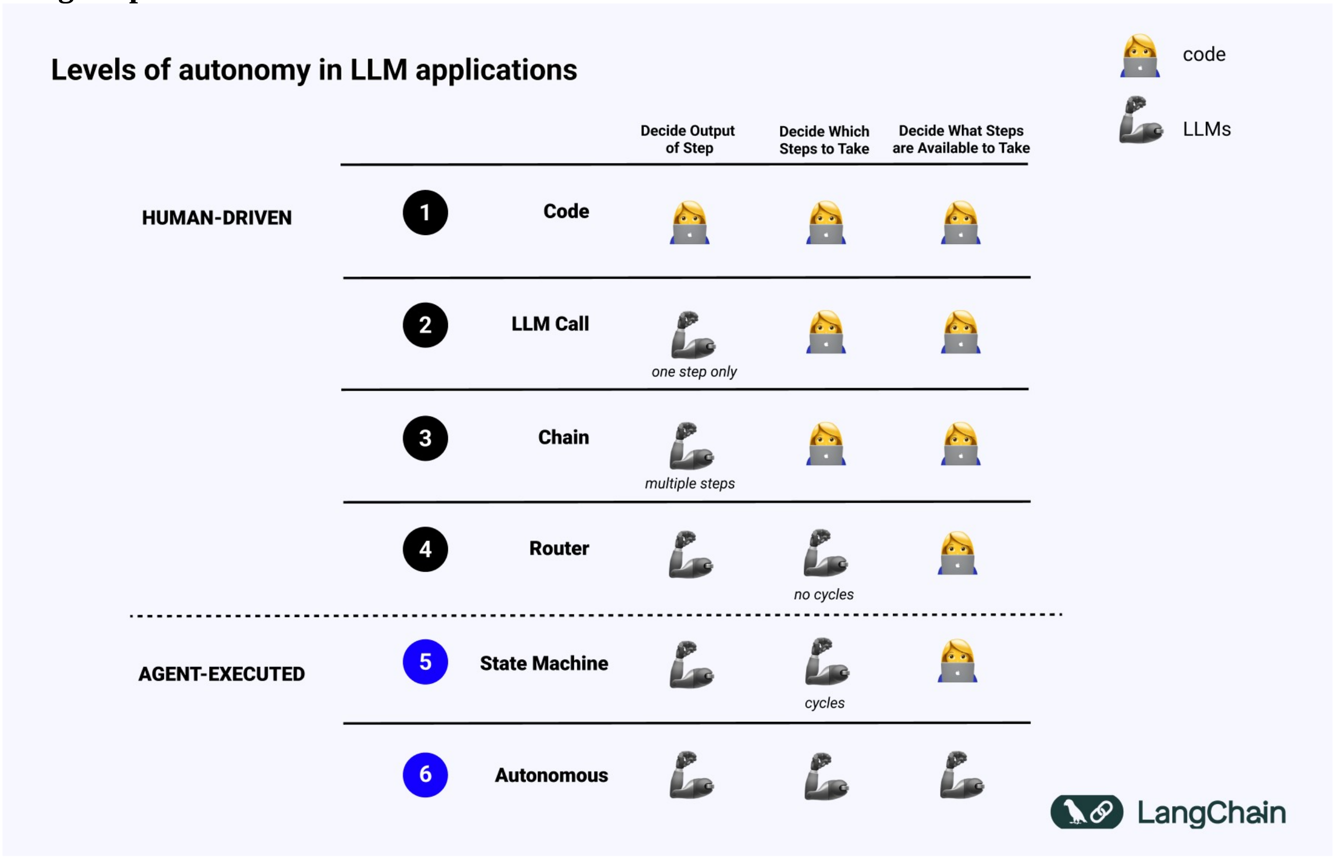
LangGraph1 Intro Notes:



Levels of autonomy in LLM applications

👩‍💻	code
🦾	LLMs

			Decide Output of Step	Decide Which Steps to Take	Decide What Steps are Available to Take
HUMAN-DRIVEN	1	Code	👩‍💻	👩‍💻	👩‍💻
	2	LLM Call	🦾 *one step only*	👩‍💻	👩‍💻
	3	Chain	🦾 *multiple steps*	👩‍💻	👩‍💻
	4	Router	🦾	🦾 *no cycles*	👩‍💻
AGENT-EXECUTED	5	State Machine	🦾	🦾 *cycles*	👩‍💻
	6	Autonomous	🦾	🦾	🦾

LangChain

LangChain explains the importance of LangGraph. In old days, in the Phase1 Code era, everything was coded (Decide output of step, Decide which steps to take, Decide what steps are available to take). In Phase1, we write the functions, manually call the functions. Manually we call functions from functions. Then, LLM came into picture and gave us some hope and that hope is the LLM calls. In Phase2, if we need some information, we were requesting these information from LLM and LLM will give you some response, which you are going to use. Decide output of the step: this output will be given by LLM then we manually decide which steps to take (what to do with the output) and decide what steps are available to take (which function to call and what sequence). Phase3 is bit complex, instead of one single call, we want to perform multiple calls. When you have multiple calls, you want to chain them. You have to perform multiple calls to the LLM and you can chain them. Say we have a prompt to a model and we get the output and that output becomes chain for the next model. Phase4 is Router. We can have different user queries right? One query / prompt regarding product, one question about delivery, one regarding payment. Say we have different functions handling different questions: one for orders, delivery and payment. When someone chats with that chatbot and says I have an issue with the payment, we need something (LLM) in between, who can determine which particular function to call for the user query. Who decides which steps to take? It's LLM again. As a programmer, as opposed to Phase2, we don't determine which function to call. We are calling it as a Router, since it is routing our request to the appropriate function. Phase5 is skipped. Phase6, in future, everything will be autonomous. If AI can decide output of step, decide which steps to take, and decide what steps are available to take, then we are not writing any code. Infinite amount of software services are created on the spot by AI. Problem with Phase6, is if AI determined what sequences of steps are available or decided what steps are available to take, then it can hallucinate. So the sequence is something humans should manage (Phase5). Say in an e-Commerce application, we have automated fully, if payment is successful, we will start doing packing and delivery. What if something went wrong, we need cycles to go back to fix the previous step. AI could hallucinate and determine the wrong sequence also.

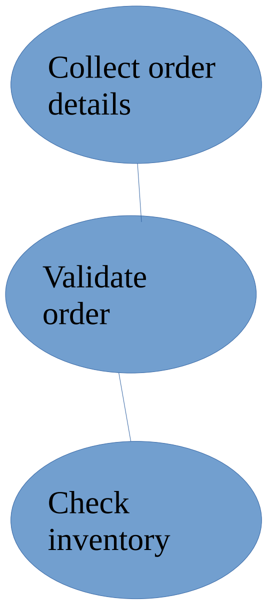
<u>What's the difference between a workflow and Agentic AI?</u>
Workflows are systems where LLMs and tools are orchestrated through predefined code paths. Yes there will be Agents and LLM calls but the entire flow will be predefined. If you have multiple agents, which agent will call which agent that will be predefined.
Agents are systems where LLMs dynamically direct their own processes and tool usage, maintaining control over how they accomplish tasks.
Workflows: predefined code paths, Agents have their own paths.

We need to orchestrate the steps, thats where LangGraph will help. First we start => Collect order details => Validate order => Check inventory? => Yes Payment process No Out of stock

We see a flow here. The moment you ask AI to manage a complex flow like this, it will hallucinate and start messing up with you. If you write a code and if you mess up, you can debug and fix. If AI hallucinate and messes up, who will fix it?

<u>Glue code:</u> Say when we see a Check Box like 'Check inventory?', the flow is not explicit and it could in either directions right? Here AI could mess up. How do we write in LangChain? If you want to achieve branching we have to write some extra code. Extra function with maybe an If-else or write a loop. That code is called a 'glue-code'. Extra code that's not part of LangChain and you are writing to make it work. The problem is, more code you write, 1. it can lead to bugs, 2. it is difficult to maintain, LangGraph helps with the above-mentioned 2 problems

<u>Challenge1 (with Langchain) is Control flow complexity</u> (If-else, conditional stuff) => while LangGraph helps you with conditional edges. In a graph, we have nodes/vertices and edges. Entire flow we can break into two parts: 1. Node, 2. Edge. Start => Collect order details => Validate order etc are called as Nodes.



Check Inventory is a conditional node: and edges are called as Conditional edges

You create a graph object and add the nodes

```
graph.add_node("CollectOrder", collect_order)
graph.add_edge("CollectOrder", "ValidateOrder")

graph.add_conditional_edges(
"ProcessPayment",
payment_router,
        {
        "success": "ShipOrder"
        "failed": "ProcessPayment"
        }
)
```

If this particular node "ProcessPayment" gives you success, you can call the next node: "ShipOrder"

Challenge2: Handling state

State means current status of the data. When you are looking into a scoreboard, every time you refresh you get a new state for that particular page. What you are updating is a state, state means current value.

What state or values we got to continuously maintain?
Order details, Payment status, Inventory, Wait list (if out of stock), Tracking info
Payment status is checked at multiple stages in the flow. If you want to maintain the state, somewhere you have to make it stateful
LangGraph you are achieving statefulness.
State management is, basically we are maintaining the data for the entire flow.
Where are you going to store the data? TypedDict is used to store this data

Challenge3: Event-driven execution

When your orders are out of stock, you will add that in the Wait list. What if the new order arrives, do we have to add new orders to the wait list manually or it should happen automatically? If it is automatic, how exactly we know we have to trigger the event? LangGraph provides the option of Event-driven. If something happens, do this. Thats possible with the help of LangGraph, Event-driven execution.

Challenge4: Fault-tolerance:

When you work with multiple agents, and this agent involves API calling, database connections, using external services. And if something breaks there, what your agents will do? They will panic. In Langchain everything works in sequence, if something breaks, it interrupts the entire flow. By default, Langchain is not fault tolerant. Langchain cannot handle such exceptions. LangGraph does provide fault-tolerance or fault-handling. Say you have 10 steps and in the 5th step some error occurred, in this case, even if you rollback, we should start from the 5th step not 1st step. LangGraph provides you with checkpoints and persistence to start from the correct step.

Challenge5: Human in the loop
Say we are doing something complex like handling sensitive data or involves money transactions, then you got to make sure you are keeping Human in the loop. With Langchain, whenever we build a chan and execute, it will not stop until the entire chain is finished. We don't want that, what we want is, if we have a chain of 10 steps, in 4th step where I am not sure as an Agent should we continue or talk with a human. How should we pause execution and get human permission? Maybe, it is about deducting money in your account or creating a file in C:\drive. Thats something we can do with the help of LangGraph. It has in-built Human in the loop workflow. It is designed in such a way that it can have interrupt points like breakpoints where user input is required.

Challenge6: Nested workflows
Complex applications often require workflows within workflows. We will talk when we talk about Subgraphs

Challenge7: Observability
In Langchain there is something called as LangSmith.
When you build applications, we want to know what's happening with the application, thats observability
https://www.langchain.com/langsmith/observability
LangSmith can manage the Langchain code but it cannot manage the user-written code
LangGraph uses Langsmith properly so you will get all the details on the dashboard

Those are the reasons why we should be using LangGraph.

Basically, Phase5 (State machine) is the LangGraph, it is not fully autonomous