My First Taste of AI Agents

投稿人：Po-Ying Fu

OpenAI Agents SDK 簡介

什麼是 Agents SDK？

- 輕量、易用的 AI 代理應用開發套件
- Swarm 的生產就緒升級版
- 結合 Python，快速構建真實世界應用

核心組件

- **Agents**：具備指令與工具的 LLM 代理
- **Handoffs**：代理間任務交接
- **Guardrails**：輸入資料驗證防護
- **Tracing**：內建追蹤，視覺化與除錯代理流程

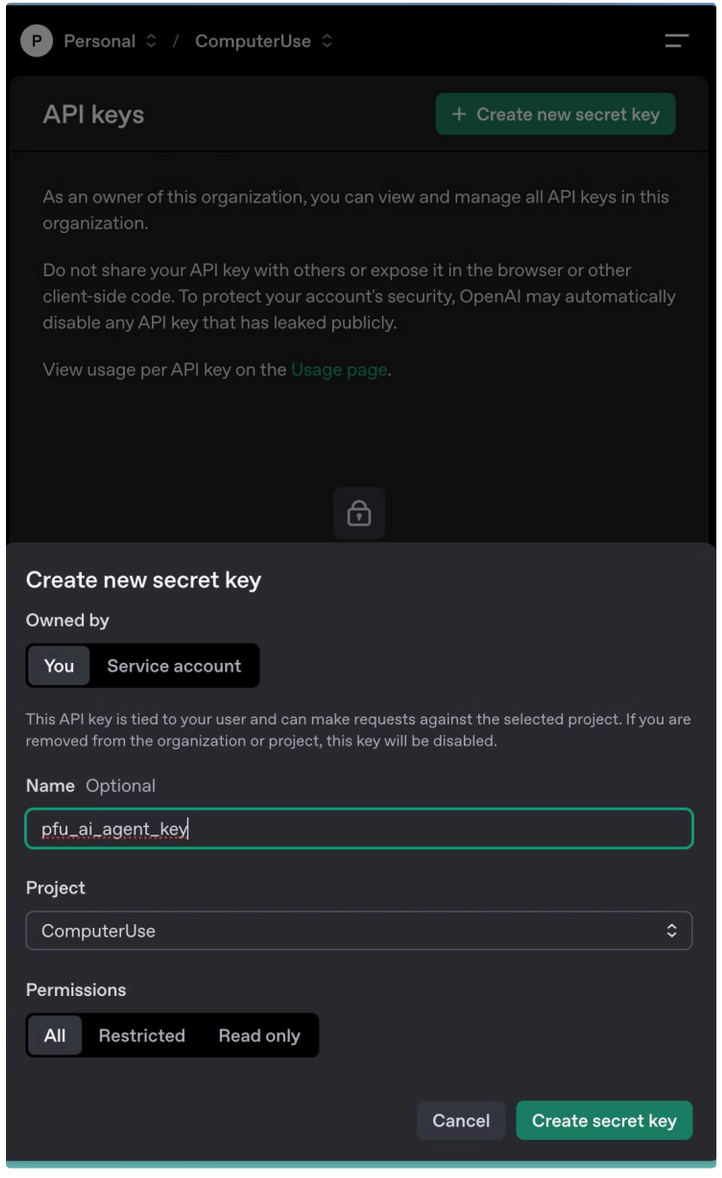
功能亮點

- 內建 **Agent Loop**：自動處理工具與 LLM 的互動循環
- **Python-first**：直接用 Python 串接代理與工具
- **Function Tools**：將 Python 函數變成代理工具
- **Guardrails**：即時阻斷不合格輸入
- **Handoffs**：輕鬆管理多代理協作

優勢

- 開箱即用，學習曲線平滑
- 可自訂化，靈活組合各類工具與代理
- 支援 OpenAI 生態系統的評估與微調

環境設定與建立 API Key



1. 建立專案與虛擬環境
2. 安裝 `openai-agents` 與 `dotenv` 套件

```
$ python -m venv .venv

$ source .venv/bin/activate

$ pip install --upgrade pip

$ pip install -r requirements.txt
```

參考文件：https://openai.github.io/openai-agents-python/

Basic Agent

- 使用 dotenv 儲存並使用 API Key
- 建立一個翻譯西班牙文的 Agent()
- Runner 執行 Agent
  - Runner.run() 非同步
  - Runner.run_sync()

```python
from agents import Agent, Runner
from agents import set_default_openai_key
from dotenv import load_dotenv
import os

load_dotenv()
openai_apikey = os.getenv("OPENAI_APIKEY")
set_default_openai_key(openai_apikey)

spanish_agent = Agent(
    name="Spanish agent",
    instructions="You translate the user's message to Spanish")

result = Runner.run_sync(spanish_agent, "Say 'Hello Wolrd' in Spanish")
print(result.final_output)
# Code within the code,
# Functions calling themselves,
# Infinite loop's dance.

```

```
PROBLEMS 3  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

(.venv) (base) ➜  ai_agent_test git:(main) ✗ python src/my_basic_agent.py
Di "Hola Mundo" en español.
```

Handoffs – 建立一個 Agent 來選擇不同功能的 Agents

```python
history_tutor_agent = Agent(
    name="History Tutor",
    handoff_description="Specialist agent for historical questions",
    instructions="You provide assistance with historical queries. "
    "Explain important events and context clearly.",
)

math_tutor_agent = Agent(
    name="Math Tutor",
    handoff_description="Specialist agent for math questions",
    instructions="You provide help with math problems. "
    "Explain your reasoning at each step and include examples",
)

# Define your handoffs
triage_agent = Agent(
    name="Triage Agent",
    instructions="You determine which agent to use based on the user's homework question",
    handoffs=[history_tutor_agent, math_tutor_agent]
)

# Run the agent orchestration
async def main():
    result = await Runner.run(triage_agent, "What is the capital of France?")
    print(result.final_output)
```

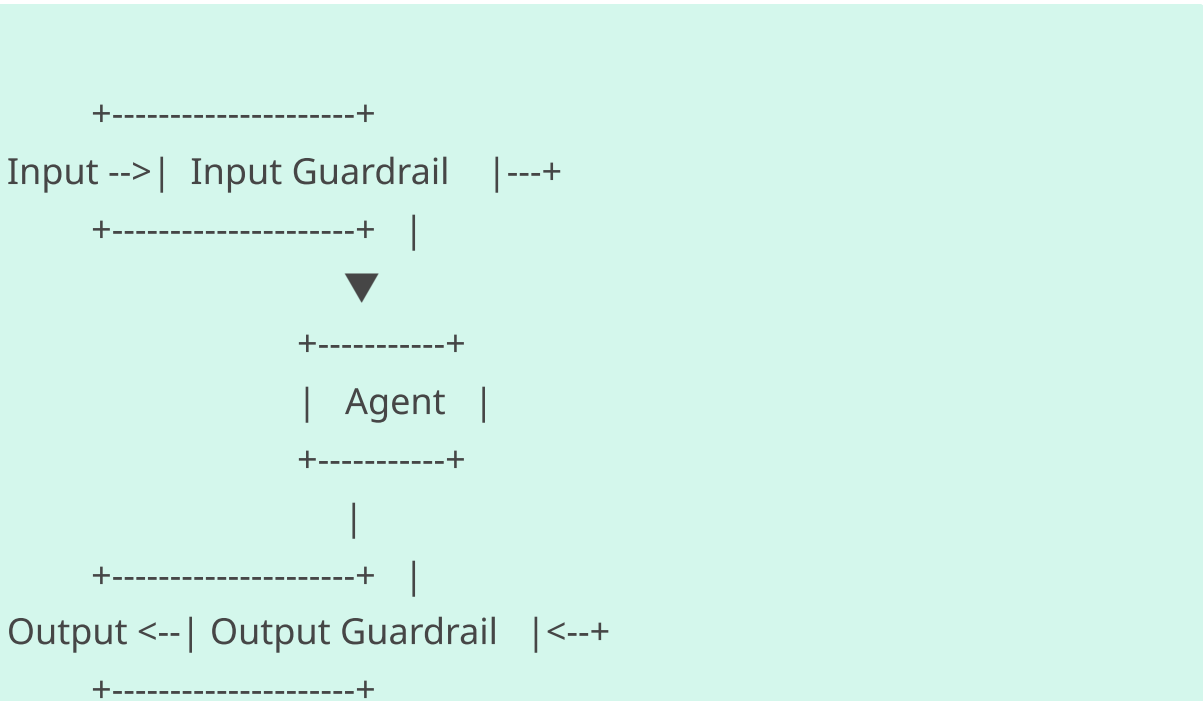
https://github.com/openai/openai-agents-python/blob/main/docs/quickstart.md#add-a-few-more-agents

OpenAI Agents SDK – Guardrails

什麼是 Guardrails？

- 代理流程中的「安全閘道」，用於驗證輸入與輸出
- 防止代理執行無效、危險或不當操作
- 可在「進入代理」或「離開代理」時進行驗證
- Guardrails 是設計綁定在 **個別 agent 實例** 上，因為不同 agent 通常有不同的安全規則或驗證邏輯。
- Handoff 發生後，parent agent 的 guardrails 不會自動應用到 child agent，需手動為每個 child agent 設定。

→ 可封裝共用 guardrails，統一自動套用



類型與架構

- **Input Guardrails**：驗證用戶請求是否合法
- **Output Guardrails**：驗證代理結果是否合規

Guardrails

Input Guardrails 三步驟

1. Input Guardrail 接收使用者輸入
2. 執行 Guardrail 函數
   a. Guardrail 函數執行並返回 `GuardrailFunctionOutput`
   b. 此結果被包裝成 `InputGuardrailResult`
3. 檢查 tripwire
   a. 檢查 tripwire_triggered 是否為 True。
   b. 若為 True，則觸發 `InputGuardrailTripwireTriggered` 異常，可用來回應使用者或終止流程。

📌 注意事項

- Input Guardrails 僅在**最先被呼叫的 agent** 中運行。
- 為什麼 guardrails 綁在 agent 而非 `Runner.run()`？
  → 因為不同的 agent 需要設計不同的 guardrails，這樣的綁定方式能提升程式碼的可讀性與可維護性。

Output Guardrails 三步驟

1. Output Guardrail 接收 Agent 完成後的輸出結果。
2. 執行 Guardrail 函數
   a. 執行自訂的 guardrail 函數，並返回 `GuardrailFunctionOutput`
   b. 該結果被包裝為 `OutputGuardrailResult`。
3. 檢查 tripwire
   a. 檢查 `agents.guardrail.GuardrailFunctionOutput.tripwire_triggered` 是否為 `True`。
   b. 若觸發 tripwire，則引發 `OutputGuardrailTripwireTriggered` 異常，允許開發者捕捉與處理。

📌 注意事項

- Output Guardrails 是用來驗證 **代理的最終輸出** 是否符合預期與安全標準

Add guardrails - Input Guardrail

```python
# Add an input guardrail agent
class HomeworkOutput(BaseModel):
    is_homework: bool
    reasoning: str

input_guardrail_agent = Agent(
    name="Guardrail check",
    instructions="Check if the user is asking about homework.",
    output_type=HomeworkOutput,
)

async def homework_guardrail(ctx, agent, input_data):
    result = await Runner.run(input_guardrail_agent, input_data, context=ctx.context)
    final_output = result.final_output_as(HomeworkOutput)
    return GuardrailFunctionOutput(
        output_info=final_output,
        tripwire_triggered=not final_output.is_homework,
    )
```

Add guardrails - Output Guardrail

```python
# Add an output guardrail agent
class ShortOutput(BaseModel):
    '''
    This is the guardrail's output type.
    '''
    is_short: bool
    reasoning: str


output_guardrail_agent = Agent(
    name="Output Guardrail check",
    instructions="Check if the response less than 50 words, return is_short=True or False otherwise.",
    output_type=ShortOutput,
)

@output_guardrail
async def response_guardrail(
    ctx: RunContextWrapper, agent: Agent, output: MessageOutput
) -> GuardrailFunctionOutput:
    result = await Runner.run(output_guardrail_agent, output.response, context=ctx.context)
    output_word_count = len(output.response.split())

    return GuardrailFunctionOutput(
        output_info = result.final_output,
        tripwire_triggered = output_word_count > 50 or not result.final_output.is_short,
    )
```

Add Guardrails to Agents

```python
# Add agents
math_tutor_agent = Agent(
    name="Math Tutor",
    handoff_description="Specialist agent for math questions",
    instructions="You provide help with math problems. "
    "Explain your reasoning at each step and include examples",
    output_guardrails=[response_guardrail],
    output_type=MessageOutput,
)

history_tutor_agent = Agent(
    name="History Tutor",
    handoff_description="Specialist agent for historical questions",
    instructions="You provide assistance with historical queries. "
    "Explain important events and context clearly.",
    output_guardrails=[response_guardrail],
    output_type=MessageOutput,
)

triage_agent = Agent(
    name="Triage Agent",
    instructions="You determine which agent to use based on the user's homework question",
    handoffs=[history_tutor_agent, math_tutor_agent],
    input_guardrails=[
        InputGuardrail(guardrail_function=homework_guardrail),
    ],
    output_guardrails=[response_guardrail],
    output_type=MessageOutput,
)
```

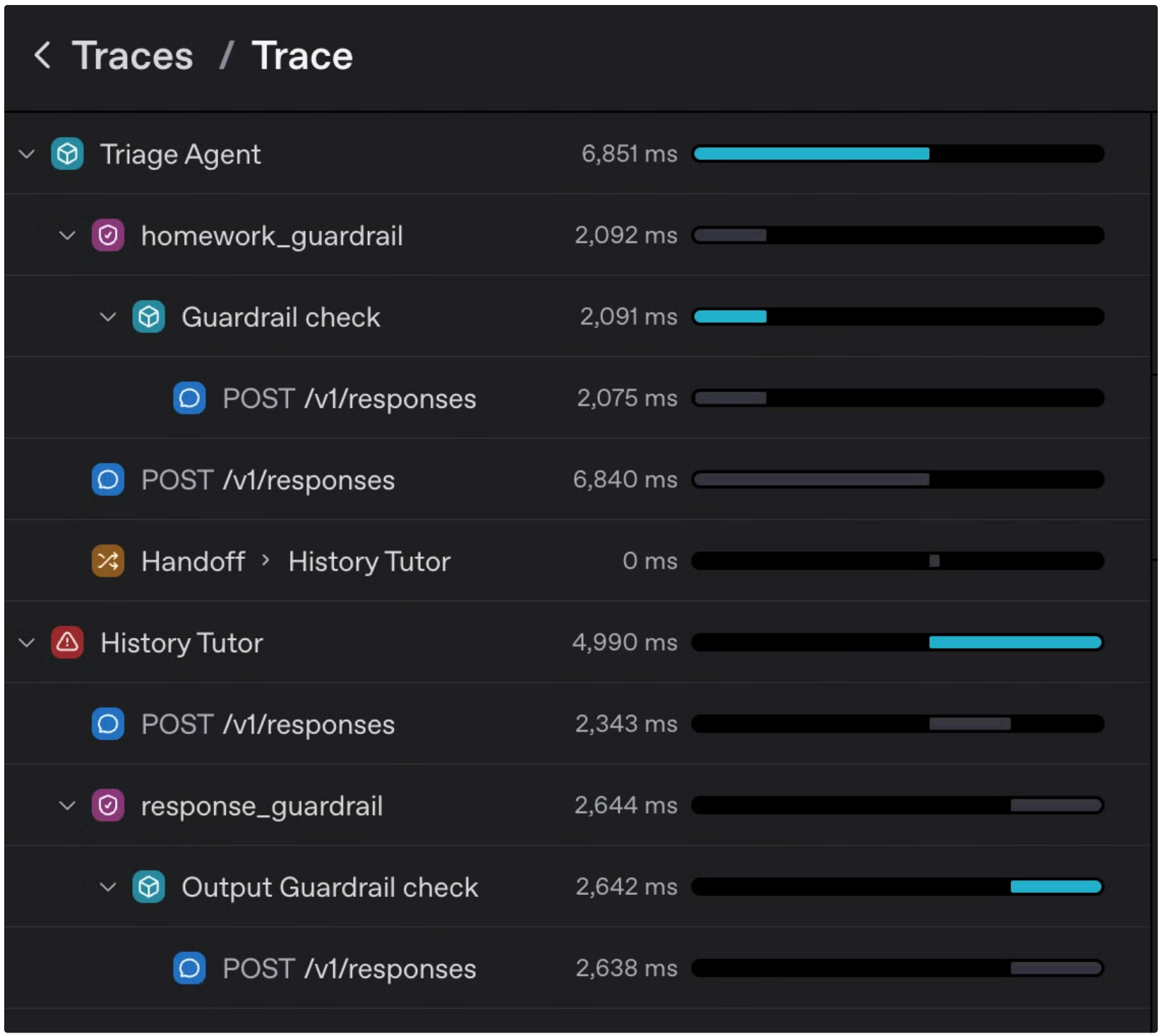
Run Agents with Guardrails

```python
async def main():
    try:
        result = await Runner.run(triage_agent, "who was the first president of the united states?")
        print(result.final_output)

        result = await Runner.run(triage_agent, "what is life")
        print(result.final_output)
    except InputGuardrailTripwireTriggered:
        print(">>> Input guardrail tripped! It's not homework.")
    except OutputGuardrailTripwireTriggered:
        print(">>> Output guardrail tripped! The output is too long.")

if __name__ == "__main__":
    asyncio.run(main())
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
(.venv) → ai_agent_test git:(main) ✗ python src/add_guardrail.py
>>> Output guardrail tripped! The output is too long.
```

Tracing – 追蹤 Guardrail 與 Handoff 順序



Agents as tools

```python
# Add the agents as tools
spanish_agent = Agent(
    name="Spanish agent",
    instructions="You translate the user's message to Spanish",
    output_type=MessageOutput,
)

chinese_agent = Agent(
    name="Chinese agent",
    instructions="You translate the user's message to Chinese",
    output_type=MessageOutput,
)

orchestrator_agent = Agent(
    name="orchestrator_agent",
    instructions=(
        "You are a translation agent. You use the tools given to you to translate."
        "If asked for multiple translations, you call the relevant tools."
    ),
    tools=[
        spanish_agent.as_tool(
            tool_name="translate_to_spanish",
            tool_description="Translate the user's message to Spanish",
        ),
        chinese_agent.as_tool(
            tool_name="translate_to_chinese",
            tool_description="Translate the user's message to Chinese",
        ),
    ],
    input_guardrails=[
        InputGuardrail(guardrail_function=translation_guardrail),
    ],
    output_guardrails=[response_guardrail],
    output_type=MessageOutput,
)
```

Run Orchestrator Agent

```python
async def main():
    try:
        result = await Runner.run(orchestrator_agent, input="Say 'Hello, how are you?' in Chinese.")
        print(result.final_output)
    except InputGuardrailTripwireTriggered:
        print(">>> Input guardrail tripped! It's not translation task.")
    except OutputGuardrailTripwireTriggered:
        print(">>> Output guardrail tripped! The output is too long.")

if __name__ == "__main__":
    asyncio.run(main())
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
(.venv) →  ai_agent_test git:(main) ✗ python src/Agents_as_tools.py
response='"Hello, how are you?" in Chinese is:\n\n你好，你怎么样？'
```

Tracing – Agent with tools + Guardrail