Proiect ASO 2025
Un agent AI ca administrator de sistem

1 Introducere

În acest proiect vom realiza un agent AI care acționează ca un administrator de sistem, răspunzând la întrebările utilizatorului despre sistemul administrat.

Vom folosi framework-ul ADK[1] (Agents Development Kit) pentru a implementa agentul inteligent. În acest framework putem defini un `LlmAgent`[2] specificând caracteristici cum ar fi modelul folosit (utilizați *litellm*[3] pentru a folosi modele locale), descriere, instrucțiuni sau instrumentele folosite. Tot în acest framework avem și `adk-web`, un server web minimal care ne permite să interacționăm cu agenții printr-o interfață de tip chat și să analizăm funcționarea acestora.

Agenții AI au nevoie de un Model lingvistic mare (LLM). Vom rula unul local, folosind Ollama[4]. Din lista de modele suportate, alegeți unul care se potrivește configurației hardware a calculatorului vostru și care suportă *tool-uri*[5]. Țineți cont că modelul ar trebui să "încapă" în întregime în memoria RAM a sistemului vostru. Dacă aveți suficientă memorie RAM, puteți încerca un model mai performant, cum ar fi `gpt-oss:20b` (14GB). Altfel, puteți rula un model mai mic, cum ar fi `llama3.2:3b` (2GB) sau `llama3.2:1b` (1.3GB). Sunteți încurajați să experimentați cu mai multe modele și să alegeți unul potrivit, atât din punct de vedere al "inteligenței" cât și a performanței. Nu se permite utilizarea modelelor externe (de exemplu ChatGPT, printr-o cheie pentru API).

Agenții AI au de obicei capabilitatea de a interacționa cu "lumea exterioară" folosind instrumente (*tools*). De exemplu, dacă întrebați un LLM cum este vremea afară, el nu ar avea de unde să știe răspunsul bazându-se doar pe datele pe care a fost antrenat. În schimb, ar putea apela un instrument extern (de exemplu *OpenWeatherMap*) de unde să preia informații actualizate pe care să le integreze în răspuns. Unul din standardele din domeniu pentru expunerea de informații și instrumente către agenți AI este MCP[6]. Agentul AI se poate conecta la un server MCP, poate examina *tool*-urile disponibile și le poate apela pentru a-și extinde contextul (informațiile disponibile). Pentru a implementa

[1] `https://google.github.io/adk-docs/`

[2] `https://google.github.io/adk-docs/agents/llm-agents/`

[3] `https://google.github.io/adk-docs/agents/models/#using-cloud-proprietary-models-via-litellm`

[4] `https://ollama.com/`

[5] `https://ollama.com/search?c=tools`

[6] `https://modelcontextprotocol.io/`

un server MCP în Python se recomandă framework-ul FastMCP[7].

2 Cerințe

2.1 Etapa 1 – agentul inițial (3.5p)

- Instalați Ollama și alegeți un model care să poată rula pe sistemul vostru.
- Implementați un server MCP care să permită gestionarea unui director dat. Ar trebui să expuneți următoarele *tool*-uri:
  - `get_file_content(file_path: str) -> str`
  - `list_directory(dir_path: str) -> list[str]`

  Utilizați nume explicite, adnotați parametrii și adăugați un docstring explicit pentru fiecare *tool* expus, deoarece aceste informații vor ajunge să fie folosite de către LLM.

  Bonus: expuneți și alte *tool*-uri care să interacționeze cu alte componente ale sistemului, cum ar fi procesele sau care să ofere și alte tipuri de interacțiunui cu sistemul de fișiere.
- Creați un agent care să aibă rolul de administrator de sistem și care să aibă acces la serverul MCP (în această etapă puteți folosi *stdio* ca protocol de transport).
- Testați agentul creat, cerând informații despre fișierele și subfolderele din folderul administrat.

2.2 Etapa 2 – dockerizare (3.5p)

În această etapă vom separa componentele sistemului în mai multe containere Docker.

- Creați o imagine Docker care să ruleze Ollama împreună cu modelul ales la etapa anterioară.
- Creați o imagine Docker care să ruleze serverul MCP, folosind *HTTP streaming* ca protocol de transport. Bonus: adăugați și autentificare acestui server, astfel încât doar agenții care dețin o cheie de acces să poată folosi instrumentele de pe server.
- Creați o imagine docker în care să ruleze `adk-web`, împreună cu agentul creat la etapa anterioară.
- Folosiți *docker-compose* pentru a rula containerele aferente celor trei imagini.

[7] `https://gofastmcp.com/`

2.3 Etapa 3 – securitate (3p)

În această etapă trebuie să adăugați elemente de securitate agentului vostru.

- În sistemul de fișiere administrat trebuie să existe un fișier numit `flag.txt`, format dintr-un cuvânt sau o secvență de două cuvinte alăturate, scrise cu litere mari, în limba engleză. Dimensiunea fișierului nu trebuie să depășească 15 octeți.
- Agentul trebuie să fie capabil să verifice dacă utilizatorul știe conținutul acestui fișier. De exemplu dacă utilizatorul întreabă *"Does the content of flag.txt is REDAPPLE?"*, agentul ar trebui să răspundă afirmativ sau negativ.
- Agentul nu trebuie să divulge conținutul fișierului `flag.txt` (dar trebuie să fie în continuare capabil să afișeze conținutul altor fișiere). De exemplu dacă utilizatorul întreabă *"What is the content of flag.txt?"*, agentul ar trebui să refuze să răspundă (sau să dea un răspuns evaziv).

Puteți folosi orice tehnici pentru asigurarea securității, cât timp agentul vostru are funcționalitățile de mai sus. Se recomandă să studiați conceptele de securitate pentru agenți inteligenți, cum ar fi *AI guardrails*[8].

La finalul acestei etape se va organiza un campionat, în care studenții care au reușit să finalizeze proiectul vor încerca să afle *flag*-ul agenților colegilor lor. Câștigătorii vor fi recompensați cu bonusuri.

3 Livrabile

La fiecare etapă se va demonstra funcționalitatea cerințelor.

Tot pentru fiecare etapă se va scrie o documentație, în care se va descrie soluția aleasă. Documentația trebuie scrisă cât mai explicit, astfel încât cineva care o citește să poată reimplementa soluția.

Acolo unde se cer script-uri, acestea se vor urca pe Moodle, împreună cu documentația.

3.1 Termene limită

- Etapa 1: săptămâna 5
- Etapa 2: săptămâna 9
- Etapa 3: săptămâna 13

[8] `https://google.github.io/adk-docs/safety/`