C4 Assignment - 503P/798S

Build a Custom ReAct Agent with Personas

1 Background & Objectives

In this assignment, you will extend your previous business chatbot project by transforming it into a **ReAct-style agent** — capable of **reasoning** and **acting** through **tool usage**.

You must implement your agent **inside one specific agent framework** (choose one):

- **LangGraph** (state-machine based workflows)
- **CrewAI** (multi-agent collaboration framework)
- **Autogen** (agent orchestration and dialogue-based reasoning)

⚠️ You may **not use prebuilt executors** like LangChain's AgentExecutor. Instead, you'll implement the ReAct loop **manually within your framework** (states/edges in LangGraph, crews in CrewAI, or agents in Autogen).

To deepen your experimentation:

- You will **create multiple agents with different personas** (e.g., Friendly Advisor, Strict Expert, Cautious Helper).
- You will **vary LLM configurations** (temperature, model, top-p, etc.).
- You will evaluate which agent + configuration worked best for your chosen use case.

Your final deliverable should function as a working assistant **and** include a reflection on how prompt style, persona, and configuration impacted performance.

If you need ideas:

- A fitness planner recommending routines
- A book advisor matching reader preferences
- A conference concierge suggesting sessions
- A legal assistant guiding document preparation

---

2 Requirements

2.1. Define Your Use Case

Create a fictional or real-world scenario for your agent. Some examples include:

- A course advisor for a university
- A home renovation consultant

- A travel planner
- A startup mentor

You must specify:

- **Agent goal** (what it helps with)
- **Target audience**
- **Tools it can use**
- **Constraints/limitations**

-> You may reuse the scenario from the previous assignment.

2.2. ReAct Loop

Write your own ReAct logic, including:

- A **system prompt** defining agent behavior and available tools
- **Tool descriptions** embedded in the system prompt or instructions
- ***Logic to detect when the model suggests an action (IMPORTANT)***
- Code to execute the tool, return the observation, and continue the loop
- Response parsing to complete the agent's final answer

For implementing the ReAct loop with different frameworks, you could:

- **LangGraph** → model the flow using **states, nodes, and conditional edges** that move from *thought* → *action* → *observation* → *answer*.
- **CrewAI** → design agents as **crew members with specific roles/personas**, coordinating their tool usage within the ReAct cycle.
- **Autogen** → create **multiple agents with distinct roles** and manage their reasoning and tool interactions through dialogue-driven loops.

Example ReAct Flow:

```
Thought: I need to look up location info.
Action: lookup_location("Paris")
Observation: Paris is the capital of France.
Thought: That helps. Now I can answer.
Answer: Paris is a great travel destination!
```

2.3. Test Different Personas & Configurations

You must:

- Define and test **at least two agent personas with different tasks** (e.g., friendly vs. expert tone)
- Experiment with prompt engineering:
    - With vs. without **Chain-of-Thought**
    - **Zero-shot** vs. **Few-shot**

    - Using **system prompts** vs. in-message instructions

- Vary LLM configurations:
    - Temperature, model type, top-p, max tokens, etc.

-> Log your experiments and **compare performance** across setups.

2.4. Reflection Questions (Include in Notebook or PDF)

- Which **persona** gave the most helpful or natural results?
- Which **prompt/config combination** performed best for your use case?
- How well did your agent **reason** and **use tools**?
- What were the biggest **challenges** in implementation?

---

3 Grading Criteria

Item	Points
Use-case summary & PDF write-up	10
ReAct logic manually implemented	35
Persona testing	25
Configuration testing	15
Creativity & clarity	15
