Name: Ahmed Yaqoob Dhedhi

OpenAI Agents SDK Questions And Answers!

---

1. What’s the main advantage of custom tool behavior functions?

The main advantage of custom tool behavior functions in the OpenAI SDK (often referred to as "function calling" or "custom tools") is that they allow you to extend the capabilities of the language model by connecting it to external system and data.

Example in OpenAI Agents SDK

- Let's imagine you want to build an AI assistant that can tell you the current weather in a city. Since the LLM doesn't have real-time weather data, you'll provide a custom tool/function calling.

2. Which method is used to execute an agent asynchronously?

In the OpenAI Agents SDK, the primary method used to execute an agent asynchronously is `Runner.run()`

Example in OpenAI Agents SDK

- To use `Runner.run()`, you'll typically need to use Python's `asyncio` library to manage the asynchronous execution.

3. What’s the purpose of RunContextWrapper?
Think of `RunContextWrapper` as a special backpack that you give to your robot *every time it starts a new job*.

Inside this backpack, you can put any useful information that the robot might need for *that specific job*, but which isn't part of the main conversation.

4. What does Runner.run_sync() do?
In the OpenAI Agents SDK, `Runner.run_sync()` is a direct way to execute an agent. Think of it as telling the agent, "Do this task now, and tell me when you're completely done."

The key word here is "synchronous." This means that when you use `run_sync()`, your program will pause and wait for the agent to finish its entire operation and provide a final result. Only after the agent is finished and has given back its output will your program move on to the next line of code.

5. What does extra="forbid" do in a Pydantic model config?
`extra="forbid"` is like saying: "Only fill in the blanks I've provided on this form. Don't add any extra information, even if you think it's helpful. If you write anything outside of the designated spots, I won't accept the form."

So, if your form asks for "Name" and "Age," and someone tries to write "Name," "Age," and "Favorite Color," the form with `extra="forbid"` will reject it completely because "Favorite Color" wasn't an expected field

6. What's the main advantage of strict schemas over non-strict?

The main advantage of strict schemas over non-strict (or "schemaless") ones, especially in the context of interactions with AI models like those provided by OpenAI, is guaranteed data integrity, predictability, and reliable programmatic handling of outputs.

7. What happens when you combine StopAtTools with multiple tool names?

Combining `StopAtTools` with multiple tool names in the OpenAI Agents SDK configures an agent to immediately halt its execution and return the output of the first invoked tool that matches any name in the specified list, without further LLM processing of that tool's result.

8. What is the purpose of context in the OpenAI Agents SDK?

Imagine you're building a highly intelligent assistant. This assistant needs to remember things, access tools (like a calendar or a calculator), and understand who it's talking to. "Context" in the OpenAI Agents SDK is like a backpack that you give to your assistant for a specific task

9. What is the purpose of context in the OpenAI Agents SDK?

`tool_use_behavior` defines how an AI agent interacts with and utilizes its external tools, ranging from full autonomy to strict, predefined usage.

10. What information does handoff_description provide?

The OpenAI Agents SDK is a framework for building multi-agent AI systems, and a key feature it provides is the ability for agents to "hand off" tasks to more specialized agents. The `handoff_description` is a crucial piece of information within this mechanism.

11. What does ToolsToFinalOutputResult.is_final_output indicate?
In the OpenAI Agents SDK, ToolsToFinalOutputResult.is_final_output is a boolean flag that indicates whether the result of a tool call (or a series of tool calls) should be considered the final output of the agent's current run.

12. What's the difference between hosted tools and function tools in terms of tool_use_behavior?

Function Tools: You provide the custom code, and your program runs it.

Hosted Tools: OpenAI provides the ready-made features, and OpenAI's systems run them.

13. How do you convert an agent into a tool for other agents?

To convert an agent into a tool for other agents in the OpenAI Agents SDK, you use the `agent.as_tool()` method. This allows a specialized agent to be wrapped as a callable function that other agents can utilize within their workflows.

14. What method returns all tools available to an agent?

In the OpenAI Agents SDK, the `Agent` class has a method called `get_all_tools()`.

15. What is the first parameter of every function tool?

The first parameter of every function tool in the OpenAI Agents SDK is a ToolContext object (or `RunContextWrapper`).

16. What is the purpose of the get_system_prompt() method?

In the OpenAI Agents SDK, the get_system_prompt() method's purpose is to provide the core instructions or "system prompt" to an AI agent.

17. What's the difference between InputGuardrail and OutputGuardrail?

InputGuardrail: Checks what you tell the AI agent *before* it processes it.

OutputGuardrail: Checks what the AI agent tells you back *after* it has generated a response.

18. What is the primary purpose of the instructions parameter in an Agent?

It directly tells the underlying Large Language Model (LLM) what the agent's purpose is, how it should behave, and what task it needs to accomplish.

19. What does the reset_tool_choice parameter control?

Whether to reset the tool choice to the default value after a tool has been called. Defaults to True. This ensures that the agent doesn't enter an infinite loop of tool usage.

20. What happens if a function tool raises an exception?
If a function tool in the OpenAI Agents SDK raises an exception, the agent run will typically fail unless you've designed the tool to catch exceptions and return an error message to the LLM instead.

21. What does the clone() method do?

Make a copy of the agent, with the given arguments changed.

22. What is ToolsToFinalOutputFunction in the OpenAI Agents SDK?

ToolsToFinalOutputFunction in the OpenAI Agents SDK is a function that tells the agent to use the output of a tool (or tools) directly as its final answer, rather than continuing to process with the language model. It's used for scenarios where a tool's result is sufficient as the complete response.

23. What is the return type of Runner.run()?
The Runner.run() method in the OpenAI Agents SDK returns a RunResult object.

This RunResult object contains information about the agent's execution, including:

- The final output of the agent.
- All the inputs and outputs during the run.
- Guardrail results.

24. What happens if a custom tool behavior function returns is_final_output=False?

If a custom tool behavior function returns is_final_output=False, the agent's Large Language Model (LLM) will run again using the tool's output to continue its reasoning and take further actions until it determines it has reached a final answer.

25. When would you use StopAtTools instead of stop_on_first_tool?

- stop_on_first_tool: Stops the agent immediately after it uses *any* tool for the first time.
- StopAtTools: Stops the agent only if it uses one of the *specific tools you list*.

26. What is the default value for tool_use_behavior in an Agent?

Run_llm_again

27. What's the key difference between run_llm_again and stop_on_first_tool in terms of performance?

run_llm_again means the AI will think *again* after using a tool, adding more time/cost but making it smarter. stop_on_first_tool means the AI stops *immediately* after using a tool, making it faster/cheaper but less smart about the tool's result.

28. Which Pydantic v2 decorator is used for field validation?

In Pydantic v2, the decorator used for field validation is @field_validator.

29. What is the purpose of model_settings in an Agent?

`model_settings` in an OpenAI Agent allows you to configure the underlying Large Language Model (LLM), controlling aspects like its creativity (`temperature`), output length (`max_tokens`), and how it uses tools (`tool_choice`). It gives you fine-grained control over the LLM's behavior.

30. How do you enable non-strict mode for flexible schemas?

To enable non-strict mode for flexible schemas in the OpenAI Agents SDK, you generally set the strict_json_schema parameter to False when defining your schemas.

31. What does the handoff_description parameter do?

The `handoff_description` parameter in the OpenAI Agents SDK provides a brief summary of an agent's function or expertise, helping other agents (like an orchestrator) decide when to transfer a task to it.

32. How do you implement schema evolution while maintaining backward compatibility?

Implement schema evolution with backward compatibility in OpenAI Agents SDK by prioritizing additive changes (adding optional fields or new tools) and, for breaking changes, versioning your tools (e.g., `tool_v1`, `tool_v2`) while maintaining older versions for a transition period, leveraging Pydantic for schema definition and robust testing.

33. Can dynamic instructions be async functions?

Both regular and `async` functions are accepted.

34. What happens when tool_use_behavior is set to 'stop_on_first_tool'?

When `tool_use_behavior` is set to `'stop_on_first_tool'` in the OpenAI Agents SDK, the agent executes the first tool call it determines is necessary, and then immediately returns the output of that tool call as its final response, without any further processing by the LLM.

35. What's the difference between mutable and immutable context patterns?

Mutable context can be changed during an agent's run, used for dynamic data like state or memory. Immutable context cannot be changed once set, used for static data like configuration.

36. What causes the error 'additionalProperties should not be set for object types'?

The error "'additionalProperties' should not be set for object types" typically occurs in JSON Schema validation (common with Pydantic/OpenAPI)

37. When should you use non-strict schemas over strict schemas?

You should use non-strict schemas in the OpenAI SDK when you want the model to have flexibility in its output and are prepared to handle potential variations or missing fields=

38. In a custom tool behavior function, what parameters does it receive?

The parameters a custom tool behavior function receives are the arguments the language model (LLM) generates based on the tool's defined schema and the conversation context.

39. What does Runner.run_streamed() return?

`Runner.run_streamed()` in the OpenAI Agents SDK returns a `RunResultStreaming` object, which allows you to asynchronously stream events from the agent's execution.

40. How do you create dynamic instructions that change based on context?

In the OpenAI Agents SDK, dynamic instructions are primarily created by providing a function to the `instructions` parameter of an `Agent`, which generates the system prompt based on a `context` object.

By: Ahmed Yaqoob Dhedhi