OpenAI Agents SDK (Simplified)

Made by: *Jamila Yaqoob Dhedhi*
***Roll Number**: 00305277*

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

Custom tool behavior functions allow tailored control over how tools are executed, enabling flexibility to meet specific needs, like conditional logic or dynamic responses.

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

`await Runner.run()`

3. What's the purpose of RunContextWrapper?

RunContextWrapper manages the execution context, ensuring proper state handling and resource cleanup during agent runs. Thus, it wraps and manages the runtime context (memory, state, etc.) for agents.

4. What does Runner.run_sync() do?

Runs the agent in sync (non-async) mode. Useful if you're not using `async`.

5. What does extra="forbid" do in a Pydantic model config?

Prevents users from passing extra, undefined fields. It prevents additional, undefined fields in a Pydantic model, enforcing strict validation.

Example Code Sinppet:

```python
from pydantic import BaseModel, ConfigDict

class User(BaseModel):
    model_config = ConfigDict(extra="forbid")
    name: str
    age: int

# This will raise an error
user = User(name="Jamila", age=20, extra_field="invalid")
```

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

Strict ensures only defined fields are allowed. Safer and predictable.

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

Stops execution when any of the listed tools are called. It pauses execution when any specified tool is called, allowing selective tool-based control.

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

It stores temporary data like tool results or user session info. It provides runtime information (e.g., user inputs, session state) to agents for informed decision-making.

9. What's the key consideration when choosing between different tool_use_behavior modes?

It's about speed vs control: e.g., `stop_on_first_tool` is faster but less flexible. There are others allow more complex tool interactions.

10. What information does handoff_description provide?

Explains why a tool handoff is happening. It explains how an agent delegates tasks to another agent or tool, aiding transparency.

11. What does ToolsToFinalOutputResult.is_final_output indicate?

It signals whether the tool's output is the final result or requires further processing.

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

Hosted tools are managed externally with predefined behavior, while function tools allow custom logic defined within the code.

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

Use `Agent.as_tool()` method.

```
tools=[
    spanish_agent.as_tool(
        tool_name="translate_to_spanish",
        tool_description="Translate the user's message to Spanish",
    ),
    french_agent.as_tool(
        tool_name="translate_to_french",
        tool_description="Translate the user's message to French",
    ),
    italian_agent.as_tool(
        tool_name="translate_to_italian",
        tool_description="Translate the user's message to Italian",
    ),
],
```

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

`agent.list_all_tools()`

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

`context` (the current agent state/context).

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

It retrieves the system prompt defining the agent's behavior and instructions.

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

Input = filter/validate user input. Output = validate LLM response.

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

Tells the LLM what the agent is and how it should behave.

19. What does the reset_tool_choice parameter control?

If True, the agent forgets previously chosen tools. It determines whether the agent resets its tool selection logic after each running.

20. What happens if a function tool raises an exception?

It's caught and shown as a tool error in the agent output.

21. What does the clone() method do?

Makes a copy of an agent, keeping its settings and state.

22. What is ToolsToFinalOutputFunction in the OpenAI Agents SDK?

I It’s a function that converts tool outputs into the final response format expected by the agent.

23. What is the return type of Runner.run()?

`RunResult` – includes outputs, tool calls, status.

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

The agent will continue running and possibly call more tools.

25. When would you use StopAtTools instead of stop_on_first_tool?

When you only want to stop at specific tools, not any. Thus, Use StopAtTools for selective pausing at specific tools, offering more granular control than stop_on_first_tool.

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

`auto` – decides behavior automatically.

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

`stop_on_first_tool` is faster; `run_llm_again` gives better results sometimes.

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

`@field_validator`

```
from pydantic import BaseModel, field_validator

class User(BaseModel):
    age: int

    @field_validator("age")
    def check_age(cls, v):
        if v < 0:
            raise ValueError("Age cannot be negative")
        return v
```

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

configures the underlying LLM’s parameters, like temperature or max tokens.

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

`extra='allow'` in Pydantic model config.

```python
from pydantic import BaseModel, ConfigDict

class User(BaseModel):
    model_config = ConfigDict(extra="allow")
    name: str
```

31. What does the handoff_description parameter do?

Describes what the tool does and why the LLM should use it.

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

Use optional fields and defaults for new values.

33. Can dynamic instructions be async functions?

Yes! They can fetch data and update instructions.

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

The agent pauses execution after the first tool call, returning the tool's output.

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

Mutable contexts can be modified during execution, while immutable contexts are read-only, ensuring consistency.

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

Occurs when schema is strict but input has extra keys.

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

When you want to allow extra fields (e.g., during development).

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

`context`, `tool_choice`, `tools`, and `agent_state`.

39. What does Runner.run_streamed() return?

A generator that yields tokens in real time.

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

Use a function for the `instructions` parameter.

```python
def dynamic_instructions(context):
    if "weather" in context.user_input.lower():
        return "You're a helpful weather assistant."
    return "You're a general assistant. Be polite and helpful."

agent = Agent(
    instructions=dynamic_instructions,
    tools=[get_weather, get_news]
)
```

Happy Learning!