Agent SDK Core Concepts

Q: What is agent?

A: Agent is an intelligent, autonomous unit that can decide and acts autonomously without getting user interruption.

In Terms Of Agent SDK

- Agent is a brain
- use tools to get real world info
- follows instructions

Agent Loop Lifecycle

Think → Act → Observe → Respond/Repeat

Features

- use multiple tools
- works step by step
- follows system level instructions
- can be orchestrated

Q: Why use Agent SDK?

A: Features Of Agent SDK

- built-in tool use.
- easy multi agent and tool orchestration
- open source and customizable.
- structured, safe and traceable.
- simple to use.
- transparent agent loop.
- python first approach.

Agent	Tool	Runner
• brain	• called by agent	• controls agent loop
• runs a loop	• do external work	• start & manages loop
• maintain context	• accept structured input	• accept user input
• construct prompt	• executes actual work	• returns final result
• predicts action	• returns str output	• handles sync/async ex
	• has metadata (name, des, inp_sch, out_sch)	• pass cont, steps, tool

user prompt → Runner

↓

Agent (Think + decide)

↓

↓ Final Answer

↓ Toolcall → Tool → Result

↓

Repeat ← Agent observe

Core Components

① AGENTS

- core building block
- LLM configured with instruction and tools

Basic Configuration

- common properties
  - instructions
  - model
  - tools

Arguments Of Agent Class

- Required
  - name
- Optional
  - instructions
  - handoffs
  - model (if using openAI)
  - model_setting
  - tools
  - tool_use_behaviour
  - output_type
  - hooks

instruction

ss

openAI)

Default Model : gpt 4o.

Context:

- use to pass or manage data or state through execution loop
- encapsulated in RunContextWrapper class
- RunContextWrapper make the context acessible by agents, tools, handoffs.

Output Type

- gives particular type of output
- structured
- can be str, pydantic class, data class

AgentOutputSchemaBase

Handoffs

- specialized sub-agents
- handoffs parameter in agent

Dynamic Instructions

- allows to dynamically generate or modify agents instructions at runtime.

Tool Choise Behaviour

-

② RUNNING AGENTS

- run agent via Runner class:
  - Runner.run():
    runs async and returns RunResult
  - Runner.run_sync():
    - sync method
  - Runner.run_streamed():
    - RunResultS

Agent Loop:

LLM call
↓
output
↓

final_output → loop ends

handoff → agent update

tool call → loop rerun

- max turns parameter passes in Runner class
- for max_turns = 1, loop can run only 1 time

Run_Config

- default_model_provider = OpenAI
- allows to configure global setting for agent run

Exception

- AgentsException
  - base class
  - a) Max Turns Exceeded
  - b) Model Behaviour Error
  - c) User Error
  - d) InputGuardrailTripwireTriggered
  - e) OutputGuardrailTripwireTriggered

③ Streaming :

- additional import :
  from openai.types.responses
  import ResponseTextDeltaEvent
- Runner.run_streamed
- returns RunResultStreaming Object
- contains 3 dataclasses
  a) Raw Responses Stream Event
     - used to get final answer chunks by chunks.
  b) Run Item Stream Event
     - used to get the steps performed by LLM ~~etc~~ e.g tool_call, tool_output etc.
  c) Agent Updated Stream Event
     - notify when new agent running

③ RESULT

- RunResult
  - run() or run_sync()
- RunResultStreaming
  - run_streamed()
- Input for next turn
  - result.to_input_list + ''
- Last_agent
  - result.last_agent.name/ins/model
- New Items:
  - MessageOutputItem, ToolCallItem etc
- Guardrail Result
  - result.input_guardrail_results
  - result.output " " " " "
- ~~RawResponses~~
  - contains model responses generated by LLM.
- Input

④ Tools

- 3 classes of tools
  1. Hosted Tools
     - built-in
     - only run in OpenAIResponsesModel
  2. Function Tools
     - python functions as tool
  3. Agents as Tools
     - agent can be used as tool
     - using .as_tool () method
- Custom Function Tool
  - provide name, des, params_json_schema, on_invoke_tool.
- Errors In Function Tool

  by default:

  default_tool_error_function

  custom error:

  - pass your custom error function

  None:

  - any tool call error will be raised.

④ Tools:

⑤ Handoffs

- represented as tools to agents

Creating Handoff:

```
triage = Agent (name = "Triage",
    handoffs = [billing_agent, handoff
                (refund_agent)])
```

Handoffs

- para in agent
- uses variable name in which details were saved as an agent name
  e.g billing_agent = Agent(name = " — ")
  in handoffs name of agent = "billing_agent"
- two-way handoff

handoff()

- one way handoff
- method contains para (agent, tool_name_overide ... etc).

⑥ TRACING

- only openAI models using agent can be traced on openAI model
- for non-openAI models we need to use external tracing processor
- enable by default
- globally disabled
  - ~~op~~ OPENAI_AGENTS_DISABLE_TRACING=1
- Single Run Disable
  - agents.Run.RunConfig.tracing_disabled=Tru

Traces And Spans

- Trace
  - entire lifecycle of logical operation
- Span
  - specific operation

Custom Tracing Processors

- user defined fun or classes
- receives & handles trace or span data

Creating Trace And Span

- manually marking & recording sections of code
  (trace = full operation, spans = specific steps)

Panaversity (ai_agents_first)

Readme.MD

Q: Why agentclass is describe as dataclass?

A: For practical and structural reason

① Simply Class Initialization

- generates __init__, __repr__, __eq__ and other methods automatically.

② Improves Readability

- keeps class clean and creative
- focus on behaviour, not boiler plate
- easy to convert agent into dict, save/load config as JSON
- safely define list and optional fields with default factories

③ Works Well With Pydantic

- dataclass make use of typehints and schemas, easy.

④ Flexibility

- Class can be

ImMutable

Frozen=True (Not used in SDK)

Mutable

field(default_factory=list)

(2): Why system prompt contained in Agent as instructions? Why can also be setted as callable?
Use prompt passed as Parameter in run method and is a class method?

A: System Prompt Vs User Prompt

- System prompt defines identity and behaviour of agent
- Instructions is part of agents definition (e.g job or persona)
- makes agent reusable across runs
- Callable is used to generate dynamic prompt

- Runner is a class method b/c we don't need to create its instance b/c it is stateless
- User Prompt passed in runner b/c User query will be different everytime. It Cannot be set once

ontained in
Why can abe

Parameter
is a

User Prompt

lentity and

agents
ersona)
cross runs
generate

thod
create
stateless
unner
different
once.

6) Purpose Of Runner

- Centralized Execution Logic
  - handles full logic for
    - Constructing msgs, running agent loop, tool call, streaming output, applying guardrails returning response
- Agent Loop Implementation
  - runner implements agent loop lifecycle : System → User → Model → Tool
    Final Output ← ... ← Model ↵
  - This loop is hard to manage manually.
  - Runner abstract its away
- Handles Context And Tracing
  - manages AgentContext
  - Optional tracing
  - Optional streaming callbacks
- Asynchorous Execution
  - easily run multiple agents concurrently

Tools

- special functions or ~~call~~ classes that can be called by an agent to perform specific tasks
- add custom behaviour to agent

Tool Class

```
class Tool(BaseModel):
    name : str
    description : str
    input_schema : Type[BaseModel]
    output_schema : Type[BaseModel]
    handler : Callable[[BaseModel], Awaitable
                [BaseModel]]
```

- Inside of manually creating this tool class @function_tool can be used.

Tool Execution Flow

- Agent receives input → LLM call tool
- SDK validates input again input_schema
- calls handler() function → validate res using output_schema
- Tool output ← Loop Continue

Tool Parameters

- tool_choice = "auto" (LLM will choose)
- "required" (must use tool)
- "none" (disable tool calling)

Tool Loop And Limits

- run_limits - (no. of looping)
- tool_max_loops - (prevent infinite loop)

Tool Error Handling

- ToolExecutionError
- InputValidationError
- ModelBehaviourError

3) Markdown

Q: • why matters in Agent SDK
A: for replies, agent often uses it to:
- format msgs clearly
- highlight ans, link, code or list
- make info readable inside apps

GPT models output markdown by default.

Q: • what is markdown
A: lightweight markup language for formatting text

elements	Description	Uses
** bold **	highlights	• To highlight specific part eg Terms and Conditions
* italic *	emphasees	
` code `	inline code	• Agent ins, tool call result, code explan
[link](url)	clickable hyperlinks	• share sources, site refrences, link doc/tools
![image](url)	show images	
- list	bullet points	? • step by step guides, summaries, tool options
1.	number list	
> quote	add block quotes	• reply msgs, emphasize content, summarize prior steps
---	insert horizontal line	• separate sections
*** bolditalic ***	bold + italic	
"code block"	complete block of code	• Tutorial, log, script generation

Important Note:

• Don't use markdown in Agent Instructions and Console logs/JD

2) Prompt Engineering

a) Definition

- art of writing effective inputs for LLM to generate desired output.
- iterative – test, refine, repeat
- LLM predicts each word based on your prompt

  good prompt = better result

b) LLM Output Configuration

- To control how model behave and respond

Output Length

- control how long the response will be
- more tokens = more cost
- too short = incomplete answers
- too long = rambling

Temperature :

- controls creativity / randomness

  0 : deterministic

  1 : creative

  > 1 : very random

g

tive inputs
desired

repeat
word based

ter result

quation
model
spond

the response will be
re cost
ete answers
g

randomness

Top-k:

- k = no. of top choices to keep
- keep only top k (no.) of tokens and discard the rest

example:

model predicts -
(Cat, dog, Car, Corn, Cake, Candle)
top_k = 3
only cat, dog and car would be selected.

- fixed no.

Top-p:

- p = popularity
- p = 1 : all tokens included
- add words until popularity adds upto 90% (if top_p = 0.9).
- not a fixed no.

	Popularity	Total
cat	25%	25%
dog	20%	45%
cake	15%	60%
Corn	10%	70%
Cup	10%	80%
Candle	10%	90%

- now all 6 words would be picked.

c) Prompting Techniques (Basic)

1) Zero Shot
- direct simple question

2) 1 shot and few shot
- 1 or more examples in prompt

3) System
- sets global behaviour of model

e.g: choose right option from the given questions

4) Role
- assigning role to model
- you are travel guide

5) Contextual
- background info for understanding

Advance

1) Chain Of Thought
- ask the model to reason step by step
- active models internal reasoning
- improves accuracy
- reduces hallucination

Used for: math, logic, scheduling, multi agent routing, decision in SDK

ques (Basic)

question
not
amples in prompt

ou of model
option from
estions

to model
guide

for understanding

t
reason

nal reasoning

ion
ulling, multiagent routing decision in SDK

2) Tree Of Thought

- generate multiple ideas or reasoning path. then evaluate and select the best
- divergent thinking and decision making

Used for: brainstorming, Tradeoff analysis, comparing alternatives

3) Step Back

- think about category or intent of task before solving it.
- activates metareasoning
- avoids jumping directly to conclusion

Used for: classifying + responding, agent routing

4) Self Consistency:

- same question multiple times and choose most common answers.

Used for: exams, fact-checking, reasoning validation

5) React

- hybrid prompting strategy

steps :

- think or reason
- tool or API call
- think again and finalize response

(This is exactly how agent SDK works)

- separates thinking and doing
- prevents hallucination

d) Automatic Prompt Engineering

- Use LLM to generate multiple versions of your prompt
- automate process, save time, improve results.

e) Code Specific Prompting

1) Writing Code
2) Explaining Code
3) Translating Code
4) Debugging Code

tratégy

alize response
SDK works)
doing

eering

ple

e,

f) Multimodal Prompting
- combine text, img, code, audio etc

Used for: captionaning, document understanding or video analysis

g) Best Practices
- provide examples
- design with simplicity
- be specific about output
- prefer instructions over constraints
- use variables
- control token length
- avoid repeating label order in every example

h) 4.1 OpenAI cookBook

- GPT 4.1 is highly literal: follow instructions exactly
- GPT 4.1 excels in multi-agent or tool based chains
- internal tests show 20% performance gain with proper prompt

(i) System Prompt Essentials

- Persistence
  "stay active until user's query is fully resolved"
- Tool-Calling
  "if unsure, use tools; don't guess"
- Planning (Optional)
  "plan before calling function, react afterwards."

(ii) Prompt Induced Planning

- plan steps before answering or tool calling
- improves clarity, organize actions
- leads to better more accurate results
- In agent SDK, this step goes in ~~ins~~
  - instruction of agent
  - prompting before tool call

(iii) Tool Calling

- use official tool field in API
- each tool has clear name, well defined parameters, precise descriptions
- for complex tools, add examples in system prompt

(iv) Positioning

- for long document context
  - repeat instruction both at start and end
  - if only once, put before context.

(v) Agent Persona

- LLM can leak or misuse

Instruction based masking

"You are support bot. Never reveal internal ticket ID"

Prompt Boundary Delimeters

"only use whats inside this block:

[name: "Ali", issue: "not found"]