

Sécurité pour le Big Data

Comprendre Retrieval-Augmented Generation (RAG)

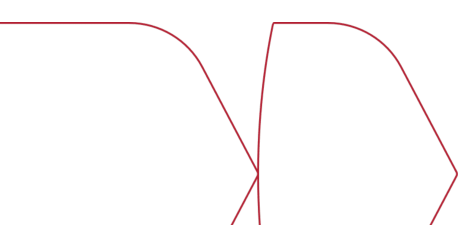
Élèves :

Etienne ESKINAZI

Enseignants :

Julien DREANO

16 juin 2025

A decorative red line art graphic in the bottom left corner, consisting of several overlapping, rounded rectangular shapes.



Lien Git du projet :
https://github.com/AbyssDenier/projet_RAG

Table des matières

1	Présentation de l'environnement construit	3
2	Principales commandes pour l'installation de l'environnement	4
3	Explication des scripts Python développés	5
4	Analyse des réponses du LLM sur le dataset HuggingFace	7
5	Conclusion	9

1 Présentation de l'environnement construit

L'objectif du projet est de mettre en place un système de RAG totalement local, c'est-à-dire un assistant IA capable de répondre à une question utilisateur en s'appuyant sur :

- un modèle de LLM local exécuté avec Ollama (ici : `gemma:2b`),
- une base de connaissances vectorielle basée sur PostgreSQL + `pgvector`,
- des scripts Python qui orchestrent l'interrogation de la base et du modèle.

Ce système permet de combiner la puissance d'un LLM avec des documents spécifiques (non vus pendant l'entraînement du modèle), ce qui rend les réponses plus précises, personnalisées, et plus faciles à sourcer.

Composition de l'environnement technique

Machine hôte

- MacBook Air, macOS
- Python 3.11
- Docker & Docker Compose
- VS Code + GitHub

Conteneurs Docker L'ensemble de l'environnement est déployé dans des conteneurs Docker, permettant une portabilité et un isolement total :

- `ollama` : moteur de LLM local, le modèle utilisé est `gemma:2b`
- `pgvector` : image `ankane/pgvector` avec PostgreSQL 15 + extension vectorielle `pgvector`

Volumes persistants

- `ollama_data` : stocke localement le modèle `gemma:2b`
- `pgdata` : stocke les données PostgreSQL

2 Principales commandes pour l'installation de l'environnement

Le projet est disponible dans le répertoire Git suivant :
https://github.com/AbyssDenier/projet_RAG.

Tout le processus d'installation est détaillé dans le fichier `README.md`, mais voici ci-dessous les principales commandes à exécuter pour prendre en main le projet.

Clonage du dépôt Git

```
git clone https://github.com/AbyssDenier/projet_RAG.git
cd projet_RAG
```

Création et activation de l'environnement virtuel

```
python3.11 -m venv venv_rag_project
source venv_rag_project/bin/activate
pip install -r requirements.txt
```

Lancement des services Docker

```
docker compose up -d
```

- Ollama écoute sur le port 11434
- PostgreSQL écoute sur le port 5433

Connexion à PostgreSQL

```
docker exec -it pgvector psql -U rag_user -d rag_db
```

3 Explication des scripts Python développés

Le projet s'articule autour de deux modes d'utilisation complémentaires, chacun reposant sur une paire de scripts Python.

Mode 1 – Utilisateur : Interrogation à partir de documents personnels

Ce mode permet à l'utilisateur d'interroger un LLM local à partir de fichiers `.txt` personnels.

Script `ingest_documents.py` :

- Lit les fichiers texte dans le dossier `documents/`
- Utilise le modèle `all-MiniLM-L6-v2` pour générer les embeddings des contenus
- Insère dans la base de données (`rag_db`) une ligne par document dans la table `documents` : `filename, content, embedding`

Script `rag_query_document.py` :

- Demande à l'utilisateur de saisir une question
- Génère l'embedding de la question avec `all-MiniLM-L6-v2`
- Récupère les 3 documents les plus proches (cosine distance) depuis la table `documents` via `pgvector`
- Construit un prompt enrichi avec le contexte des documents récupérés
- Interroge le modèle `gemma:2b` via la ligne de commande `ollama` (exécuté dans un conteneur Docker)
- Affiche la réponse du LLM

Mode 2 – Évaluation : Test automatique sur un dataset de référence

Ce mode permet d'évaluer automatiquement la performance du pipeline RAG en comparant les réponses du LLM à des réponses attendues issues d'un dataset connu.

Script `ingest_dataset.py` :

- Charge un sous-ensemble (par exemple 100 exemples) du dataset HuggingFace `rag-dataset-12000`
- Pour chaque ligne, extrait `question, context, answer`
- Vectorise le `context` avec `all-MiniLM-L6-v2`
- Stocke les trois champs ainsi que l'embedding dans la table `dataset` de `rag_db`

Script `rag_query_and_evaluate_dataset.py` :

- Sélectionne un échantillon (par exemple 5) de questions aléatoires dans la table `dataset`
- Pour chaque question :
 - Génère son embedding
 - Récupère les 3 contextes les plus proches dans la table `dataset` (cosine distance)
 - Construit un prompt enrichi et le soumet au LLM `gemma:2b` via `ollama`
 - Compare la réponse générée avec la réponse attendue en calculant la similarité cosinus
- Enregistre tous les résultats dans un fichier CSV `evaluation_rag_dataset.csv` pour analyse

Cette organisation permet de démontrer à la fois :

- la capacité du système à répondre à des questions personnelles avec des documents spécifiques (mode utilisateur),
- et sa performance mesurable sur un jeu de données standardisé (mode évaluation).

Workflow RAG – Mode utilisateur

Le fonctionnement du système RAG en mode utilisateur suit les étapes suivantes :

1. L'utilisateur saisit une question.
2. L'embedding de la question est généré à l'aide du modèle `all-MiniLM-L6-v2`.
3. Une recherche vectorielle est effectuée dans la base `pgvector`.
4. Les 3 contextes les plus proches sont sélectionnés.
5. Un prompt enrichi est construit en intégrant ces 3 contextes.
6. Le prompt est soumis au modèle `gemma:2b` via `Ollama`.
7. La réponse générée par le LLM est affichée à l'utilisateur.

```
• (projet_rag) Air-de-Etienne:rag_llm_project etienneeskinazi$ docker compose up -d
WARN[0000] /Users/etienneeskinazi/Documents/MS_BGD/P4/projet_RAG/rag_llm_project/docker-compose.yml: the attribute `version` is obso
e, it will be ignored, please remove it to avoid potential confusion
[+] Running 2/2
  ✓ Container pgvector Started
  ✓ Container ollama Started
• (projet_rag) Air-de-Etienne:rag_llm_project etienneeskinazi$ /Users/etienneeskinazi/my_venv/projet_rag/bin/python /Users/etiennee
i/Documents/MS_BGD/P4/projet_RAG/rag_llm_project/rag_query_document.py
Pose ta question : Capitale de la France ?

Réponse du LLM :
La capitale de la France est Paris.
• (projet_rag) Air-de-Etienne:rag_llm_project etienneeskinazi$
```

FIGURE 1 – Exécution du script `rag_query_document.py` : question posée à l'utilisateur et réponse générée par le modèle `gemma:2b` via `Ollama`.

4 Analyse des réponses du LLM sur le dataset Hugging-Face

L'ensemble des résultats de l'évaluation automatique est disponible dans le fichier CSV `evaluation_rag_dataset.csv`. Ce fichier a été généré à partir du test effectué sur un sous-ensemble du dataset HuggingFace `rag-dataset-12000`.

Pour chaque question issue du dataset :

- la réponse générée par le LLM est comparée à la réponse attendue fournie dans le dataset,
- la similarité cosinus est calculée entre les deux réponses vectorisées,
- cette similarité donne une mesure de précision : *plus elle est proche de 1, plus la réponse du LLM est jugée pertinente et fidèle au contenu attendu.*

Cette méthode permet donc d'évaluer objectivement la capacité du système RAG à fournir des réponses cohérentes et précises sur un jeu de données standardisé.

Génération du contexte

Pour chaque question posée, le système RAG commence par générer son embedding vectoriel à l'aide du modèle `all-MiniLM-L6-v2`. Cet embedding est ensuite comparé à ceux déjà stockés dans la base `PostgreSQL`, afin d'identifier les documents les plus proches sémantiquement. Cette recherche s'appuie sur la fonction de distance cosinus (\cos) fournie par l'extension `pgvector`. La distance cosinus mesure l'angle entre deux vecteurs, ce qui la rend particulièrement adaptée aux embeddings textuels normalisés, car :

- elle ignore la longueur du texte,
- elle capte efficacement la proximité sémantique,
- elle est largement recommandée dans les applications de traitement du langage naturel (NLP).

Le système récupère ainsi les trois documents les plus proches (ceux présentant la plus faible distance cosinus) et les concatène pour construire un *contexte enrichi*. Ce contexte est ensuite injecté dans le prompt envoyé au LLM (`gemma:2b`), afin qu'il produise une réponse plus précise, ancrée dans une base de connaissances pertinente.

Analyse des résultats

ID	Question	Réponse attendue	Réponse RAG	Sim. cosinus
331	What are some of the original features of the 1930's detached bungalow ?	internal doors, picture rails...	internal doors, skirting boards, arched entrance...	0.6876
227	What is the impact of decreasing negative posts ?	rise in positive posts	"not provided in context"	0.7123
60	What happens with Keldeo and legendary Pokémon ?	Keldeo learns Secret Sword	Keldeo learns Secret Sword and transforms	0.8987
292	Main challenge with command line ?	Time to learn	Need time & effort to read docs	0.8515
332	Notable changes in T-Mobile Moto X update ?	Print support, bug fixes	Résolution, Android 4.0, etc.	0.5487

TABLE 1 – Extraits d'évaluation du pipeline RAG sur des données issues du dataset `rag-dataset-12000`.

Les scores de similarité cosinus montrent que le pipeline RAG génère globalement des réponses pertinentes (souvent supérieures à 0,7), notamment lorsque le contexte contient explicitement l'information attendue (exemple : question 60, score de 0,8987). Cependant, certaines réponses restent incomplètes ou imprécises (questions 227 et 332), en raison d'un contexte mal adapté ou trop vague. Cela illustre une limite importante : la qualité de la réponse dépend fortement de la qualité des documents récupérés.

Des améliorations possibles seraient les suivantes :

- récupérer un plus grand nombre de documents (top 5) pour élargir le contexte
- utiliser des modèles d'embedding plus puissants, le projet utilise actuellement le modèle `all-MiniLM-L6-v2` pour générer les embeddings textuels.
- ajouter un filtrage sémantique sur le contexte avant de l'envoyer au LLM, c'est-à-dire, sélectionner uniquement les passages des documents récupérés qui sont réellement pertinents pour la question.

5 Conclusion

Ce projet a permis de mettre en place un système complet de Retrieval-Augmented Generation (RAG) entièrement local, combinant une base vectorielle PostgreSQL/pgvector, un moteur LLM exécuté via Ollama, et des scripts Python pour orchestrer l'ensemble. Deux modes d'utilisation ont été implémentés : un mode interactif pour interroger le LLM à partir de documents personnels, et un mode d'évaluation automatique utilisant un dataset standardisé.

L'évaluation du pipeline sur un sous-ensemble du dataset `rag-dataset-12000` montre que le système est capable de produire des réponses globalement pertinentes, avec des scores de similarité cosinus souvent satisfaisants. Toutefois, certaines limites apparaissent dès que le contexte fourni est trop éloigné ou imprécis par rapport à la question posée.

Pour améliorer les performances, plusieurs pistes sont envisageables : renforcer la qualité du contexte via un filtrage sémantique plus fin, augmenter le nombre de documents récupérés, ou encore adopter des modèles d'embedding plus performants que `all-MiniLM-L6-v2`, actuellement utilisé.