Jade

Le Brouster

02/02/2026

BUT SD

Création d’un chatbot permettant de discuter avec votre jumeau virtuel qui connait tout de votre profil professionnel.

Qu’est-ce qu’un RAG ?

Le terme RAG (Retrieval-Augmented Generation) désigne une technique qui permet à l’IA de rédiger une réponse claire et adaptée à la question posée en allant chercher les informations provenant de bases de données internes.

Contrairement à un LLM classique qui répond uniquement à partir de ses connaissances internes acquises lors de l’entraînement, un LLM RAG va chercher des informations dans nos propres fichiers et utiliser ces informations pour construire une réponse précise, contextualisée et fiable.

Cette approche garantit que le modèle ne s’appuie pas sur sa mémoire “approximative” mais plutôt sur des données réelles et contrôlées. Dans mon projet, les données proviennent de fichiers Markdown contenant les informations de mon portfolio.

La qualité du RAG dépend fortement de la qualité des documents fournis : textes complets, structure claire (titres, sous-titres) et découpage en *chunks* cohérent.

Attention, un mauvais découpage ou des documents incomplets entraînent des réponses moins pertinentes.

Présentation des fichiers

Fichiers Markdown :

- **compétences.md** : les langues parlées et les compétences informatiques.
- **formations.md** : le BAC passé et les études supérieures réalisées.
- **identité.md** : l ’âge, les qualités, le sport pratiqué et les passions.

Fichiers Python :

- **load_chunk_index.py** : Chargement des variables d’environnement, lecture des fichiers Markdown, chunking et insertion dans l’index vectoriel Upstash.
- **test_query.py** : Script intermédiaire permettant de tester la recherche RAG sans interface.
- **test_agent.py** : Mise en place de l’agent conversationnel. Il interroge l’index vectoriel via une fonction annotée @function_tool, récupère les chunks pertinents et génère une réponse enrichie.
- **streamlit_test.py** : Interface utilisateur (historique, affichage des messages et interaction avec l’agent) via la fonction poser_question().

Comment créer un LLM RAG ?

Documentation utile : Introduction aux Agents & Comment lancer un Agent

1) Préparation des données
   Les informations du portfolio ont été organisées dans des fichiers Markdown structurés avec # et ##. Ce découpage facilite la création de *chunks* cohérents et améliore la qualité de la recherche sémantique.

2) Indexation vectorielle
   Le fichier **load_chunk_index.py** est la base du RAG, il :
   - charge les variables du fichier **.env** (clé API, URL Upstash, token)
   - lit les fichiers Markdown
   - découpe le contenu en *chunks*
   - insère chaque chunk dans Upstash Vector avec un identifiant unique

3) Test intermédiaire
   Le fichier **test_query.py** est utile pour débuguer car il vérifie que :
   - l'index est bien créé
   - la recherche renvoie les bons documents
   - les chunks sont cohérents

4) Mise en place de l'agent
   Le fichier **test_agent.py** :
   - charge automatiquement la configuration
   - interroge l'index vectoriel via @function_tool
   - renvoie les documents les plus pertinents
   - génère une réponse enrichie en français (avec émojis dans mon cas)

   poser_question() sert à envoyer la question à l'agent et à récupérer sa réponse.

5) Création de l'interface Streamlit
   Le fichier **streamlit_test.py** permet de créer une interface permettant d'utiliser le RAG comme un chatbot. Il repose sur 3 éléments :
- st.session_state : mémorisation de l'historique des messages
- st.chat_message : affichage des échanges sous forme de bulles
- poser_question() : envoi de la requête à l'agent et affichage de la réponse

Comment lancer le LLM RAG ?

Il est conseillé d'utiliser VSCode, d'avoir une version supérieure à Python 3.10 et d'utiliser le modèle gpt-4.1-nano. Voici les étapes à suivre :

N°1 : Lancer dans le terminal : ".\.venv\Scripts\Activate.ps1"

N°2 : Lancer dans le terminal : "pip install -r requirements.txt"

N°3 : Lancer dans le terminal : pytest -s

N°4 : Lancer manuellement "load_chunk_index.py" via la flèche VSCode

(Optionnel : Lancer manuellement "test_query")

N°5 : Lancer manuellement "test_agent.py"

N°6 : Lancer dans le terminal : "pip install openai streamlit"

N°7 : Lancer manuellement "Streamlit_test.py"

N°8 : Lancer dans le terminal : "streamlit run .\src\streamlit_test.py"

Comment maintenir cet Agent IA à jour ?

Pour maintenir ce dossier, il est conseillé de garder l'arborescence et de mettre à jour et relancer :

- Fichiers **Markdown** (Data/) lorsque les données du portfolio évoluent.
- Fichier **.env** lorsque la clé API change ou si on recrée un index.
- **load_chunk_index.py** après chaque modification des fichiers Markdown
- Les imports si on renomme un fichier.
- **streamlit_test.py** pour modifier l'apparence : titre, style ou messages d'accueil