat/ollama>