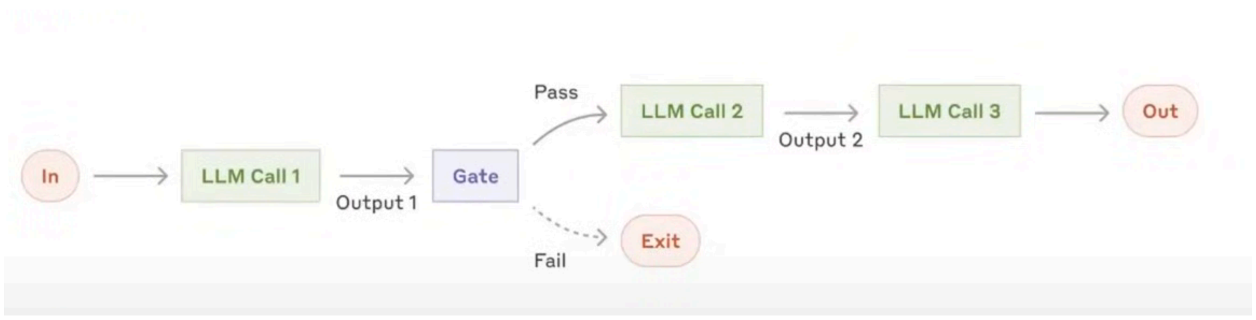
LangGraph Core Concepts

LLM Workflow

- Workflow that uses LLM in the execution stage

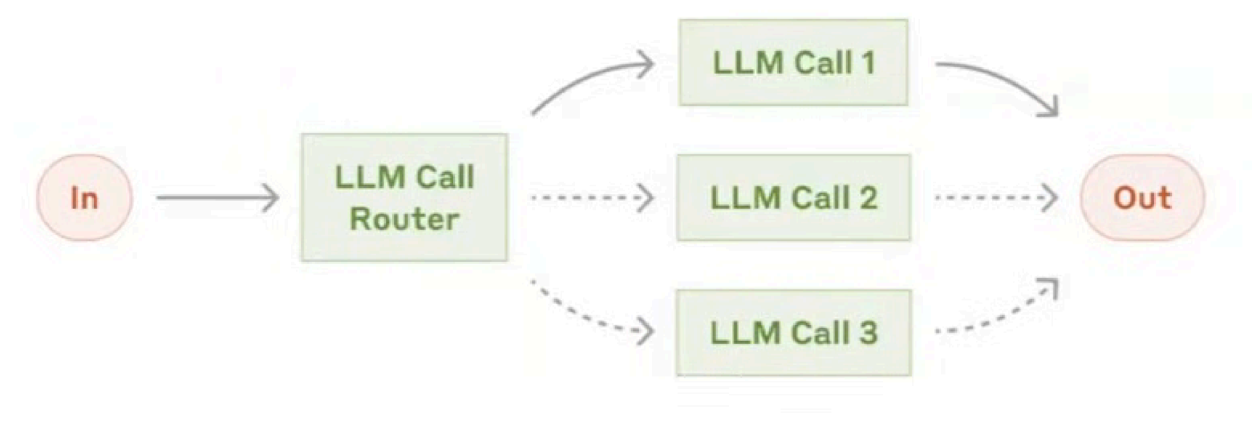
Common Workflows:

Prompt Chaining:



- In this wf, you call LLM multiple times

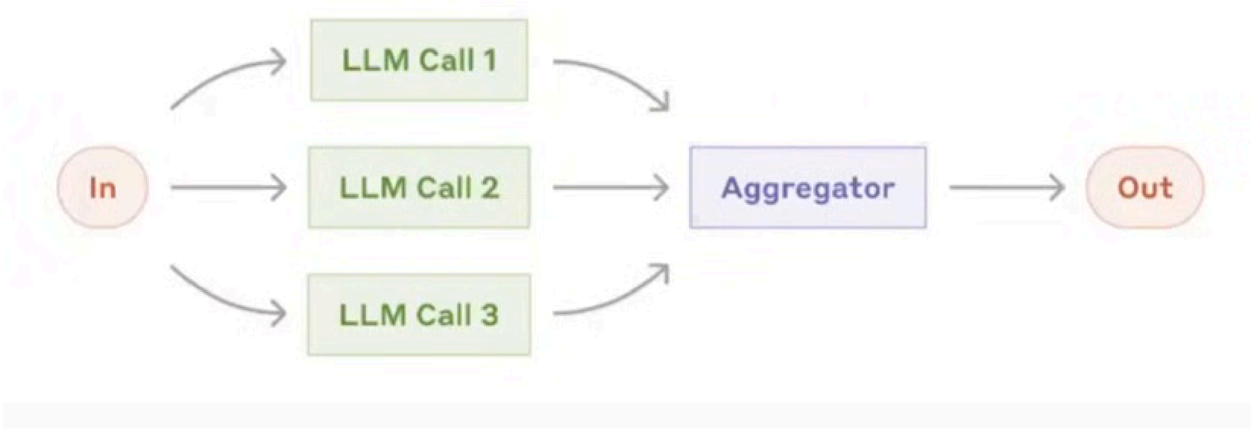
Routing:



- LLM gets a **Query**

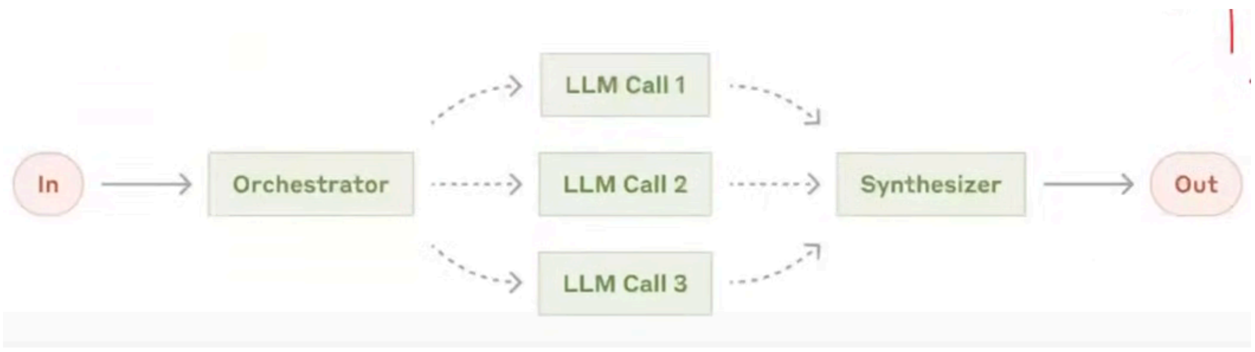
- It classifies the query (Refund, Sales, General, etc.)
- It routes the query to respective LLM

Parallelization:



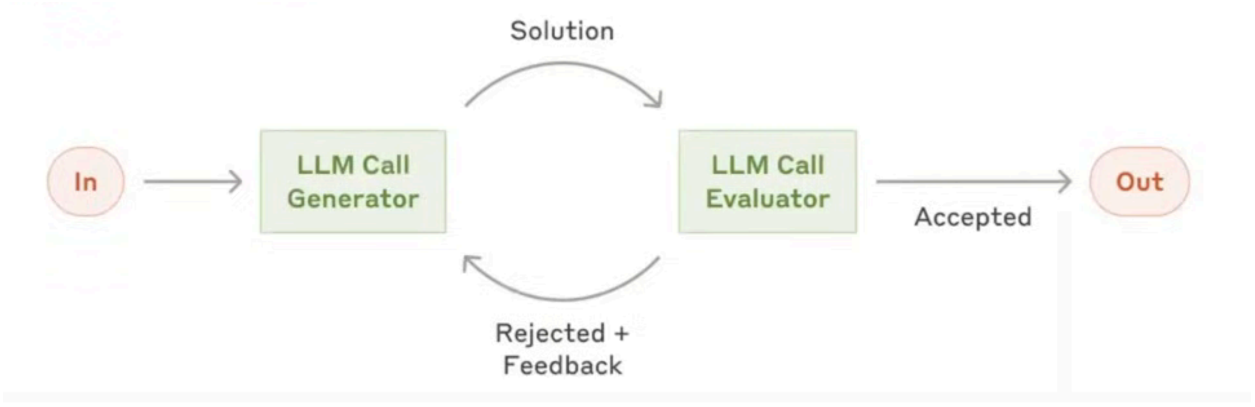
- Split a task into multiple subtasks
- **Execute all the subtasks parallelly**
- Combine the results

Orchestrator Worker:



- Similar to Parallelization
- Only difference is you do not know the nature of the subtasks
    - It is not pre-decided what LLM will do which task

Evaluator Optimizer:



- LLM is given a task which can't be successfully executed once
- Eg. generating email, blog, etc.

Steps:

1. Generator LLM generates the **response → Send this to evaluator**
2. We provide Evaluator LLM with concrete evaluation criteria
3. Evaluator accepts/rejects the solution based on the criteria
4. Gives a **feedback** if rejected
5. Generator will again generate a new response based on the feedback
6. 🔁This loop continues until evaluator is satisfied

!!Graphs, Nodes, Edges

Nodes:

- Every node is a python function for a particular task

Graph:

- A set of python functions connected through **edges**

State

- Every workflow needs some data for execution
- **State** is a **data structure** that holds information about the current execution context of a workflow or graph.
- **State** is typically represented by a dictionary or a `TypedDict` in Python.
- You add all the points in key, value pairs

```
class AgentState(TypedDict):
    name: str
    age: int
    skills: List[str]

# State used by the nodes
state = {
    "name": "John",
    "age": 30,
    "skills": ["python", "ML"]
}
```

- State is received by each node
- All node returns state

Reducers

A **reducer** is a **special function** that tells LangGraph *how to combine multiple updates* to the same state key when:

- Several nodes run **in parallel**
- Or the same node runs multiple times (loops)

Without reducers, LangGraph wouldn't know whether to **replace**, **append**, **merge**, etc.

Why It’s Needed?

Imagine you have a `messages` key in your state where every node adds a new message.

If Node A and Node B both update `messages` at the same time:

- Without a reducer → One might overwrite the other.
- With a reducer → Both get combined in the way you define.

> **It tells you if the response will be replaced, added or merged**

💡 **For example, let's say your state tracks a list of messages. Every time a node generates a new message, you don't want it to replace the old list; you want it to be added to the end. A reducer can handle this by defining a simple function that appends the new message to the existing list.**