JACQUIER Jean-Emmanuel
BUT Info 2
R.15 IA Introduction aux modèles

Rapport – Mini-projet RAG sur les examens

1. Objectif du projet

Ce projet a pour but de concevoir un petit système **RAG** (*Retrieval-Augmented Generation*) capable de répondre aux questions des étudiants concernant l'organisation des examens, à partir d'un ensemble limité de documents situés dans le dossier `data/`.
L'idée est de comparer deux approches :
- un mode **RAG**, où le modèle s'appuie uniquement sur les informations récupérées dans les documents ;
- un mode **no-RAG**, où le modèle répond sans aucun contexte.

Le système doit s'assurer que toutes les réponses s'appuient exclusivement sur les textes du corpus, tout en citant leurs sources et en indiquant clairement quand une information n'est pas disponible.

2. Données et ingestion

Le dossier `data/` contient quatre fichiers textuels : une FAQ, le règlement des examens, les consignes de rattrapage et les directives de surveillance.
Le script `src/ingest.py` lit ces documents, les découpe en *chunks* de taille fixe avec recouvrement, puis ajoute à chacun des métadonnées (`doc_id`, `chunk_id`, `source`). Ces segments serviront de base pour l'indexation.

3. Indexation et recherche

Le script `src/rag_chat.py` propose une fonction `build_index()` qui :
- charge les *chunks* créés par `ingest.py`,
- calcule un *embedding* pour chacun à l'aide du modèle `nomic-embed-text`,
- puis enregistre ces vecteurs dans un index FAISS (`faiss.IndexFlatL2`) sauvegardé sous le nom `faiss.index`.

Les métadonnées et textes originaux sont parallèlement stockés dans `metadata.pkl`.
La fonction `search_topk(question, k)` encode une question donnée, interroge l'index FAISS et renvoie les `k` passages les plus proches, accompagnés de leur score et de leur source.
Enfin, `format_context(chunks)` assemble ces extraits en un contexte clair, avec des citations explicites du type `[source:chunk_id]`.

4. Génération : modes no-RAG et RAG

Le mode **no-RAG** repose sur la fonction `llm_no_rag(question)`, qui envoie directement la question au modèle `llama3:latest` via une API locale, sans ajouter de contexte.
Le mode **RAG**, quant à lui, utilise `llm_rag_answer(question, context)` : un prompt système impose alors au modèle de s'appuyer uniquement sur les extraits fournis et de répondre « Je ne sais pas. » si l'information est absente, tout en citant systématiquement les sources.

Une boucle interactive `rag_chat_loop()` permet d'enchaîner ces étapes : chargement de l'index, récupération des passages pertinents, génération de la réponse et affichage du résultat avec citations.

5. Tests et comparaison entre les deux modes

Le fichier `tests/test_questions.py` regroupe six questions types (retard, absence, calculatrice, résultats, justificatif global, examen en ligne).
Le script `run_comparison()` réalise, pour chacune d'elles :
- une réponse **no-RAG**, puis une réponse **RAG** ;
- un tableau comparatif au format Markdown présentant les deux textes et les sources citées ;
- l'export du tout dans `comparaison_noRAG_vs_RAG.md`.

Les résultats montrent que les réponses no-RAG restent souvent vagues ou inventent des détails, tandis que le mode RAG se base réellement sur les documents du corpus (FAQ, règlement, etc.) et indique clairement quand il ne trouve aucune information pertinente.

6. Utilisation et reproductibilité

Le `README.md` décrit l'installation des dépendances (`faiss-cpu`, `numpy`, `requests`, ainsi que les modèles `nomic-embed-text` et `llama3` via Ollama), puis la procédure complète :
1. Créer un environnement virtuel Python et installer les librairies nécessaires.
2. Construire l'index à l'aide de `build_index()`.
3. Lancer les tests comparatifs avec `python -m tests.test_questions`.
4. Interagir librement avec le chatbot via `python -m tests.own_questions`.

L'ensemble du projet est donc entièrement reproductible : en partant du dossier `TP LLM + RAG`, on peut reconstruire l'index, exécuter la comparaison et poser ses propres questions au système.