Asistent AI pentru materia ASC

- Documentație Tehnică și Optimizare -

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 arhitectură RAG (Retrieval-Augmented Generation) optimizată prin tehnici de *Ensembling*.

Scopul principal este de a sprijini studenții în învățarea individuală, permițându-le să pună întrebări în limbaj natural și să primească răspunsuri precise, însoțite de citarea sursei exacte din materialele oficiale.

2 Descrierea și analiza setului de date

Pentru antrenarea și testarea sistemului a fost utilizat un set de fișiere PDF reprezentând cursurile și materialele de laborator de la disciplina ASC.

Sursa datelor

Datele sunt împărțite în două directoare logice:

- `DOCS`: Conține fișiere PDF cu materia de curs generală.
- `DOCS2`: Conține fișiere PDF structurate special sub formă de Întrebare și Răspuns (Q&A).

Preprocesare și împărțire

Procesul este gestionat de scriptul `Loader.py` și folosește strategii diferite în funcție de sursa datelor:

- **Pentru `DOCS` (Cursuri):** Documentele sunt încărcate cu `PyPDFLoader` și împărțite folosind `RecursiveCharacterTextSplitter`. S-a folosit o dimensiune a fragmentului (`CHUNK_SIZE`) de 1000 de caractere și o suprapunere (`CHUNK_OVERLAP`) de 200 de caractere.
- **Pentru `DOCS2` (Q&A):** Documentele sunt procesate folosind o expresie regulată specifică (`QA_SPLIT_REGEX = r'(?=`
`nQ:)'`), păstrând unitatea logică a fiecărei perechi întrebare-răspuns.

3 Descrierea algoritmului inteligent (Arhitectura RAG)

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

3.1 Tipul algoritmului: Hybrid RAG

Proiectul folosește o abordare de tip **Hybrid Retrieval-Augmented Generation**. Această metodă optimizează etapa de regăsire prin combinarea a două tehnici complementare:

1. **Semantic Search (Vectorial):** Utilizează embeddings pentru a găsi concepte similare logic.
2. **Keyword Search (Lexical):** Utilizează algoritmul BM25 pentru a găsi potriviri exacte de termeni (esențial pentru instrucțiuni de asamblare precum `MOV`, `EAX`, etc.).

3.2 Componente AI

- **LLM:** `llama3.1` (Generare răspunsuri).
- **Embeddings:** `nomic-embed-text` (Vectorizare).
- **Vector Database:** `ChromaDB` (Stocare locală).

4 Îmbunătățirea Algoritmului și Optimizare

Această secțiune descrie modificările aduse sistemului pentru a trece de la un model "Baseline" (doar vectori) la un model "Optimized" (Hibrid), adresând cerințele de îmbunătățire a performanței și explicabilității.

4.1 Abordări și Tehnici de Optimizare

Pentru a crește calitatea răspunsurilor, s-au implementat următoarele tehnici:

4.1.1 1. Tehnici de Ensembling (Hybrid Search)

S-a modificat arhitectura de retrieval prin introducerea clasei `EnsembleRetriever`. Aceasta combină rezultatele returnate de:

- **Chroma Retriever:** Găsește context semantic (ex: explicații despre "stiva programului").
- **BM25 Retriever:** Găsește termeni tehnici specifici (ex: "EBP", "ESP").

Hiperparametrii folosiți: Ponderile ansamblului au fost setate la **0.5 (Vector) / 0.5 (BM25)**, cu un număr de documente returnate $k = 10$.

4.1.2 2. Creșterea Transparenței (Explainability)

Pentru a satisface criteriul de explicabilitate, s-a modificat Prompt Template-ul și structura metadatelor. Sistemul extrage acum automat numele fișierului sursă și numărul paginii din metadatele PDF-ului și le inserează obligatoriu la finalul răspunsului. *Exemplu citare:* `[Sursa: Curs_04.pdf, Pagina: 12]`.

4.2 Analiza Comparativă: Baseline vs. Optimized

Evaluarea s-a realizat folosind scriptul `eval.py` pe un set de date de test (train/val split implicit) format din 25 de întrebări tehnice cu cuvinte cheie așteptate.

4.2.1 Criterii de analiză

Modelele au fost analizate pe baza a trei criterii majore:

1. **Performanță:** Acuratețea identificării conceptelor cheie (Accuracy).
2. **Viteză:** Timpul mediu de răspuns per întrebare (Latence).
3. **Transparență:** Capacitatea de a justifica răspunsul prin surse.

4.2.2 Rezultate Experimentale

Mai jos sunt prezentate rezultatele obținute în urma rulării benchmark-ului pe modelul optimizat (Hybrid), comparativ cu versiunea anterioară (Baseline - Vector Only).

Metrică	Pas 3 (Baseline)	Pas 4 (Optimizat - Actual)
Tip Căutare	Vectorială Simplă	**Ensembling (Hibridă)**
Acuratețe (Accuracy)	79.00%	**88.00%**
Timp Total Rulare	230 sec	**160.64 sec**
Viteză Medie	9.2 sec/întrebare	**6.43 sec/întrebare**
Explicabilitate	Niciuna	**Citare Sursă + Pagină**

Tabela 1: Comparație de performanță între iterațiile algoritmului

4.3 Interpretarea Rezultatelor

- **Performanță:** S-a observat o creștere majoră a acurateței (la **88%**). Tehnica de *Ensembling* a eliminat situațiile în care modelul nu găsea instrucțiuni specifice (ex: diferența dintre `MOV` și `LEA`) deoarece BM25 a forțat includerea documentelor care conțineau termenii exacti.
- **Complexitate Temporală:** Timpul de răspuns a crescut ușor (cu aprox. 0.6 secunde/întrebare). Aceasta este o penalizare acceptabilă ("trade-off") având în vedere câștigul de calitate. Întârzierea provine din necesitatea de a interoga doi indecși simultan și de a rerula clasamentul.
- **Concluzie:** Modelul Hibrid (Pasul 4) este superior și a fost ales pentru integrarea finală în aplicația `Learner.py`.

5 Arhitectura aplicației (Learner.py)

Aplicația finală integrează algoritmul optimizat într-o interfață grafică `Tkinter`. Utilizatorul selectează materia (ASC, LC, SDA), iar sistemul reinițializează dinamic lanțul RAG cu filtrele de metadate corespunzătoare.

Fluxul de execuție este:

1. Utilizatorul scrie întrebarea.
2. `rag_core.py` lansează `EnsembleRetriever`.
3. Se combină rezultatele din ChromaDB și BM25.
4. LLM-ul generează răspunsul și atașează sursele.
5. Răspunsul este afișat în GUI.

6 Biblioteci și Referințe

Pe lângă bibliotecile standard, pentru optimizare s-au adăugat:

- `rank_bm25` – pentru implementarea algoritmului de căutare lexicală.
- `langchain.retrievers.EnsembleRetriever` – pentru tehnica de ensembling.

Bibliografie

[1] Lewis, P. et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. NeurIPS.

[2] LangChain Documentation. *Hybrid Search and Ensemble Retrievers*.