Dokumentáció – Agentic RAG Prototípus

1. Feladat és cél

A feladat célja egy Agentic RAG (Retrieval-Augmented Generation) alapú chatbot prototípus fejlesztése, amely:

- demonstrálja az agentic működést: autonóm döntéshozatalt, részfeladatok lebontását, visszacsatolást, adaptív működést,
- felhasznál külső tudásforrásokat (PDF dokumentumok),
- bemutatható notebook formájában működik,
- nem használ fizetős API-kat (helyi GGUF modellel fut, `llama-cpp-python`).

A cél nem egy éles rendszer, hanem egy működő prototípus, amely bemutatja az agentikus logikát és a tervezési döntések átgondoltságát.

---

2. Architektúra áttekintés

A rendszer LangGraph alapú gráfban működik, amely az agentikus logikát irányítja.

Fő komponensek:

1. **Planner**
   - Feladata a komplex kérdések részfeladatokra bontása (heurisztikus szabályok).
   - Biztosítja a multi-step reasoning-et.
2. **Retriever (FAISS + Embedding)**
   - A PDF-ekből kinyert chunkokat FAISS indexben tároljuk.
   - Embedding modell: `sentence-transformers/all-MiniLM-L6-v2` (ha nem elérhető, fallback determinisztikus embedding).
   - `top_k=12` dokumentumot keres.
3. **Reranker**
   - CrossEncoder (`cross-encoder/ms-marco-MiniLM-L-6-v2`) finomítja a sorrendet.
   - Ha nem elérhető, fallback az eredeti vektoralapú sorrend.
4. **Synthesize (LLM válasz)**
   - Prompt sablon + kontextus chunkok + kérdés.
   - LLM: `llama-cpp-python` helyi GGUF modellel (`mistral-7b-instruct-v0.2.Q4_K_M.gguf`).
   - Fallback: DummyLLM.
5. **Critic (értékelő)**
   - Groundedness score (áthatás alapú 0–1 között).
   - Eredmény alapján döntés:
     - `done` → válasz elfogadva
     - `next_task` → következő részfeladat
     - `refine` → újrapróbálkozás

6. **LangGraph Controller**
   - A pipeline komponenseit láncolja.
   - Critic döntése alapján adaptív elágazásokat irányít.
   - Iterációs limit: `max_iter=2`.

---

3. Technológiai döntések

- **Python + Notebook**: érthető, reprodukálható, bemutatható.
- **LangGraph**: agentikus működés megvalósítására.
- **FAISS**: gyors, skálázható keresés.
- **Sentence-Transformers**: embedding alap.
- **CrossEncoder**: finom relevancia rangsorolás (opcionális).
- **llama-cpp-python**: helyi LLM futtatás, nincs API-kulcs szükséglet.
- **Fallback mechanizmusok**: minden körülmények között bemutatható prototípus.

---

4. Teljesítmény és bottleneck elemzés

A notebook mér minden komponensnél időt, majd Pandas táblázatban és matplotlib grafikonon összesíti.

Jelenlegi tapasztalatok:

- **Leglassabb lépés**: LLM válasz generálás.
- **Másodlagos bottleneck**: embedding számítás nagy korpusznál.
- **Gyors komponensek**: Planner és Critic.

---

5. Tesztelési stratégia

A notebookban smoke tesztek találhatók: - FAISS index létrejött-e, - Planner ad-e részfeladatot, - Retrieve működik-e, - Agent választ ad-e, - Critic score helyes tartományban van-e.

Javasolt további tesztek:

- **Integrációs teszt** több PDF + több kérdés.
- **Stressz teszt** nagy dokumentumkorpuszra.
- **Ground truth validáció** ismert válaszokkal.

---

6. Notebook prezentálhatósága

A notebook tartalmaz: - Markdown magyarázatokat minden modulhoz, - Példakérdéseket és válaszokat, - Bottleneck grafikonokat, - Teszt szekciót.

Ezáltal a rendszer önmagában bemutatható, prezentáció nélkül is.

---

7. További fejlesztési lehetőségek

- **Komplexebb Planner** (pl. neurális feladatbontás).
- **Okosabb Critic** (LLM-alapú értékelés, több szempont).
- **Memória modul** (korábbi válaszok felhasználása).
- **Skálázhatóság** (GPU gyorsítás, több PDF).
- **Webes interfész** a könnyebb bemutatáshoz.