

Asistent AI pentru materia ASC

1 Introducere și scopul proiectului

Proiectul realizat are ca scop dezvoltarea unui asistent AI educațional capabil să răspundă la întrebări legate de materia Arhitectura Sistemelor de Calcul (ASC), utilizând conținutul cursurilor de la facultate. Aplicația se bazează pe un model de inteligență artificială pre-antrenat (LLaMA 3.1, accesat prin Ollama) și folosește o bază de date vectorială construită pe baza cursurilor în format PDF.

Scopul principal este de a sprijini studenții în învățarea individuală, permițându-le să pună întrebări în limbaj natural și să primească răspunsuri precise, extrase din materialele oficiale ale cursului.

2 Descrierea și analiza setului de date (Small Data)

Pentru antrenarea și testarea sistemului a fost utilizat un set de fișiere PDF (peste 30), reprezentând cursurile și materialele de laborator de la disciplina ASC, stocate în directorul DOCS.

Sursa datelor

Toate materialele provin din resurse oferite de facultate, destinate exclusiv uzului didactic. Nu conțin date personale sau informații sensibile.

Structura datelor

- **Format:** PDF (text și formule)
- **Domeniu:** Arhitectura Sistemelor de Calcul
- **Număr total documente:** 32
- **Mărime totală:** aprox. 30–50 MB
- **Limbă:** Română/Engleză

Preprocesare și împărțire

Procesul este gestionat de scriptul `Loader.py`. Documentele PDF au fost încărcate folosind biblioteca `langchain_community.document_loaders` (clasa `PyPDFLoader`) și apoi împărțite în fragmente (chunks) de 1000 de caractere, cu o suprapunere de 150 caractere, utilizând `RecursiveCharacterTextSplitter`.

Această etapă este esențială pentru ca sistemul să poată căuta rapid și precis bucăți de text relevante din cursuri.

3 Descrierea algoritmului inteligent (AI)

Logica centrală a sistemului este definită în `rag_core.py`.

3.1 Tipul algoritmului

Proiectul folosește o abordare de tip RAG – Retrieval-Augmented Generation. Această metodă combină două componente:

- **Retrieval (Regăsire de informații)** – caută fragmente relevante în baza de date vectorială.
- **Generation (Generare)** – modelul AI (LLaMA 3.1) formulează un răspuns coerent folosind doar contextul extras.

3.2 Modelul de limbaj folosit

Modelul ales este `llama3.1`, accesat prin framework-ul Ollama. Este folosit atât pentru generarea răspunsurilor, cât și pentru generarea de embeddings.

3.3 Embeddings și baza de date vectorială

Pentru reprezentarea semantică a textelor s-a folosit componenta `OllamaEmbeddings` (`model="llama3.1"`). Acești vectori sunt stocați într-o bază de date `Chroma`, salvată local în directorul `chroma_db`.

3.4 Prompting și Multi-Query Retriever

Pentru a îmbunătăți calitatea contextului regăsit, sistemul nu folosește un retriever simplu.

- **Prompt Template:** S-a definit un *system prompt* specific (`PROMPT_TEMPLATE`) care instruieste modelul să acționeze ca un expert, să răspundă concis, în română, și să se bazeze exclusiv pe contextul furnizat.
- **MultiQueryRetriever:** În loc să caute direct întrebarea utilizatorului, sistemul folosește un LLM (`Ollama` cu `temperature=0.2`) pentru a genera mai multe variante ale întrebării originale. Căutarea se face pentru toate aceste variante, crescând șansa de a găsi documente relevante.

4 Arhitectura și metodologie experimentală

4.1 Structura generală a aplicației

Aplicația este compusă din trei module principale:

- `Loader.py` – Script rulat o singură dată pentru a citi PDF-urile din `DOCS2`, a le fragmenta și a construi baza de date vectorială (`Chroma DB`).
- `rag_core.py` – Modulul central care definește funcția `initialize_rag_system()`, construiește lanțul RAG, încarcă modelele și configurează retriever-ul.

- `Learner.py` – Aplicația principală. O interfață grafică (GUI) bazată pe `Tkinter` care permite interacțiunea utilizatorului cu modelul. Folosește `threading` pentru a nu bloca interfața în timpul generării răspunsului.
- `eval.py` – Script separat pentru evaluarea cantitativă a performanței modelului pe un set de întrebări predefinite.

4.2 Fluxul de lucru (RAG)

1. Se încarcă baza de date vectorială `Chroma` și modelul de embeddings.
2. Se inițializează modelul LLM `llama3.1` (prin `Ollama`).
3. Se configurează un `MultiQueryRetriever` care folosește LLM-ul pentru a genera variante de întrebări.
4. Se construiește lanțul RAG (retriever + prompt + LLM + output parser).
5. Utilizatorul introduce o întrebare în interfața `Learner.py`.
6. `MultiQueryRetriever` generează variante și extrage cele mai relevante fragmente (top $k = 8$).
7. Fragmentele sunt injectate în `PROMPT_TEMPLATE` ca și context.
8. Modelul LLM (cu `temperature=0.4`) generează un răspuns pe baza contextului și îl afișează în fereastra de chat.

4.3 Parametri importanți

- **Model LLM:** `llama3.1`
- **Embedding model:** `OllamaEmbeddings(llama3.1)`
- **Bază de date:** `Chroma` (din `chroma_db`)
- **Retriever:** `MultiQueryRetriever`
- **Număr fragmente returnate:** $k = 8$
- **Temperatură LLM (răspuns):** `0.4`
- **Temperatură LLM (query-gen):** `0.2`
- **Dimensiune chunk text:** 1000 caractere, overlap 150

5 Evaluare și Metodologie Implementată

Evaluarea este implementată activ în scriptul `eval.py`.

5.1 Setul de date pentru evaluare

S-a definit o listă de 20 de întrebări-cheie din materia ASC, numită `EVAL_SET`. Fiecare întrebare din set este însoțită de o listă de cuvinte cheie esențiale (`expected_keywords`) care ar trebui să apară într-un răspuns corect.

Exemplu de item din `EVAL_SET`:

```
{
  "question": "Ce reprezintă octetul ModR/M?",
  "expected_keywords": ["mod", "registru", "memorie",
                        "operanzi", "offset", "modrm"]
}
```

5.2 Metodologia de testare

Scriptul `eval.py` parcurge automat fiecare întrebare din `EVAL_SET`:

1. Trimite întrebarea către sistemul RAG (inițializat prin `rag_core.py`).
2. Primește răspunsul generat de AI.
3. Compară (case-insensitive) răspunsul AI cu lista `expected_keywords`.
4. Calculează un prag de succes: un răspuns este considerat "Corect" dacă conține cel puțin jumătate (rotunjit în sus) din numărul total de cuvinte cheie așteptate.
5. Afișează la consolă statusul (Corect/Gresit) și cuvintele cheie lipsă (dacă e cazul).

5.3 Metrica de acuratețe

La finalul rulării, scriptul calculează o metrică de acuratețe generală bazată pe numărul de răspunsuri care au îndeplinit pragul de cuvinte cheie.

$$\text{Acurate\cetea} = \frac{\text{Num\c4r r\c4spunsuri considerate corecte}}{\text{Num\c4r total \c4ntreb\c4ri (20)}} \times 100\%$$

Această metodă, deși nu evaluează corectitudinea semantică perfectă, oferă o măsură cantitativă rapidă a relevanței contextului extras și a capacității modelului de a formula răspunsuri pe baza lui.

6 Rezultate și discuții

Scriptul de evaluare `eval.py` permite testarea rapidă a modificărilor aduse sistemului (ex. schimbarea parametrilor, a modelului, a prompt-ului).

Limitări

- Evaluarea bazată pe cuvinte cheie este simplistă. Un răspuns poate fi semantic corect, dar poate folosi sinonime neincluse în lista `expected_keywords`.
- Calitatea depinde de conținutul cursurilor și de acuratețea textului extras din PDF.

Avantaje

- Sistemul este local, fără dependență de cloud (folosind Ollama).
- `MultiQueryRetriever` oferă o regăsire de context mai robustă.
- Evaluarea automată (`eval.py`) permite *regression testing* și optimizare iterativă.

În viitor, se pot adăuga:

- O interfață web modernă (ex. FastAPI + React/Streamlit).
- Salvarea istoricului conversațiilor.
- Un mecanism de evaluare mai complex, posibil bazat pe un LLM ca judecător (LLM-as-a-judge).

7 Biblioteci folosite

- `tkinter` – interfața grafică
- `threading`, `sys`, `os` – utilitare de sistem
- `langchain_classic.retrievers` – (pentru `MultiQueryRetriever`)
- `langchain_core` – (pentru `StrOutputParser`, `RunnablePassthrough`)
- `langchain_ollama` – (pentru `OllamaEmbeddings`)
- `langchain_community` – (pentru `Ollama`, `PyPDFLoader`, `DirectoryLoader`)
- `langchain_chroma` – (pentru `Chroma DB`)
- `langchain_text_splitters` – (pentru `RecursiveCharacterTextSplitter`)

8 Referințe

- Meta AI – LLaMA 3.1 Model Card
- LangChain Documentation – <https://python.langchain.com/>
- Ollama – <https://ollama.ai>
- ChromaDB – <https://docs.trychroma.com>