

OpenAI Agents SDK - Open Source

Playlist link

<https://www.youtube.com/playlist?list=PL0vKVrkG4hWovpr0FX6Gs-06hfsPDEUe6>

Inner working of OpenAI Agents SDK | Deep Dive

- Sir Asharib class link →
<https://www.youtube.com/live/5RIADVKEEd8> (unlisted)
- Complete video link →
<https://www.youtube.com/watch?v=Xkg6JBUFFkPY>
- Sir made guide for inner working →
https://github.com/AsharibAli/agentic-ai-projects/blob/main/openai_agents_sdk.md

Core Concepts of Agent

- To see **tracing** (while you are using openAI API) go to this link <https://platform.openai.com/docs/overview> → dashboard → tracing section, here you will see the entire workflow of agents you run till date.
- When you used openai models (which are paid) to make agents, you will get tracing functionality by default on above link
- If you used Gemini or any other model in openai sdk, you will need to used some tracing provider in order to trace it.

Core Concepts of Agent

working of OpenAI Agents SDK

Personal / Default project

Playground Dashboard Docs API reference

DASHBOARD

Logs

Traces

Assistants

Batches

Evaluations

Fine-tuning

Storage

Usage

API keys

Cookbook

Forum

Traces

Workflow Search... Group Search... Metadata

Workflow	Flow	Handoffs	Tools	Execution time	Created
Agent workflow	Coordinator Agent ▶ Asharib	1	0	4.89s	Jun 2, 2025, 3:28 PM
Agent workflow	Coordinator Agent ▶ Language Agent	1	0	35.72s	Jun 2, 2025, 3:24 PM
Agent workflow	Smart Assistant	0	1	17.26s	Jun 2, 2025, 3:08 PM
Agent workflow	Smart Assistant	0	0	5.74s	Jun 2, 2025, 2:58 PM
Agent workflow	Smart Assistant	0	0	17.23s	Jun 2, 2025, 2:50 PM
Parallelization translation	spanish_agent ▶ spanish_agent ▶ spanish_agent ▶ translation_picker	0	0	3.22s	Jun 1, 2025, 4:06 PM
Parallelization translation	spanish_agent ▶ spanish_agent ▶ spanish_agent ▶ translation_picker	0	0	3.62s	Jun 1, 2025, 4:05 PM
LLM as a judge	story_outline_generator ▶ evaluator ▶ story_outline_generator ▶ evaluator ▶ st	0	0	495.28s	May 31, 2025, 6:28 PM
Orchestrator evaluator	orchestrator_agent ▶ spanish_agent ▶ french_agent ▶ italian_agent ▶ synthes	0	3	10.01s	May 31, 2025, 12:33 PM
Agents as tools example	orchestrator_agent ▶ italian_agent ▶ spanish_agent ▶ french_agent ▶ synthes	0	3	32.73s	May 31, 2025, 12:23 PM
Agents as tools example	orchestrator_agent ▶ italian_agent ▶ spanish_agent ▶ french_agent	0	3	17.09s	May 31, 2025, 12:22 PM
Agents as tools example	orchestrator_agent ▶ synthesizer_agent	0	0	4.12s	May 31, 2025, 12:22 PM
Routing example	english_agent	0	0	1.49s	May 31, 2025, 10:44 AM
Routing example	english_agent	0	0	1.70s	May 31, 2025, 10:44 AM

Tools

- Tools are simple python functions which will build to implement custom logic
- Litellm is simple which helps to access different provider tools, like Gemini, Anthropic etc.
- OpenAI called their latest APIs as:
 - Response API → <https://platform.openai.com/docs/api-reference/responses>
 - Completion API → <https://platform.openai.com/docs/guides/completions>
 - Assistant API → <https://platform.openai.com/docs/assistants/>

Open AI SDK

What is the OpenAI SDK?

- A **Software Development Kit** that simplifies integration with OpenAI's APIs.
- Offers tools and abstractions to interact with models like **GPT-4**, **DALL-E**, **Whisper**, etc.
- Enables rapid development of AI-powered applications in various programming environments.
- Release on 11th March 2025 by OpenAI

Open AI SDK

Key Features

- Easy-to-use API client in popular languages (Python, JavaScript, etc.)
- Supports:
 - Chat Completions (GPT models)
 - Image Generation (DALL·E)
 - Audio Transcription (Whisper)
 - Embeddings
 - File and fine-tuning operations
- Built-in support for streaming, error handling, and retries.
- Python-First Design
- Built-in Agent Loop
- Interoperability
- Simplified Multi-Agent Workflows
- Real-World Applications

Open AI SDK

Why Use It?

- Speeds up development with minimal boilerplate code.
- Officially maintained and optimized by OpenAI.
- Ensures compatibility with OpenAI's latest features and API changes.

How to Get Started

- Install via package manager (e.g., `pip install openai`)
- Authenticate with API key
- Use pre-built functions for common tasks

Core Concepts : The Power of Simplicity in Design

The Agents SDK is designed to be **easy to use** but also **powerful**. It's built around 4 main parts:

1. Agents

Smart AI assistants (LLMs) with instructions, tools (like web search), and safety rules. They understand tasks and take actions.

- Pre-built AI models with custom instructions and tools.
- Can understand tasks and respond intelligently.
- Follow built-in safety rules for reliable output.

Core Concepts : The Power of Simplicity in Design

2. Handoffs

Agents can pass tasks to other agents if needed—like teamwork between AI assistants.

- Agents can transfer tasks to other, more specialized agents.
- Enables teamwork between multiple agents.
- Helps handle complex or multi-step tasks more efficiently.

3. Guardrails

Safety checks that control what agents can say or do, helping avoid mistakes or risks.

- Safety checks on what the agent can say or do.
- Prevents harmful, incorrect, or risky outputs.
- Keeps the AI working within set boundaries.

Core Concepts : The Power of Simplicity in Design

4. Tracing & Observability

Helps you see what the agent is doing, step by step—great for debugging and improving your app.

- Shows a clear step-by-step view of what the agent is doing.
- Helps developers debug, monitor, and improve performance.
- Makes it easier to understand and control agent behavior.

OpenAI's SWARM framework

Origins & Concept

- The idea of multi-agent collaboration has been explored in AI research for years.
- OpenAI began experimenting with agent collaboration to handle complex, multi-step tasks more efficiently.
- Initial concept of **SWARM** emerged in 2023, around the same time OpenAI introduced its Agents and tool-use features.
- Internally, OpenAI ran experiments where multiple agents (GPT-based) worked together to solve tasks like coding, planning, and research.
- The term “**SWARM**” became more publicly known in **late 2023 to early 2024** through: Research talks, OpenAI demonstrations, Community interest in **multi-agent collaboration**
- SWARM is still **experimental**, not yet a formal product.

OpenAI's SWARM framework

What is SWARM?

- SWARM is a new experimental framework from OpenAI for building multi-agent systems.
- It lets multiple AI agents work together like a team to solve complex tasks.
- Inspired by how humans collaborate, using division of labor, communication, and coordination.

Key Goals

- **Scalability:** Break large tasks into smaller subtasks for agents to handle.
- **Specialization:** Use different agents with different skills.
- **Autonomy + Collaboration:** Agents work independently but coordinate as a team.

OpenAI's SWARM framework

How SWARM Works

OpenAI's experimental Swarm framework is meant more as an educational tool than a production-ready system—but it introduces several key design patterns for orchestrating multi-agent systems. In Swarm, agents aren't monolithic; instead, they're designed with specialized roles and communicate through clearly defined patterns. Here are some of the core design patterns:

1. Prompt Chaining (Chain Workflow):

This pattern involves breaking down complex tasks into a sequence of simpler, manageable steps, where each step builds upon the previous one. The Agents SDK supports this by allowing developers to define agents that execute specific functions in a predetermined order, ensuring a structured approach to task completion.

2. Routing:

Routing entails directing tasks to the most appropriate agent based on the task's nature. The Agents SDK facilitates this through its handoff mechanism, enabling agents to transfer control to other agents better suited to handle specific subtasks, thereby optimizing task management.

OpenAI's SWARM framework

3. Parallelization:

This pattern focuses on executing multiple subtasks concurrently to enhance efficiency. With the Agents SDK, developers can design agents that operate in parallel, leveraging the SDK's orchestration capabilities to manage simultaneous processes effectively.

4. Orchestrator-Workers:

In this design, an orchestrator agent decomposes a complex task into smaller subtasks and assigns them to worker agents. The Agents SDK's architecture supports this by allowing an orchestrator agent to oversee the workflow and delegate tasks to specialized worker agents, ensuring coordinated task execution.

5. Evaluator-Optimizer:

This pattern involves iterative improvement through feedback loops, where an evaluator agent assesses the performance of other agents and suggests optimizations. The Agents SDK's guardrails feature enables the implementation of such evaluative mechanisms, allowing for continuous performance enhancement and adherence to desired behaviors.

OpenAI's SWARM framework

Core Components

Component	Description
Agent	Language models with specific instructions or skills.
Tasks	Jobs assigned to agents, can be split or passed.
Workspace	Shared space where agents read/write info (like a whiteboard).
Tools	External actions agents can use (e.g., web, code, memory).
Communication	Messaging system between agents to share progress or ask for help.

Benefits of SWARM

- **Handles complex, multi-step problems**
- **Encourages AI teamwork**
- **Easier to debug and understand workflows**
- **Supports modular development** (add or replace agents as needed)

OpenAI's SWARM framework

Example Use Cases

- Research assistants collaborating on a report.
- Customer service agents working together on complex tickets.
- Writing, coding, or brainstorming as a group of specialists

Current Status

- Still in experimental stages (as of 2024–2025)
- Not public yet — demonstrated internally by OpenAI
- Possible future feature in OpenAI's Agents SDK

UV

uv is a modern Python package manager and build system developed by Astral (formerly part of the pdm project). It is designed to be extremely fast, reliable, and easy to use, and is built in Rust for performance.

Key Features of uv:

- **Ultra-Fast:** Much faster than pip and poetry due to being written in Rust.
- **Unified Tooling:** Acts as a drop-in replacement for pip, virtualenv, and pip-tools.
- **Deterministic Installs:** Ensures reproducible builds by resolving dependencies into lockfiles.
- **PEP 582 Support:** Supports local package installation without virtual environments.

Chainlit

Chainlit is an open-source Python framework designed to build and share conversational AI apps powered by LLMs (Large Language Models). It allows developers to quickly create, test, and deploy AI assistants with a front-end interface — all from Python.

Key Features of uv:

- **UI Built-In:** Auto-generates a chat user interface — no front-end needed.
- **LLM Support:** Works with OpenAI, Hugging Face, LangChain, and others.
- **Fast Prototyping:** Build LLM-powered apps in minutes.
- **Realtime Interaction:** Supports async messages, tool use, and streaming.
- **Developer Tools:** Logs, debugging, and interaction tracing included.

Making first Agent using Gemini API

Learn_agentic_ai/01_ai_agent_first:

- We have covered **step 05_chainlit** and **06_chatbot/chatbot** in it
- We have made chatbot one on google colab and other on VS code/cursor using chainlit
- Below is the link of google colab working

https://colab.research.google.com/drive/1mkYAOwlC0yaV0ho2N2BbMZrgXDRGWnaX#scrollTo=-j2Nfiz_C83g

- We have also worked on hello_agent using chainlit when present in folder

07_streaming

- Streaming means the chatbot shows you its response word-by-word or phrase-by-phrase, almost as it's thinking, instead of making you wait for the entire answer to be calculated and then displayed all at once.

Why is this good?

- ***Faster perception:*** You see a response sooner, which feels faster even if the total time is the same.
- ***More engaging:*** It's more like a conversation with a real person because you're seeing the answer develop.
- ***Handles longer answers better:*** Long responses feel less daunting when they appear gradually.

Check folder and repo for code

Repo link → [https://github.com/panaversity/learn-agentic-ai/tree/main/01 ai agents first/07 streaming](https://github.com/panaversity/learn-agentic-ai/tree/main/01%20ai%20agents%20first/07%20streaming)

08_tools

- The OpenAI Agents SDK provides a robust framework for integrating various tools into agents, enabling them to perform tasks such as data retrieval, web searches, and code execution. Here's an overview of the key points regarding tool integration:

Types of Tools:

1. Hosted Tools: These are pre-built tools running on OpenAI's servers, accessible via the [OpenAIResponsesModel]. Examples include:

- WebSearchTool: Enables agents to perform web searches.
 - Try it in Colab: File Search Tool Example
- FileSearchTool: Allows retrieval of information from OpenAI Vector Stores.
 - Try it in Colab: Computer Tool Example
- ComputerTool: Facilitates automation of computer-based tasks.
 - We will use model=computer-use-preview-2025-03-11
 - Note: The model "computer-use-preview" is not available.

2. Function Calling: This feature allows agents to utilize any Python function as a tool, enhancing their versatility.

3. Agents as Tools: Agents can employ other agents as tools, enabling hierarchical task management without transferring control.

Check Repo for detail → [https://github.com/panaversity/learn-agentic-ai/tree/main/01 ai agents first/08 tools](https://github.com/panaversity/learn-agentic-ai/tree/main/01%20ai%20agents%20first/08%20tools)

OPENAI Documentation Github Page

- Below is the link of document

<https://openai.github.io/openai-agents-python/>

Why use the Agents SDK

The SDK has two driving design principles:

1. Enough features to be worth using, but few enough primitives to make it quick to learn.
2. Works great out of the box, but you can customize exactly what happens.

OPENAI Documentation Github Page

Here are the main features of the SDK:

- **Agent loop:** Built-in agent loop that handles calling tools, sending results to the LLM, and looping until the LLM is done.
- **Python-first:** Use built-in language features to orchestrate and chain agents, rather than needing to learn new abstractions.
- **Handoffs:** A powerful feature to coordinate and delegate between multiple agents.
- **Guardrails:** Run input validations and checks in parallel to your agents, breaking early if the checks fail.
- **Function tools:** Turn any Python function into a tool, with automatic schema generation and Pydantic-powered validation.
- **Tracing:** Built-in tracing that lets you visualize, debug and monitor your workflows, as well as use the OpenAI suite of evaluation, fine-tuning and distillation tools.

@dataclass

Understanding Python Dataclasses

- Dataclasses, introduced in Python 3.7, are a powerful way to create classes primarily used to store data.
- They reduce boilerplate code often associated with defining classes for data storage, such as `__init__`, `__repr__`, `__eq__`, and `__hash__` methods.

Why Use Dataclasses?

- Before dataclasses, you might have used plain classes or namedtuple for data structures. While functional, they often required a lot of repetitive code.
- Dataclasses simplify this by automatically generating common methods based on type hints.

@dataclass

Benefits of Dataclasses:

- **Less Boilerplate:** Automatically generates `__init__`, `__repr__`, `__eq__`, `__hash__` (if `mutable=False`), and `__str__`.
- **Readability:** Clearly defines the fields and their types, making the code easier to understand.
- **Type Hinting:** Integrates seamlessly with type hints, improving static analysis and code clarity.
- **Mutable by Default:** Unlike `namedtuple`, dataclasses are mutable by default, but you can make them immutable.
- **Default Values:** Easy to assign default values to fields.

Basic Usage

- To create a dataclass, you import the `dataclass` decorator from the `dataclasses` module and apply it to your class.
- We can define methods, class variable and class methods in `@dataclass`

@dataclass

- Official Code Link → https://github.com/panaversity/learn-agentic-ai/tree/main/00_openai_agents/00_python_syntax
- Code also saved in folder →
G:\osamabinadnan_files\giaic\quarter_04\OpenAI_SDK\OpenAISDK_Working_from_YTPlaylist\Video03_Divingin_sourcecodeof_OpenAIAGentsSDK

```
from dataclasses import dataclass

@dataclass
class Book:
    title: str
    author: str
    pages: int
    price: float

# Creating an instance
book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", 180, 12.99)
print(book1)
print(f"Title: {book1.title}, Author: {book1.author}")

# Dataclasses are mutable by default
book1.price = 10.50
print(book1)
```

@dataclass

- System Prompt vs user Prompt
- <https://openai.github.io/openai-agents-python/ref/agent/>
- Callable method

```
76     name: str
77     """The name of the agent."""
78
79     instructions: (
80         str
81         | Callable[
82             [RunContextWrapper[TContext], Agent[TContext]],
83             MaybeAwaitable[str],
84         ]
85         | None
86     ) = None
```

Understand behind the scene code of Runner

- RSI → Recursive self improvement, in future agent will improve itself by using its memory
- run, run_sync and run_stream are class level static methods which help to run the flow of agents
- Runner works on a loop, which is called **Agent loop**
- RunResultStreaming is also a @dataclass

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Dataclass (@dataclass)

- A dataclass is a shortcut for creating classes that store data. It automatically creates things like the `__init__` (constructor) and `__repr__` (string representation) methods for you. Think of it like this:
- Instead of writing a whole class just to store some variables, you can use `@dataclass` to make your code shorter and cleaner.

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Dataclass (@dataclass)

Example:

python

Copy Edit

```
from dataclasses import dataclass

@dataclass
class User:
    name: str
    age: int
```

Now you can easily create a user:

python

Copy Edit

```
u = User(name="Alice", age=25)
print(u) # Output: User(name='Alice', age=25)
```



Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Generics

- Generics let you write code that can work with any type, without being specific about which one. It makes your code reusable and type-safe. Think of it like this:
- You're saying, “I don't care what type it is yet — I'll fill that in later.”

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Generics

python

Copy Edit

```
from typing import TypeVar, Generic

T = TypeVar('T') # This is a placeholder for any type

class Box(Generic[T]):
    def __init__(self, content: T):
        self.content = content
```

Now you can make a box of anything:

python

Copy Edit

```
box1 = Box("Hello") # Box[str]
box2 = Box(123)      # Box[int]
```



Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Callable

- A Callable is just something you can call like a function. In Python, functions are callables, and so are objects with a `__call__` method. Think of it like this:
- If you can do `something()`, then something is a callable.

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Callable

python

Copy Edit

```
from typing import Callable

def greet(name: str) -> str:
    return f"Hello, {name}!"

def use_function(f: Callable[[str], str]):
    print(f("World"))

use_function(greet) # Output: Hello, World!
```

You're saying: "Give me a function that takes a string and returns a string."

Agent Loop | Tool Call | Hands off | Memory | Guardrails

- Agent is nothing but LLM call
- OpenAI brought ChatCompletionAPI first time
- Then many companies made wrapper on OpenAI's ChatCompletionAPI like LangChain, EasyLLM etc.
- At the bottom is RestAPI → on it ChatCompletion API (which adopted by most of the companies) to chat with LLM → then Langchain
- You just import openai library, change base url, instead to go to openai server, it will go somewhere else (on mentioned URL)
- We can do handoff in LangGraph, AutoGen and in OpenAI Agent SDK as well. Best is OpenAI SDK

Agent Loop | Tool Call | Hands off | Memory | Guardrails

OpenAI Agent Core Concept

- Story begin with LLMs, different companies made it, user asked AGI level questions to them
- OpenAI made it standard at first place when it achieved AGI level
- Whoever (Google, Meta etc.) achieved AGI wrote its SDK.
- It means that they made package in python to talk with LLMs
- Client (Your PC) → send request to model's server
- Your PC send request to model server, the server response it and give you answer.

Agent Loop | Tool Call | Hands off | Memory | Guardrails

- Earlier, this work you can do by making http request restAPI and talk to it
- Then these companies made easy way to used LLMs which they called **SDK**.
- You just need to install python package of SDK; implement it function and ask to your agent.
- The functions of SDK, we can call it ChatCompletion, Assistant API, Responsive API etc.

Agent Loop | Tool Call | Hands off | Memory | Guardrails

A Good Software (OpenAI Agent SDK)

- Good software will be made in layers, otherwise it becomes confusing, same is the case for OpenAI SDK
- First layer is LLM, → on which it wrap with wrapper for example FastAPI for inference → then you call it from client using RestAPI, although we can call in any language but industry is going toward python then we are calling it in python
- LLM → API Server → client (ChatCompletion API which is calling Rest) → Agent SDK

Tool Call

- Tool calling means that an AI agent (like ChatGPT) can use special tools (like a calculator, web search, or code runner) to help it solve problems or get answers.

Example:

- Imagine an AI that can't do math very well in its head.
- So, when you ask it: "What's 37×82 ?", it says, "Let me use my calculator tool!".
- It calls a tool (the calculator), gets the answer, then replies to you.

Tool Call

Does LLMs have access of tools schemas, if yes, how it looks like and works??

- Yes, **LLMs (like ChatGPT or other AI agents)** can have access to **tools** via what's called a **tool schema** (also called function schema, API schema, or OpenAPI spec). This tells the LLM **what tools are available, how to use them, and what inputs/outputs** they take.

What is a Tool Schema?

- A tool schema is like a menu or instruction sheet that tells the LLM:
 - "Here is a tool you can use. This is what it does. Here's how to use it."
- It's usually defined in JSON format, like this (see below picture):

Tool Call

```
{  
  "name": "get_weather",  
  "description": "Get the current weather for a given city.",  
  "parameters": {  
    "type": "object",  
    "properties": {  
      "city": {  
        "type": "string",  
        "description": "The name of the city to get the weather for."  
      }  
    },  
    "required": ["city"]  
  }  
}
```

Tool Call

How Does the LLM Use It?

1. Reads the tool schema during setup or runtime.
2. When you ask something like:
"What's the weather in Paris?", the LLM thinks:
"I have a tool called get_weather and it needs a city. Let me call it with city: 'Paris'."
3. It generates this tool call: ➔

```
{  
  "tool_name": "get_weather",  
  "arguments": {  
    "city": "Paris"  
  }  
}
```

Tool Call

4. The system (not the LLM itself) executes the tool, gets the result (e.g., 22°C and sunny), and gives that back to the LLM.

5. The LLM then replies:

“It’s 22°C and sunny in Paris right now!”

Where Are These Schemas Used?

- **OpenAI Function Calling**
- **LangChain Tools**
- **AutoGPT / Agentic frameworks**
- **Custom APIs** in enterprise AI setups

Stateless Nature of the API

- LLMs are stateless, it means it don't have memory to recall last conversation until and unless you made it stateful.
- To make stateful LLMs send all conversation in every request.
- So, OpenAI made OpenAI SDK over ChatCompletionAPI (stateless layer), which remember all session. You don't need to send all your previous conversion to it.

Agent Loop

An Agent Loop is when an AI agent (like ChatGPT acting like a smart assistant) keeps doing a cycle of:

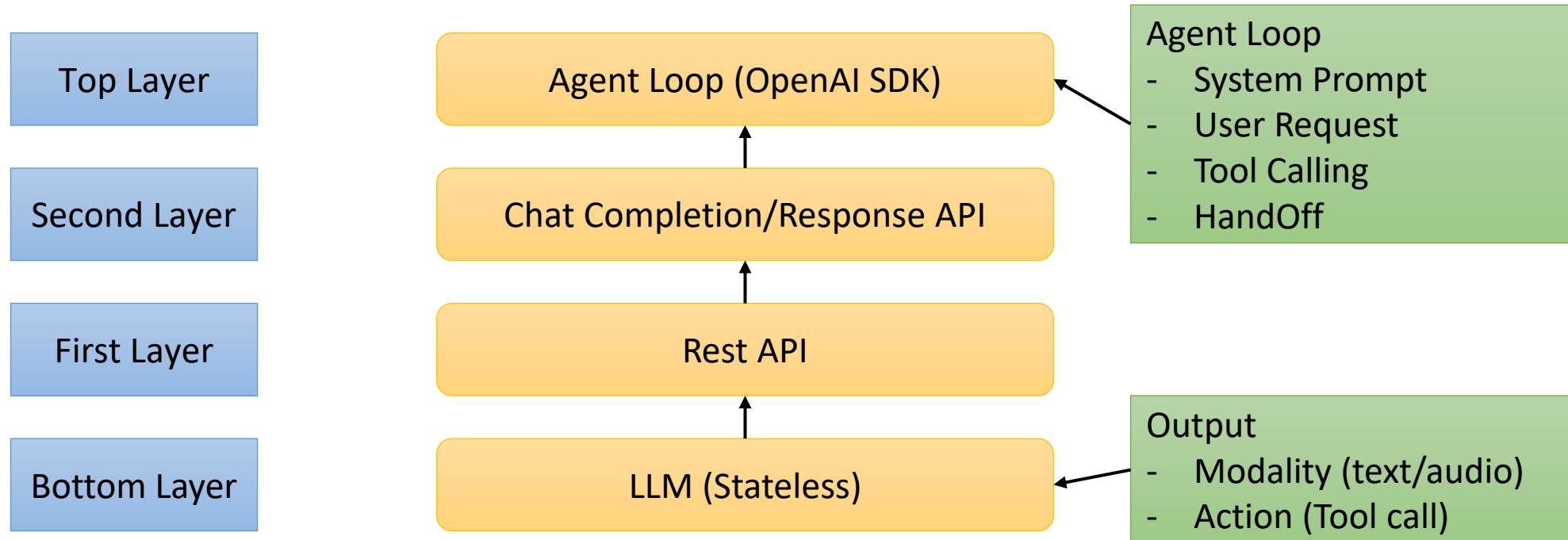
1. Thinking 🧠
2. Taking action (like calling a tool) 🔧
3. Observing the result 👁️
4. Thinking again based on the new result...
5. Repeat 🔄

Until it reaches a final answer or goal ✅.

Thought → Action → Observation → [repeat until done]

Agent Loop

- Agent has short term state, mean all conversation, so whatever task agent will do using tools, it will do by itself.
- Lang chain and Autogen also were also doing the same



Agent Loop / stateless LLMs

- All tools/functions will be called within this loop and all will be done by using OpenAI SDK.
- **First thing is** “Tools are called by Agent loop itself, depend on the user query.”
- As we set LLMs have stateless protocol, so **how can we make agents which don't remember anything?**
- To resolve this problem, LLMs set that you can proceed 2 types of requests simultaneously, first is called **system prompt** other is **user prompt**.
- So, the **second thing is**, “There are two types of prompts”

Agent Loop / stateless LLMs

System Prompt

- A **system prompt** is like a set of instructions or rules given to the AI before it starts the conversation.
- Example system prompt:
 - “You are a helpful and friendly assistant who speaks simply and clearly.” 🧠
- The agent reads this first and keeps it in mind throughout the chat.

User Prompt

- A **user prompt** is what the user says or asks during the conversation.
- Example user prompt:
 - “Can you explain how gravity works in simple terms?”
- The AI uses both the system prompt and user prompt together to decide how to answer.

Agent Loop / stateless LLMs

Do you need to send the system prompt every time in Chat Completion API?

Yes, if you want the AI to remember the instructions.

The Chat Completion API does not store memory between requests.
So:

- If you want consistent behavior, you should include the system prompt in every request as the first message (with "role": "system").
- Think of it like reminding the AI: “Here’s who you are again, and now here’s what the user wants.”
- **Third thing** is, “function calling”
- All these three things are **abstract** (hide) in OpenAI SDK.

Handoff

- In the OpenAI SDK, especially in the context of multi-agent workflows or tool-using agents, a handoff means passing control or a task from one agent (or function/tool) to another based on logic or capability.
- **Handoff** = When one agent (or function within an agent) **decides it can't complete a task** and **delegates it** — either to:
 - A human (human handoff),
 - Another AI agent (agent-to-agent handoff),
 - Or a tool/function (tool/function handoff).
- Agent is the combination of system prompt, user prompt, tool description/calling.
- Handoff is done in tool calling and handoff itself is a kind of tool calling but its abstraction is separate.

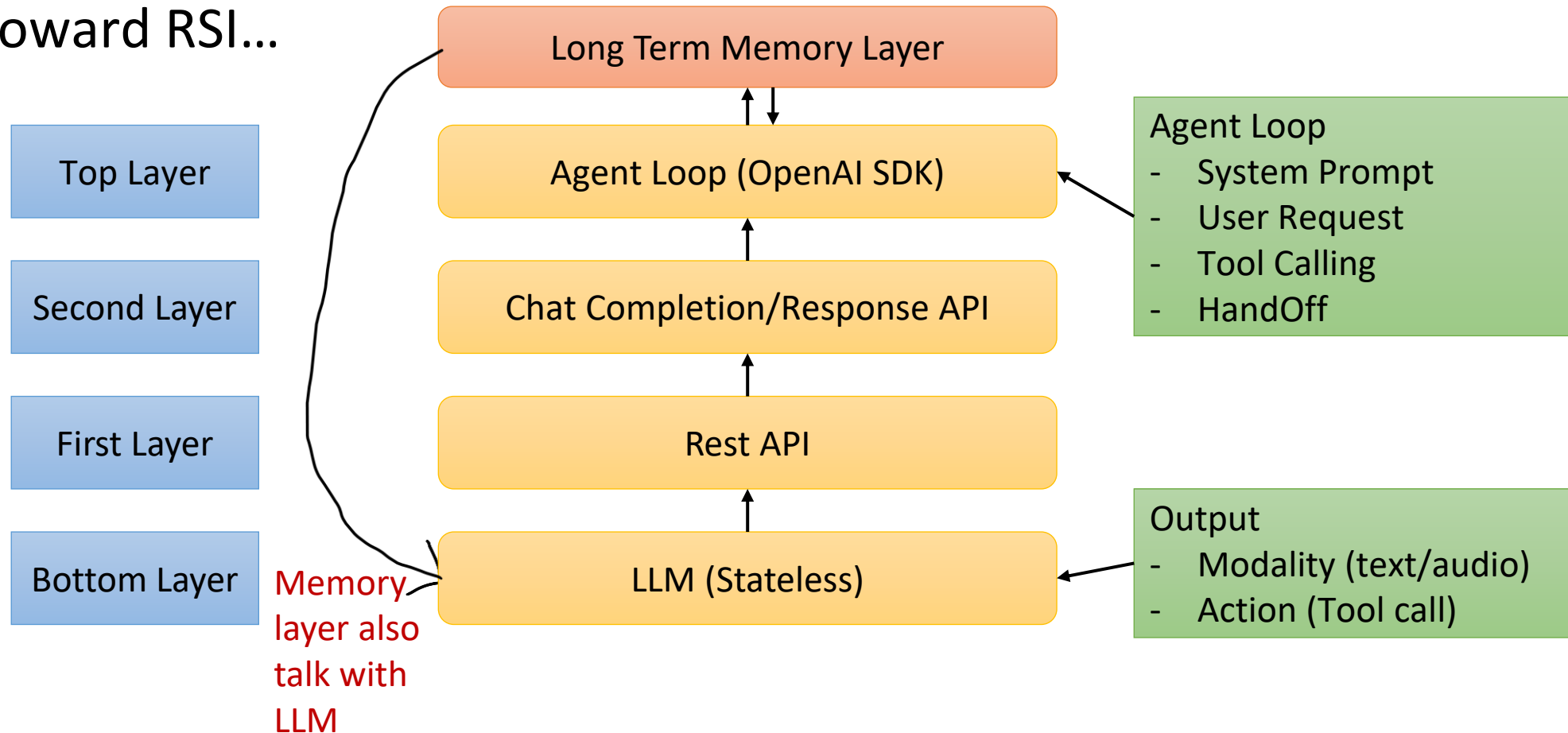
Handoff

- The handoff and tool calling implementation are same from inside because same API is used in both.
- There are 4 types of messages to understand loop, it's all about talk between agent and LLMs
 1. System Prompt/Instruction/Persona
 2. User Prompt/Your Questions
 3. Tool Message (process tool to give answer)
 4. Assistant message/AI message/Agent Reply to user

Memory (Long term)

Memory Layer

- Agent learn from interactions.
- Toward RSI...




Memory (Long term)

- In the **OpenAI Assistants / Agents system**, "memory" means:
- The ability for an agent to **remember facts, preferences, or events** from **past conversations** and use them in future interactions — like a human would.
- This allows the agent to **build context over time**, not just within one conversation.
- You can available memory in tool calling and in system prompt
- Memory layer is also talk with LLM

Memory (Long term)

What is RSI in this context?

- **RSI = Retrieval System Interface**
- It's a part of the architecture that supports memory by retrieving relevant past information (like user history, prior chats, facts) when the agent needs it.
-  *In simple terms:*
- RSI is the system that helps the agent "remember" things by retrieving stored information (from vector databases, notes, past interactions, etc.) and feeding it into the current conversation.

Memory (Long term)

How It Works (Simplified Flow):

- User: *"Remind me what I said about my favorite vacation spot?"*
- Agent uses **RSI** to **search its memory store** (e.g., vector DB or long-term notes).
- It finds: *"User said they love Bali in June."*
- That info is inserted into the context or used to answer.

In memory, we will call LLM in order to reduced the usage of token and just get the context of previous conversations, not full conversation, it means memory will also used LLMs

Memory Management

Memory management is very important:

- To memorize communication between agents
- To remember work while agent performs individual task, keeping in short term or in long term memory
- To remember what tool has done and remember its answer/reply

All these type of stuffs manage via memory layer and to manage this memory in efficient way, there is a package which we called **LangMem**

Work Flow of Agents

- There is a certain difference between work flow and Agentic workflow
- According to Anthropic design pattern article's findings, agents and LLMs decide what will be the sequence of workflow through handoff and tool calling.
- Article link → <https://www.anthropic.com/engineering/building-effective-agents>
- Workflow and agent could be of short term or long term, short term means one session, long term means keeping flowing and state

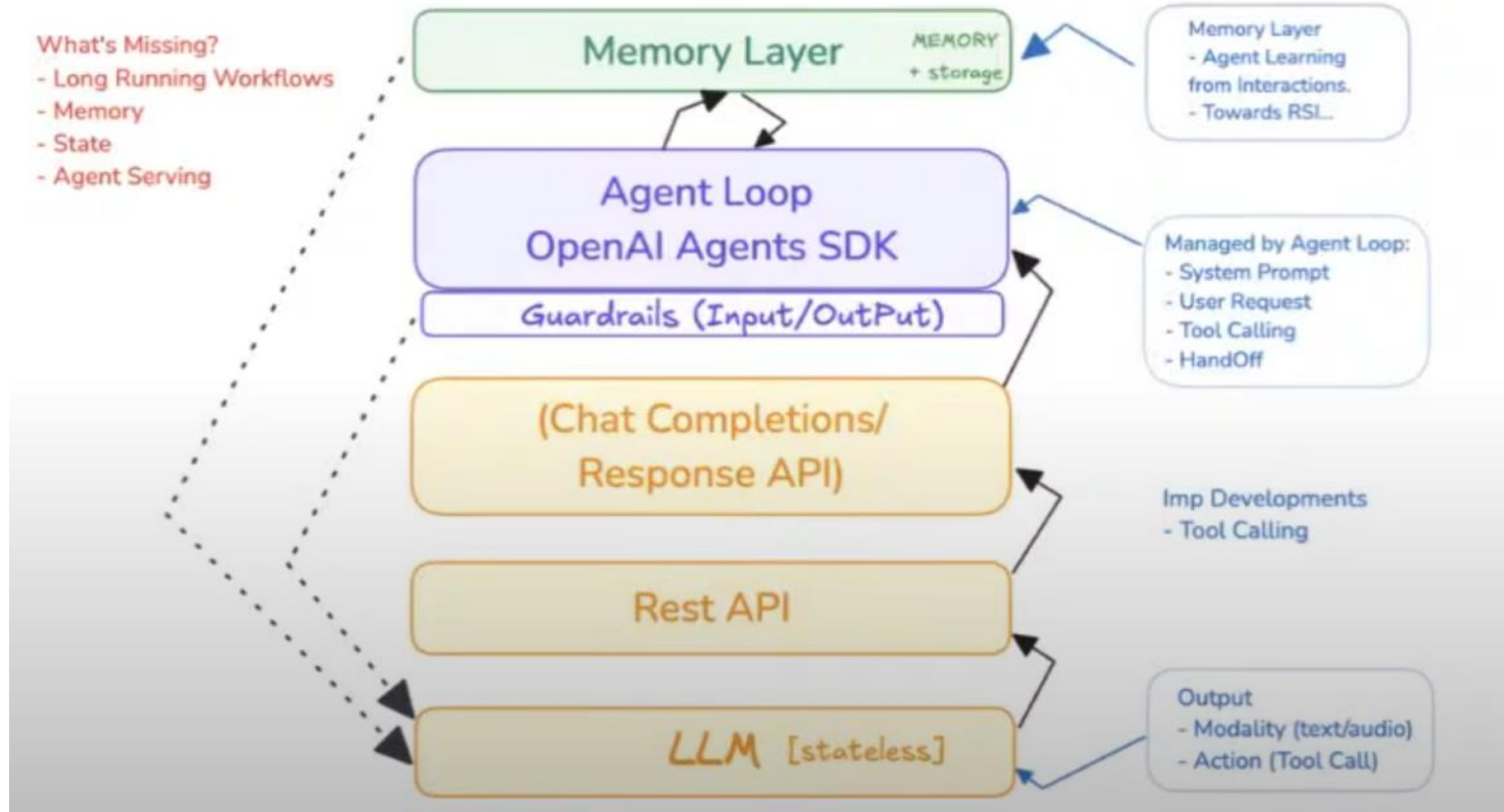
Work Flow of Agents

- There are few implementation for long term memory till this lecture date.
 1. Either you can used AWS step function
 2. Or you can used AWS long running container
 3. Or you can used LangGraph → for medium
 4. Or you can used Temporal services → <https://temporal.io>

Guardrails

- Guardrails are rules, filters, or controls that help keep AI agents safe, reliable, and on-topic during generation or decision-making.
- They limit or guide what the AI can say or do, especially to:
 - Prevent harmful outputs
 - Enforce task boundaries
 - Stay within brand or domain rules
 - Improve trust and accuracy
- Guardrails also talk with LLM.

Guardrails



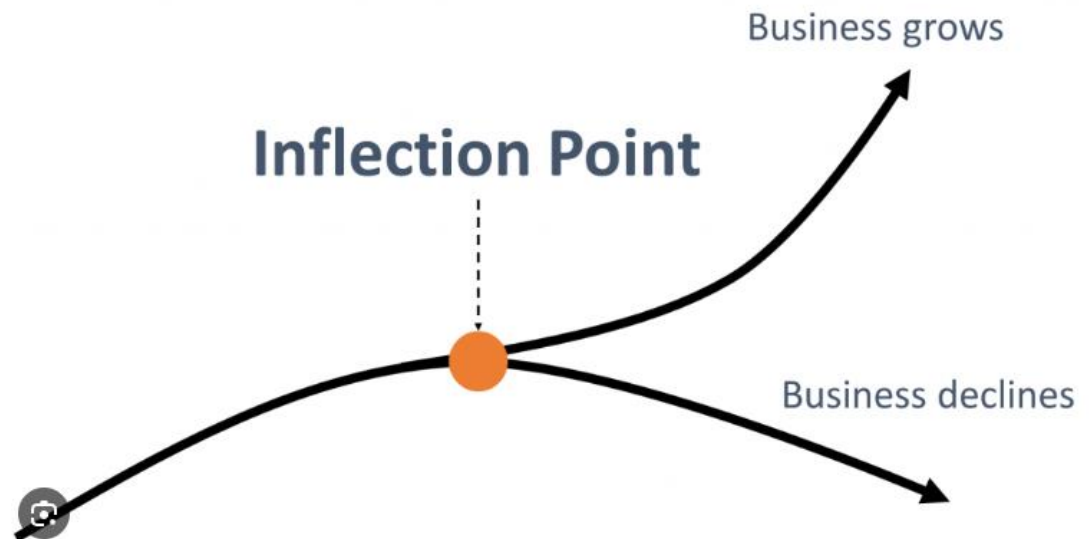
OpenAI SDK

OpenAI SDK provides everything except:

- Memory layer
- And little to no possibly to define long running workflows, it will be implemented by yourself using different external libraries.

Inflection point

- In business, Andy Grove, the former CEO of Intel, defined a strategic inflection point as a significant event that causes a major change in a company or industry, requiring a fundamental shift in strategy.
- It's a turning point where the competitive environment changes, potentially leading to either significant growth or decline. Grove used the term to describe a moment where a company must adapt its approach or face obsolescence.



OpenAI SDK Discussion

- In June 2023, openai introduced function calling and added in chat completion AI.
- Microsoft's Autogen made agent framework before their competitor and released version 0.2
- There are 4 options to make agents
 - **CrewAI:** → comparably easy but lack of logic. Crew → agents → tasks → and crews itself in flows, debugging is very hard/complicated
 - **LangGraph:** → hard to understand states, they mixed up 2 things, i.e., short term agentic (work) flow and long-term workflow.
 - **AutoGen:** Made decision to make agent once again with improve design and workflow which causes division in team so they apart their ways.
 - **OpenAI SDK:**

OpenAI SDK Discussion

- To do analysis properly and in good way, write good prompt
- Few LLMs are become more intelligent, ChatGPT 4.5, Gemini 2.0 pro, Grok 3.
- OpenSDK ➡ 30% framework, 60% python, 10% external libraries for memory and other functionality.

Workflow vs Agents

Workflows (Procedural Execution Style)

- A **workflow** is a **predefined sequence of steps** that the system follows to complete a task. It's deterministic and modular.

Key Characteristics:

- **Static or semi-dynamic**: Steps are known beforehand; the flow may include conditionals or branches, but the control logic is hard-coded or scripted.
- **Orchestrated**: A controller (like a workflow engine or planner) explicitly decides the next step.
- **Predictable**: Good for high-reliability, compliance, or constrained environments.
- **Task decomposition**: Often uses subtasks and tool calls as modular, well-defined operations.



Workflow vs Agents

Workflow Example

Example:

To write and send an email:

markdown

 Copy  Edit

1. Ask user for subject
2. Ask user for body
3. Format the email
4. Send via API

Workflow vs Agents

Agents (Autonomous Execution Style)

An agent is an autonomous, goal-driven entity that uses reasoning to decide what to do next, possibly exploring and adapting dynamically.

Key Characteristics:

- **Dynamic planning**: Agents choose the next action based on current state, tools, and goal.
- **Looping control**: Agents often use think-act-observe cycles (e.g., ReAct or MRKL patterns). More autonomy: Agents decide how to complete the goal, not just follow steps.
- **Stateful**: Agents remember intermediate states, outcomes, failures.
- **Emergent behavior**: Sometimes solutions arise that weren't explicitly programmed.



Workflow vs Agents

Example Agents (Autonomous Execution Style)

Example:

Given a goal like “book a flight to NYC,” the agent might:

markdown

 Copy  Edit

1. Search for flights
2. Extract data
3. Decide it's too expensive
4. Ask user for alternative dates
5. Retry
6. Book via API

No predefined script — the agent decides each step based on the current context and goal.

Short-Term Workflows vs Long-Term Workflows

Short-Term Workflows (Microplans)

Definition:

- **Templated sequences of steps** to achieve subgoals within a single agent loop or session.

Characteristics:

- Focused on **immediate tasks** (e.g., "summarize a document," "extract a table").
- Often cached or retrieved from toolkits (e.g., function graphs).
- Agent **calls or instantiates** them dynamically.

How Agents Use Them:

- As building blocks to solve parts of larger goals.
- Invoked when a known pattern matches the current subgoal.
- Can be **executed as-is**, modified, or aborted mid-way.

Short-Term Workflows vs Long-Term Workflows

Long-Term Workflows (Macroplans)

Definition:

- **Extended, goal-level strategies** or plans that span **multiple steps, sessions, or time horizons**.

Characteristics:

- Represent high-level plans (e.g., “launch a product,” “plan a vacation”).
- May persist across sessions or agent memory.
- Include checkpoints, dependencies, and progress tracking.

How Agents Use Them:

- As **strategic scaffolding**: guides the agent in sequencing major stages.
- Enables **resumability** and **temporal reasoning** over days/weeks.
- Often stored in vector memory, graph structures, or long-term state.

In Practice: Agent + Workflow Fusion

- Agents don't replace workflows — they use them:
- **Short-term workflows:** Like calling a subroutine.
- **Long-term workflows:** Like following a project plan.
- The agent chooses **when** to execute them, **how** to adapt them, or **whether** to abandon them based on reasoning and observations.

Open AI SDK Discussion

OpenAI SDK:

- Is not workflow but agent.
- Does dynamic routing
- Is short term

This make easy to use and you can use it for smaller single session task due to short term.

MCP (Model Context Protocol)

- **MCP (Model Context Protocol)** is an emerging concept/protocol designed to **enable coordination between large language models (LLMs), agentic systems, tools, and environments**.
- It's most prominently associated with **OpenAI's research into AI agents**, introduced around early 2024.
- MCP is a **high-level abstraction layer or interface** that helps manage **context, state, and communication** in complex AI-agent architectures.

What is MCP (Model Context Protocol)?

MCP stands for **Model Context Protocol**. It is a **communication and coordination protocol** designed to:

- **Standardize the interaction between models and their environment** (tools, memory, world states).
- **Allow agents to persist knowledge, state, and goals** across steps or tasks.
- **Help orchestrate multiple agents or model instances** working together, potentially asynchronously.
- **Support long-lived, autonomous agentic behaviors** beyond single-turn prompts.

Think of it as a “**software bus**” or **orchestration layer** for agent-like systems using LLMs.



What does MCP do with Agents?

MCP is a **key enabling infrastructure for building advanced AI agents**, particularly those that:

- **Need memory** (beyond what a single model context can hold).
- **Perform multi-step reasoning or planning.**
- **Interact with multiple tools, APIs, or real-world systems.**
- **Persist identity, goals, or context across time.**
- Coordinate **multiple model instances** (e.g., planner, executor, critic, etc.).

Concretely, MCP does things like:

- **Tracks agent state:** Goals, plans, world model, memories, tasks in progress.
- **Manages context:** So, the model can resume tasks with relevant context without reloading everything.
- **Routes messages:** Between agents, tools, APIs, or humans.
- **Encapsulates prompts + metadata:** Such as system message, tool responses, or user instructions.

Key Capabilities

Key Capabilities

Feature	Description
Context Management	Loads and persists relevant information between agent steps.
Tool Routing	Manages how the model calls and receives outputs from external tools.
Memory Integration	Lets the agent remember past interactions, tasks, or experiences.
Task Decomposition	Coordinates how agents break down and execute complex goals.
Agent-to-Agent Messaging	Enables agents to collaborate or specialize (e.g., sub-agents for planning vs acting).



Example Use Case

Imagine an AI executive assistant that:

1. Remembers your calendar.
2. Plans a trip.
3. Books hotels, reschedules meetings.
4. Learns from feedback.
5. Coordinates with sub-agents (e.g., a flight booking agent, a budget optimizer).

MCP allows this assistant to maintain context over hours/days, route tasks to the right agents or tools, and operate autonomously without forgetting what it's doing between steps.

MCP Discussion

- You can give external info to LLMs by below 3 ways:
 1. You can train model on new data → which is obviously a hard job
 2. You can make RAG (Retrieval Augmented Generation) system, like convert all your document data into vector and place in vector database. **RAG retrieves relevant external documents** (usually stored in a vector database) and then **uses them to generate more accurate and grounded responses**.
 3. You can use external data using tool/function calling, the hardest thing in function calling is to make schema of function, i.e., name, parameter, required parameters, what function does?, what response function will give after processing etc.
- MCP found this (schema issue) gap, MCP hired 1100 sources so developer can connect them to get external info which is very hard to do and time consuming.

MCP Discussion

- It was release in Dec 2024 and it is getting maturity day by day.
- For those 1100 sources, developer had to write tools and schema, that is why MCP was released, **a protocol in which all 1100 sources tools and schema has been written in standard way.**
- MCP operates on a **client server architecture**

MCP operates on a client-server architecture:

- **MCP Hosts:** These are the AI applications (like a chatbot or an IDE plugin) that need access to external data or capabilities.
- **MCP Clients:** These sit within the host and manage secure, one-to-one connections to servers.
- **MCP Servers:** These are lightweight programs that expose specific tools, data, or resources (e.g., a GitHub server might provide repository access) to the AI.

Development to Deployment & Introduction to DACA Design Pattern

Discussion

- If we created virtual machine, it's obvious it is stateful (memory), but the problem is we can't scale it up.
- Cloud owners determined that the virtual machine is not as good respect to scalability and made it light weight version which is 'containers', containers are stateless.
- You give request to container, container ups → load state from database → do your assign work → it's saved some stuff database → Give response back to you → then forget everything about you.
- So, the design pattern which give you maximum scalability those are stateless containers of '**Docker**'. These are also called 'Serverless Containers'

Development to Deployment & Introduction to DACA Design Pattern

Discussion

- As of today, the latest state-of-the-art stuff of cloud are serverless container (managed) or stateless container.
- OpenAI SDK has record of complete conversation, when you call it in stateless method then you can save all conversation in database by single call, by this you can make scalable agent how? Stateless protocol, you made stateless FastAPI server which is inside container, when the request comes, it pick detail from database
- To make applications, we have to merge cloud native technology and agentic AI, by merging these we can even make systems of Microsoft level.

Development to Deployment & Introduction to DACA Design Pattern

Discussion

- When we try to make our system serverless then it core root in Docker, we make microservices from docker and to manage these microservices (containers) using Kubernetes.
- **Kubernetes** auto manage containers, if million, billion users come to your app, it will up million billion containers/microservices
- Although, Kubernetes is very hard to learn and we have to make serverless system, so we will learn Docker (as a base for serverless) and **DAPR**.

DACA (Dapr Agentic Cloud Ascent)

Dapr Agentic Cloud Ascent (DACA) is a design pattern for building and scaling agentic AI systems using a minimalist, cloud-first approach. It integrates the OpenAI Agents SDK for agent logic, MCP for tool calling, Dapr for distributed resilience, and a staged deployment pipeline that ascends from local development to planetary-scale production. DACA emphasizes:

- **AI-First Agentic Design:** Autonomous AI agents, powered by the OpenAI Agents SDK, perceive, decide, and act, with **MCP** enabling tool access and **A2A** facilitating intelligent agent-to-agent dialogues.
- **Agent-Native Cloud Scalability:** Stateless containers deploy on cloud platforms (e.g., Azure Container Apps, Kubernetes), leveraging managed services optimized for agent interactions.
- **Stateless Design:** Containers that scale efficiently without retaining state.
- **Dapr Sidecar:** Provides state management, messaging, and workflows.
- **Cloud-Free Tiers:** Leverages free services for cost efficiency.
- **Progressive Scaling:** From local dev to Kubernetes with self-hosted LLMs.

DACA (Dapr Agentic Cloud Ascent)

Discussion:

- **Cloud Ascent** refers to the process of developing an application locally—such as on a laptop during the development phase—and seamlessly scaling it to support millions of users in the cloud as demand grows.
- And this cloud is for agents
- Is DACA is framework or design?? Check appendix III in below link
- https://github.com/panaversity/learn-agentic-ai/blob/01e344ba85ec36134c783b5ceef45a12a9bb7e68/comprehensive_guide_daca.md#appendix-iii-daca-a-design-patter-or-framework

DACA (Dapr Agentic Cloud Ascent)

Discussion:

- The **Dapr Agentic Cloud Ascent (DACA)** is best classified as a **design pattern**, though it has elements that might make it feel framework-like in certain contexts. Let's break this down to clarify its nature and why it fits the design pattern label, while also addressing the nuances that might lead to confusion.
 - **Framework:** A reusable, pre-written set of code or libraries that provides a structure for developing applications. It often dictates the flow (e.g., React, Django).
 - **Design Pattern:** A general, reusable solution to a common software design problem. It's a conceptual template, not actual code (e.g., Singleton, Observer).
 - **In short:** Framework = implementation + structure. Design Pattern = conceptual solution

DACA (Dapr Agentic Cloud Ascent)

Discussion:

- In DACA, we are using DAPR (DAPR is tech which is used to distribute microservices which communicate with each other), Agents (any tech used to make agents) and Cloud (any tech for cloud native)
- **DACA as a Pattern:** It's a high-level strategy for agentic AI systems, focusing on architecture (three-tier, EDA), principles (statelessness, HITL), and deployment stages (local to planet-scale). It doesn't provide a runtime or library—you build the system following its guidance.
- **Not a Framework:** DACA doesn't offer a pre-built runtime, APIs, or enforced conventions. While it suggests tools (e.g., Dapr, FastAPI), these are optional, and the pattern's core is about *how* to structure the system, not *what* to use.