Test Task: Python Engineer - LLM Team
O.Foundation / O.XYZ – Sovereign AI & $OI Ecosystem

Overview

Your task is to design and implement a minimal but production-minded LLM "Agent Execution Core" in Python that can safely run tools, track state, and expose a simple API that an AI CEO / Operator could use inside the O Ecosystem. The focus is on clean Python architecture, robust reasoning/tooling abstractions, and readiness to plug into decentralized infra and AI-native products.

Goal

Build a Python service that:

- Accepts a natural-language task request and a list of available tools (with schemas).
- Uses an LLM to choose and sequence tools, maintaining a minimal agent state (memory of prior steps and results).
- Exposes a simple HTTP API that another system (e.g., an AI CEO orchestration layer) can call to execute tasks and retrieve structured results.

The task does not require integration with real onchain or decentralized infra, but the design should clearly be ready for extension into the O Ecosystem (e.g., onchain tools, governance tools, $OI accounting tools).

Requirements

Core Functionality

- Implement an Agent class in Python that:
    - Accepts: user goal, optional context, and a set of tool definitions (name, description, input schema).
    - Chooses which tool(s) to call and in what order using an LLM backend (can be OpenAI-style; mock or real is fine as long as it is abstracted).
    - Maintains a step-by-step execution trace: tool calls, arguments, results, and intermediate reasoning steps (you can redact the actual LLM "thoughts" but preserve structure).
    - Returns a final structured result object (status, output, trace).
- Implement at least 3 concrete tools:
    - WebSearchTool: given a query string, returns a mocked or real search result summary list.

    - MathTool: evaluates safe arithmetic expressions, with input validation and error handling.
    - GovernanceNoteTool: appends a short “governance note” (string) into a simple in-memory store keyed by “proposal_id”.

System / API

- Wrap the Agent in a minimal HTTP API (FastAPI or similar):
    - POST /run-task – body: { "goal": str, "context": optional, "tools": optional list of tool names } returns final result + trace.
    - GET /governance-notes/{proposal_id} – returns all notes for that proposal.
- The service should be runnable locally via a single uvicorn (or similar) command.

Architecture & Code Quality

- **Execution Infrastructure:** The agent core must be designed to be runnable via orchestration framework **Trigger.dev**. The service should expose clear execution entry points for job scheduling and reliability.
- Clean, modular Python architecture:
    - Separate modules for: agent core, tools, API layer, and LLM client abstraction.
    - Type hints and basic docstrings for public methods.
    - Clear error handling paths (e.g., tool call fails, LLM call fails, invalid input).
- **The LLM client must be abstracted behind an interface so it can later be swapped for an internal “Sovereign AI CEO” model. For this task, the client must use the Vercel AI SDK (v5 or later) as the underlying communication layer** with the LLM backend (e.g., OpenAI-compatible).
-

UX / Behavior Expectations

- The agent should:
    - Decide when to call tools vs. answer directly.
    - Be able to call multiple tools in sequence to achieve a goal (e.g., search + math + governance note).
    - Provide clear, machine-readable traces suitable for later analysis by an AI CEO layer.
- You may use any AI coding assistants to speed up development.

Deliverables

- GitHub repository containing:
    - Source code for the Python service (agent core, tools, API).
    - A short README that includes:
        - High-level architecture diagram or description.

        - How to run the service locally.
        - How to run 2–3 example tasks that demonstrate:
            - Single-tool usage.
            - Multi-step, multi-tool usage.
            - GovernanceNoteTool usage (e.g., adding notes to a “proposal”).
- Optional but strongly preferred:
    - A minimal Dockerfile and instructions to run via Docker.
    - A brief note on how you would extend this core to:
        - Integrate with decentralized infra (e.g., onchain tools).
        - Plug into an “AI-led organization” (e.g., periodic scheduled tasks or automated governance workflows).
- Access:
    - Share the GitHub repo with the recruiting contact provided.
    - Include any environment variable expectations (e.g., LLM API keys) in .env.example only, not committed secrets.

Constraints

- Timebox: Aim to complete within 24 hours from receiving the task.
- You may use any Python web framework (FastAPI preferred) and any LLM backend as long as the abstraction is clean.
- No need to overbuild UI; focus is on backend architecture, agent design, and clarity of reasoning traces aligned with an AI-led, sovereign-intelligence-oriented stack.