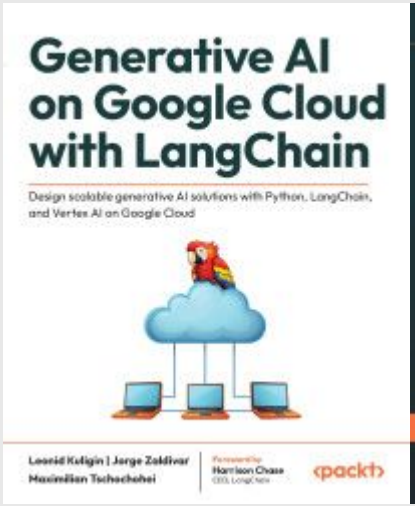
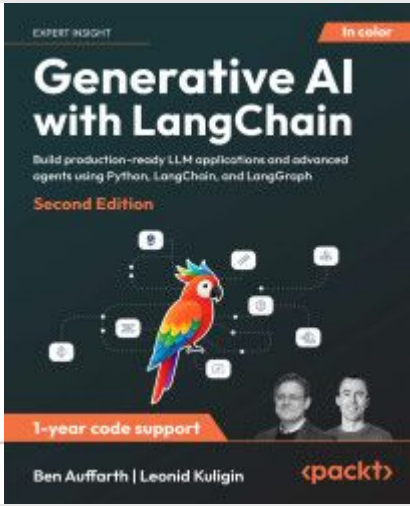
Speaker

- Leonid Kuligin
- >20 years in tech
- > 10 years working on production ML applications





Our agenda

1:00-1:30	LangGraph as a framework. Integration with tools.	Intro to LangChain and LangGraph. Building ReACT agent from scratch.
1:30-2:00	Orchestration	What is an agent? Single-agent systems. ReACT pattern.
2:00-2:15	break	
2:15-2:45	Controlled generation	How to use tool calling for controlled generation?
2:45-3:00	break	
3:00-3:30	plan-and-solve agent	going through detailed demo
3:30-4:00	Q&A	

Agenda

01 LangChain and LangGraph

02 Single-agent systems (ReACT)

03 Controlled generation

04 Questions

05

06

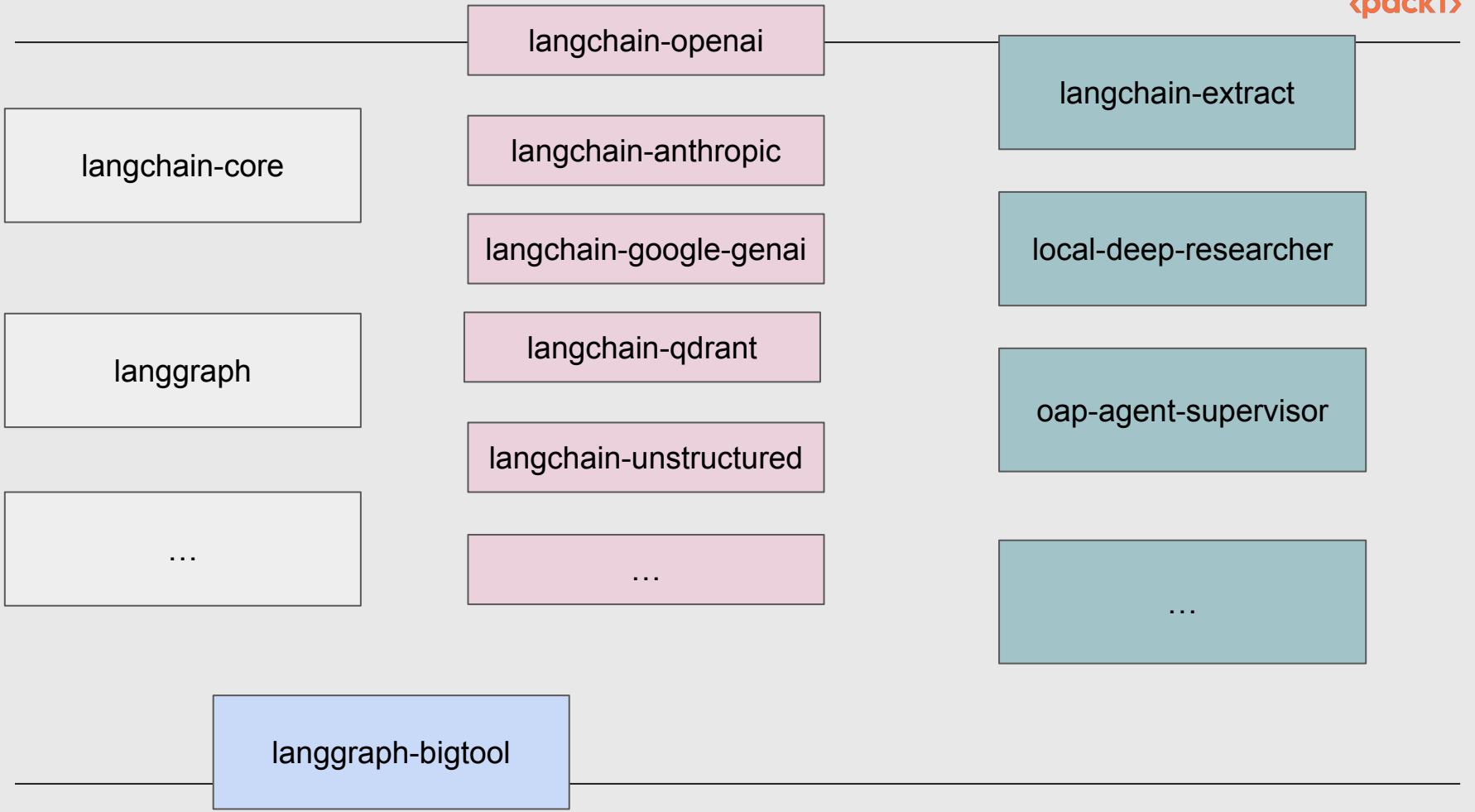
Today, our focus is on single-agent systems.
Tomorrow, we'll look into multi-agentic ones.

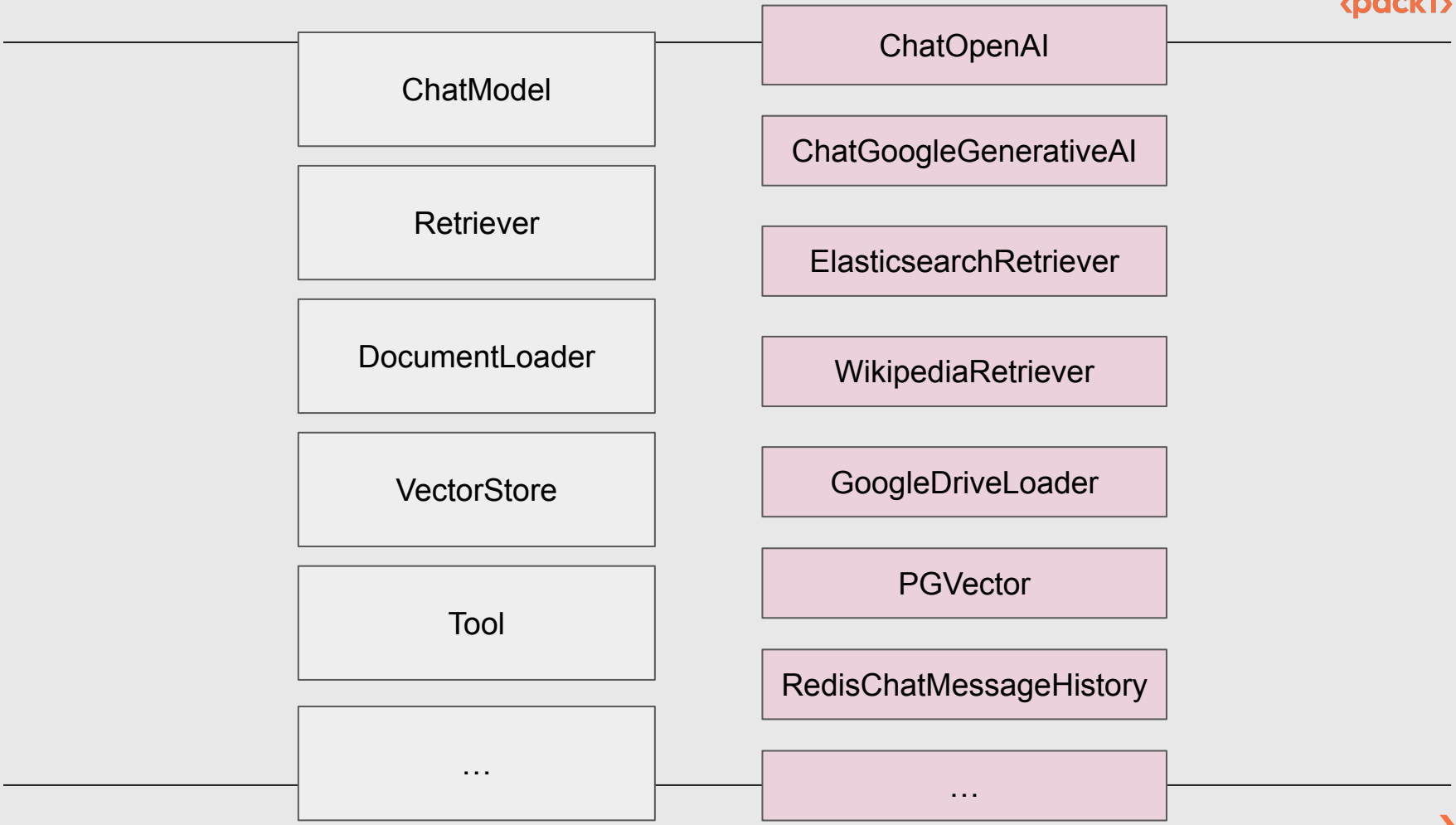
01

LangChain and LangGraph

LangChain

- Plenty of integrations: All vendors follow general interfaces defined by **langchain-core**
- Lots of useful built-in blocks for application developers: Defining tools, running retrievals, trimming message history, …


langchain-openai
langchain-extract
langchain-core
langchain-anthropic
langchain-google-genai
local-deep-researcher
langgraph
langchain-qdrant
oap-agent-supervisor
langchain-unstructured
…
…
…
langgraph-bigtool


ChatModel
Retriever
DocumentLoader
VectorStore
Tool
…
ChatOpenAI
ChatGoogleGenerativeAI
ElasticsearchRetriever
WikipediaRetriever
GoogleDriveLoader
PGVector
RedisChatMessageHistory
…

To recap: LangChain interfaces

Runnable (and RunnableSerializable)

- https://python.langchain.com/api_reference/core/language_models/langchain_core.language_models.chat_models.BaseChatModel.html
- https://python.langchain.com/api_reference/core/language_models/langchain_core.language_models.base.BaseLanguageModel.html
- https://python.langchain.com/api_reference/core/runnables/langchain_core.runnables.base.Runnable.html

To recap: LangChain interfaces

- Runnable (and RunnableSerializable)
- PromptTemplates
- ChatModel
  - BaseMessage
  - callbacks
- bind and bind_tools

LangChain Expression Language (LCEL)

- You don’t need to deal with Runnables
- Everything would be auto-converted (on the best effort basis)

LCEL magic

- | is a bitwise comparison operator in Python
- A Runnable overrides a hidden method __or__
  - https://python.langchain.com/api_reference/_modules/langchain_core/runnables/base.html#Runnable
- Python evaluation order: “while evaluating an assignment, the right-hand side is evaluated before the left-hand side”

LCEL magic

- a | b is equivalent to b.__or__(a)
- Everything is converted to RunnableSequence under the hood

How does LangChain handle tools

- LLM output is an AIMessage: It has a *tool_calls* component, which is a list of calls (with the *name* of the tools and *args* representing the payload)
- Create a ToolMessage that contains an output of the tool: *tool_call_id* is part of the metadata used to link ToolMessage to *tool_calls*

Why do we need a framework?

- Many building blocks and good abstractions
  - Short-term memory
  - State management
  - Reliability
  - Streaming
  - HITL
  - …
- Observability
- Modularity + configuration management

LangGraph

- We build a *graph*, or a workflow
- … by connecting *nodes* with *edges*
- *…*and maintains a *shared* state
- Like any orchestrator, it optimizes the execution by running nodes in parallel when possible
  - Key distinctions: it’s not a DAG, cycles are allowed and welcomed!

Node

- A callable that takes a *state* and returns updates to this *state*
- State is a map of key-value pairs. Two types of updates:
  - Replace the value
  - Apply the *reduce* operation: updated value = reduce(old value, new value)
- Order in which updates are applied to the general state is guaranteed
- A state can’t be updated within the node

Edge

- Direct or *conditional* edges
- Conditional edges allow branching by running a callable on the current state - an edge can’t update the state
- Complex cases: A node can schedule another node execution by applying hand-off or map commands

State updates

- Replace the value
- Use a built-in or custom *reducer* to combine (reduce) the previous value and the new one

Configuration

You can pass a config that also has a custom *configurable* dict as a second (optional) argument to the node

DEMO time

02

Single-agent Systems: ReACT

To do any task on a computer, you have to tell your device which app to use. …
In the next five years, this will change completely. You won't have to use different apps for different tasks. You'll simply tell your device, in everyday language, what you want to do. And depending on how much information you choose to share with it, the software will be able to respond personally because it will have a rich understanding of your life.
In the near future, anyone who's online will be able to have a personal assistant powered by artificial intelligence that's far beyond today's technology.

- Bill Gates

When building applications with LLMs, we recommend finding the simplest solution possible, and only increasing complexity when needed. This might mean not building agentic systems at all. Agentic systems often trade latency and cost for better task performance, and you should consider when this tradeoff makes sense

Anthropic (https://www.anthropic.com/engineering/building-effective-agents)

https://x.com/AndrewYNg/status/1801295202788983136

What is an agent?

- Multiple definitions exist (Anthropic, Google, LangChain, OpenAI, …)
- Duck typing: If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck

An agent…

- Solves a [complex] task: There is no deterministic solution
- Uses an LLM to solve it
- Interacts with external environments
  - by using *tools*
  - in an intelligent manner
- Is not a predetermined workflow; an LLM [partially] keeps control over the execution flow

Are LLMs not smart enough? How do agents differ from prompt engineering?

What do we know so far

- An LLM predicts tokens in an autoregressive manner
- Scaling laws lead to *in-context learning* and *emerging capabilities*
  - ↑ data
  - ↑ trainable weights
  - ↑ compute

Where LLMs fail

- Updating of their knowledge is difficult and expensive
  - New (fresh) events
  - Domain knowledge
  - Business logic
- Hallucinations
- Interaction with the world

Our answers

- RAG
- Tools (single-agent systems)
- Multi-agent systems
- …

New concepts

- Tool
- ReACT
- Orchestration

Tool

LLMs can generate tokens that do the following:

- Instruct whether to call an API and which one
- Represent a valid payload
- Represent a response for a user by considering previous calls

What is a tool?

- Tools are non-differentiables (i.e., external blocks for an LLM)
- A tool has a description:
  - Certain purpose (it's good for this specific task)
  - Clear input schema (typically, an OpenAPI specification)
- A tool returns a structured and understandable output (text, JSON, success or failure, …)
- Tools can be accessed by LLMs through an external orchestrator

Why tools?

- Examples of poor performance:
  - LLMs are (were) not good at math
  - LLMs don’t know the current weather forecast
- What can we do? Fine-tune or re-train the model to deepen or refresh its knowledge!

Instead, in many cases it’s easier to give an LLM access to external tools that would give it access to specialized skills or fresh knowledge

The final prompt is constructed on the provider's side. There's typically an API available, and you don't care how the input prompt is organized or how to parse the output.

LLMs are good at reasoning

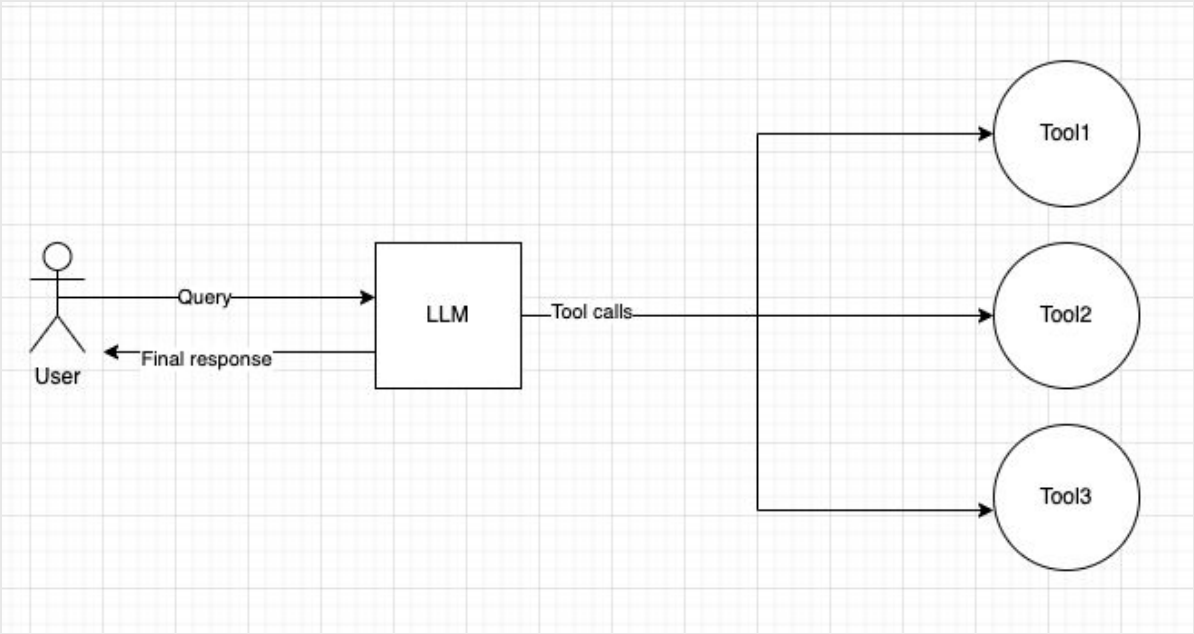
- CoT
- Few-shot CoT
- Self-consistency
- …

LLMs are good at action

- Tool use
- Grounding
- RAG
- …

ReACT

- Reason – ACT: Introduced in 2022 (https://arxiv.org/abs/2210.03629)
- Key idea: Combine two tasks
  - Generate reasoning traces (similar to CoT)
  - Generate task-specific actions


Tool1
Query
LLM
Tool calls
Tool2
Final response
User
Tool3

ReACT

- Expands to new tasks
  - Web browsing
  - Writing code
  - Deep research
  - …
- New architectures
  - Memory
  - Multi-agentic collaboration & communication
  - Planning
  - …

ReACT agent

- Introduce a *while* loop until it decides to answer to the user (exit condition)
- LLM calls a tool -> tool is called externally -> output is appended to the prompt
- On each step, our input to an LLM increases
  - User prompt -> tool call 1
  - User prompt, tool call 1, tool result 1 -> tool call 2
  - User prompt, tool call 1, tool result 1, tool call 2, tool result 2, … -> tool call 3
  - …
  - User prompt, tool call 1, tool result 1, tool call 2, tool result 2, … -> final response

Agents

- Embed an LLM within a program
- *Workflows* vs *agents:* How much an LLM can control the flow?
- You need an external *orchestrator* that maintains the context and the state

Is an agent an FSM?

Is an agent an FSM?

- Embed an LLM within a program
- *Workflows* vs *agents*
  - how much an LLM can control the flow?
- You need an external *orchestrator* that maintains the context and the state

What have we learned so far?

- What is a tool
- LLMs can produce useful generations that decide which tool to call and how to call them
- It’s useful to introduce an external program (a.k.a. an orchestrator that manages the LLM itself) and enhance an LLM with tools to improve its performance on certain tasks
- ReACT combines reasoning and action planning with an external managed loop

Defining tools with LangChain

- You can always provide a full OpenAPI spec
- Tool:
  - Define the schema
  - Define a corresponding sync (and async) function to run the tool

But can we do better? What if we want to wrap an existing function as a tool?

Defining tools with LangChain

- @tool decorator for a function
- Runnable.as_tool
- convert_runnable_to_tool
- StructuredTool.from_function
- …

DEMO time

03

Controlled Generation

Tools for controlled generation

Sometimes we need a controlled generation, i.e., an answer should follow a specific structure

What does it mean that LLMs are good at controlled generation?

What are alternatives to controlled generation?

Controlled generation

- We don’t want wrong instructions + complex parsers
- Many LLMs support *json_mode* (that enforces a JSON generation)
- We also can use tool (function) calling!

What is a natural way of defining a tool for controlled generations?

Controlled generation

- We already have dataclasses/Pydantic models:
  - We can easily auto-compile them to OpenAPI specs
  - … and create a corresponding instance from the LLM's output!
- LC has a special .*with_structured_output* method

04

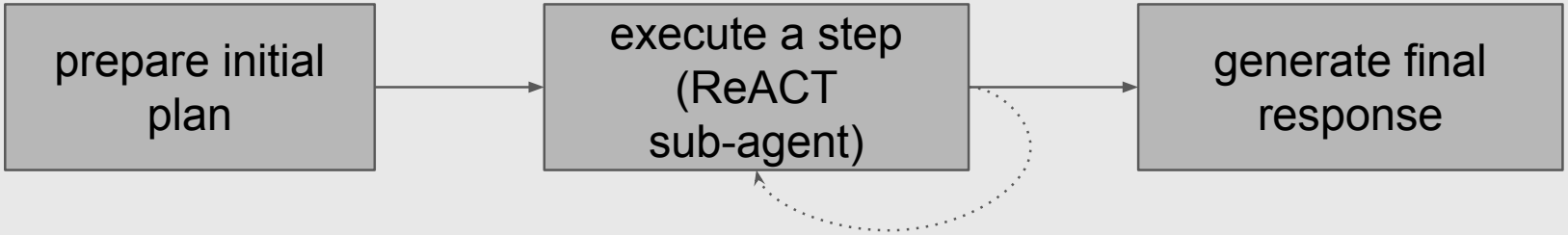
Plan-and-Solve Agent

What is the first thing you do when facing a complex task?

What is the first thing you do when facing a complex task?
You prepare a plan!

Decomposition pattern

- Split a complex task into smaller one
  - You can use an LLM for that!
  - Break down the complex task into sequence of actionable steps
- Avoid very long prompts with many instructions
- Be reasonable


prepare initial plan
execute a step (ReACT sub-agent)
generate final response

Design decisions

- How our state looks like
- How we force an LLM to generate a machine-readable plan
- Prompt templates (how we pass information about previous steps)
- Control recursion depth
- …

Demo time

Questions?