

## Rapport Projet RAG

Je suis fière de vous présenter mon projet RAG.

Vous trouverez le lien Github avec l'ensemble du code par ici : [GitHub](#)

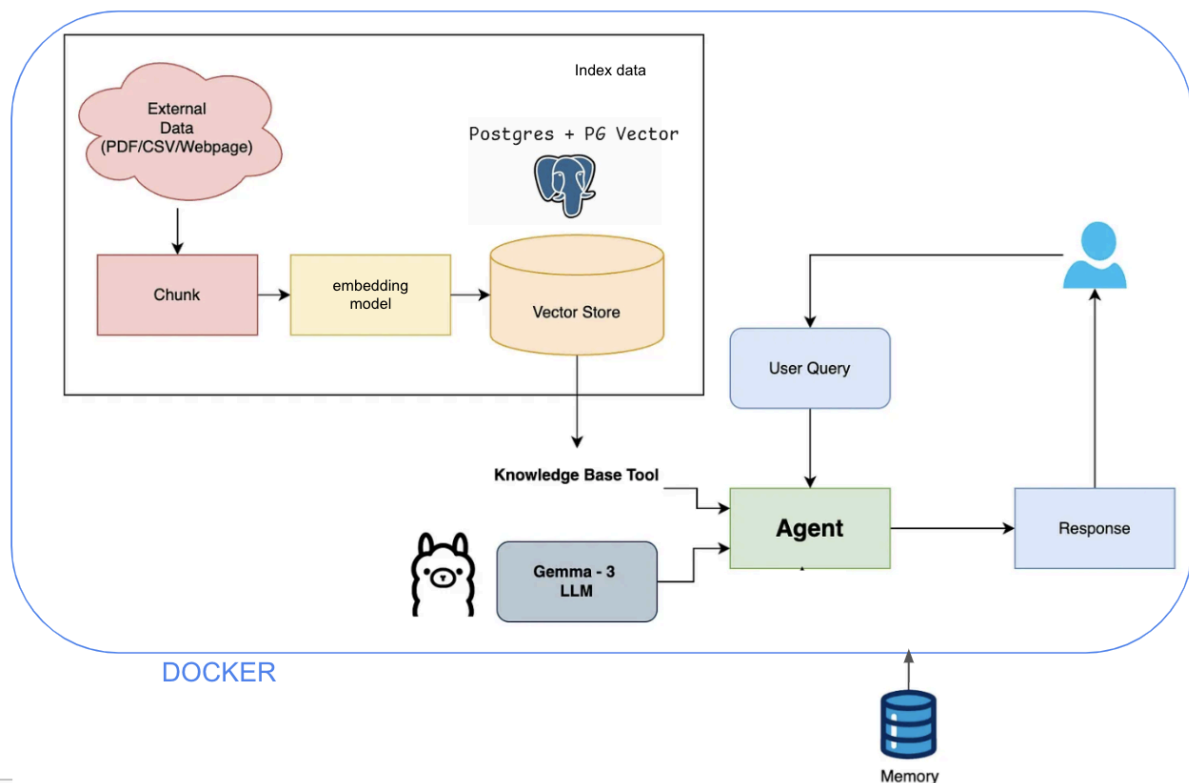
L'objectif de ce projet est d'implémenter un système fonctionnel de RAG en utilisant un LLM qui tourne localement avec Ollama server. Ce système doit être capable d'ingérer une base de connaissance externe et de l'utiliser pour enrichir les réponses du LLM à des questions spécifiques. Dans ce rapport, j'expliquerai mes choix d'architectures, je décrirai le workflow de traitement et j'évaluerai la bonne prise en compte du système RAG par le LLM.

### I. Choix de l'environnement et des composants techniques

#### **A. Architecture globale**

L'environnement est orchestré par Docker Compose et est composé de trois services principaux : le serveur Ollama, la base de données vectorielles Postgre (avec pgvector) et l'application python du RAG.

*Schéma de l'architecture :*



## B. Déploiement

Le brief projet propose deux options : l'installation directe sur une machine virtuelle (VM) ou l'utilisation de conteneurs Docker.

J'ai choisi d'utiliser la conteneurisation avec Docker pour les raisons suivantes :

- Les conteneurs offrent un environnement isolé et léger, or je n'ai pas une machine très puissante. Contrairement à une VM, le conteneur ne virtualise pas le système d'exploitation : il s'appuie directement sur la puissance de calcul de la machine hôte, ce qui le rend moins gourmand en ressources.
- Plus simple et généralisable, cette approche me permet de définir l'ensemble de mon environnement (base de données, serveur LLM, application) dans un seul fichier de configuration (le `docker-compose.yml`). De plus cela me permet de gérer facilement les différents services comme des conteneurs connectés. Ils peuvent facilement interagir ensemble dans l'architecture.
- De plus, si je désire réutiliser ce projet et passer le système en production dans le futur, Docker facilitera le déploiement.

Pour assurer la persistance de l'environnement, j'ai monté deux volumes : `ollama_data` et `postgres_data`. Le volume `ollama_data` permet de stocker les modèles téléchargés, m'évitant ainsi de les re-télécharger à chaque redémarrage du conteneur. Et celui `postgres_data` permet de garder les données ingérées en base de données lorsque le conteneur postgres est redémarré.

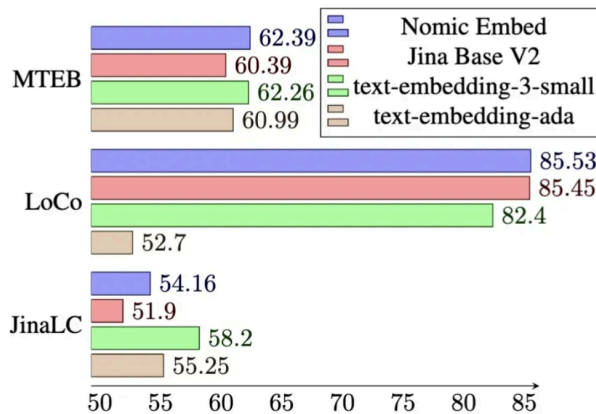
Les configurations sont détaillées plus loin dans le rapport.

## C. Composants techniques

Pour faire tourner le LLM localement, j'utilise **Ollama server** (la version conteneurisée avec Docker). Il est open source, simple à déployer et il dispose d'un catalogue de modèles prêts à l'emploi. Il fournit une API pour l'inférence des LLM (c'est-à-dire pour générer les réponses), ainsi que pour la création d'embeddings.

Concernant le modèle de LLM, j'utilise **Gemma** (gemma3:4b) qui est décrit dans le brief projet comme un LLM léger nécessitant peu de puissance de calcul, ce qui est idéal pour un environnement de développement local. D'autant plus qu'il est très performant avec un contexte étendu à 128k (au lieu de 32k). Dans un usage futur, si je souhaite intégrer des capacités multilingues, il sera intéressant de passer sur Llama.

Pour le modèle d'embedding, gemma3 n'ayant pas de capacités d'embedding, j'ai dû utiliser un modèle spécifique. J'ai choisi **nomic-embed-text** qui semble être l'état de l'art actuel. Il est open source et est le modèle le plus téléchargé dans la librairie Ollama (lorsque je recherche avec le mot clé *embed*). Avec un contexte augmenté, il dépasse les modèles répandus tels que OpenAI Ada-002 ou text-embedding-3-small et permet de traiter des longues séquences de texte.



Pour la base de données, comme spécifié dans le brief, j'ai utilisé la combinaison **PostgreSQL + pgvector**. L'extension me permet de stocker les chunks de texte et leurs embeddings afin de réaliser des recherches de similarités vectorielles. De plus, il existe une image Docker officielle qui intègre pgvector, facilitant ainsi le déploiement.

Enfin pour l'application RAG, j'ai créé deux scripts en Python qui gèrent le processus d'ingestion des données dans la base de connaissance (lecture, découpage, embedding, stockage) et le processus de requête (récupération des informations, envoi au LLM, génération de réponse). Les processus sont détaillés dans la partie "Scripts commentés". L'application est montée via un volume Docker, pour que je puisse développer de manière itérative sans avoir à reconstruire l'image Docker à chaque modification du code Python (TRÈS utile). Grâce à la commande `tail -f /dev/null`, le code reste actif ce qui permet l'exécution ponctuelle des scripts avec la commande `docker compose exec` (voir partie processus d'installation et de lancement). Enfin j'utilise la librairie **LangChain** comme le suggère le brief. Elle fournit un framework pour interagir avec les LLMs, gérer les embeddings, ou encore manipuler les bases de données vectorielles. Mais elle permet surtout de construire et orchestrer des chaînes RAG complexes.

## II. Processus d'installation et de lancement du RAG

*Le processus d'installation est également décrit dans le Readme du projet.*

### A. Prérequis :

Docker Desktop (ou Docker Engine + Docker Compose) doit être installé sur votre machine.

### B. Processus d'installation et de démarrage du projet :

1. Cloner le dépôt :

```
git clone [https://github.com/audedef/rag-llm-local.git]
cd rag-llm-local
```

2. Placer la base de connaissance dans le projet :

Déposer les fichiers (au format “.txt” ou “.pdf” par exemple) dans le répertoire “rag\_app/knowledge\_base/”. De mon côté, pour l'exercice, j'utilise Hugging Face et sa bibliothèque Dataset pour importer directement le dataset.

3. Démarrer les services Docker :

Commencer par lancer Docker Desktop puis depuis le dossier rag-llm-local, lancer la commande :

```
docker-compose up --build -d
```

Cela va construire l'image “rag\_app”, démarrer Ollama, PostgreSQL et “rag\_app”.

4. Vérifier l'état des services :

```
docker-compose ps
```

Il faut attendre que tous les services soient "healthy".

5. Télécharger le modèle Gemma sur Ollama si ce n'est pas déjà fait :

```
docker exec -it ollama_server ollama run gemma3:4b
```

On utilise un petit modèle qui requiert une mémoire de seulement 3.3GiB

La première fois, Ollama téléchargera le modèle Gemma, ce qui peut prendre du temps.

Grâce au volume Docker, il n'y aura pas besoin de re-télécharger le modèle après redémarrage de Docker.

Puis le modèle d'embedding :

```
docker exec -it ollama_server ollama pull nomic-embed-text
```

Il est possible de s'assurer que les modèles sont toujours dans le conteneur avec la commande :

```
docker exec -it ollama_server ollama list
```

6. Ingérer la base de connaissance :

Exécuter le script d'ingestion depuis le conteneur “rag-app” :

```
docker compose exec rag-app python ingest.py
```

Cela va traiter les documents, créer les embeddings et les stocker dans PostgreSQL.

7. Interroger le système RAG :

Exécuter le script de requête depuis le conteneur “rag-app” :

```
docker compose exec rag_application python query.py
```

Il est maintenant possible de poser des questions au LLM enrichi du système RAG en les tapant directement dans le terminal.

## C. Evaluation

Afin de prouver l'efficacité du système RAG, un script d'évaluation est à disposition.

Il comporte déjà un ensemble de questions prédéfinies, basées sur la base de connaissance Hugging Face ingérée. Si une autre base de connaissance a été ingérée, modifier les questions dans le script.

Exécuter le script avec :

```
docker compose exec rag_application python evaluate.py
```

Le résultat sortira sous forme de dataframe dans votre terminal, et sous forme de fichier markdown téléchargé dans votre dossier local.

## D. Nettoyage

Pour arrêter et supprimer les conteneurs et les volumes Docker

```
docker compose down -v
```

Attention car cela supprimera aussi les modèles Ollama et les données PostgreSQL.

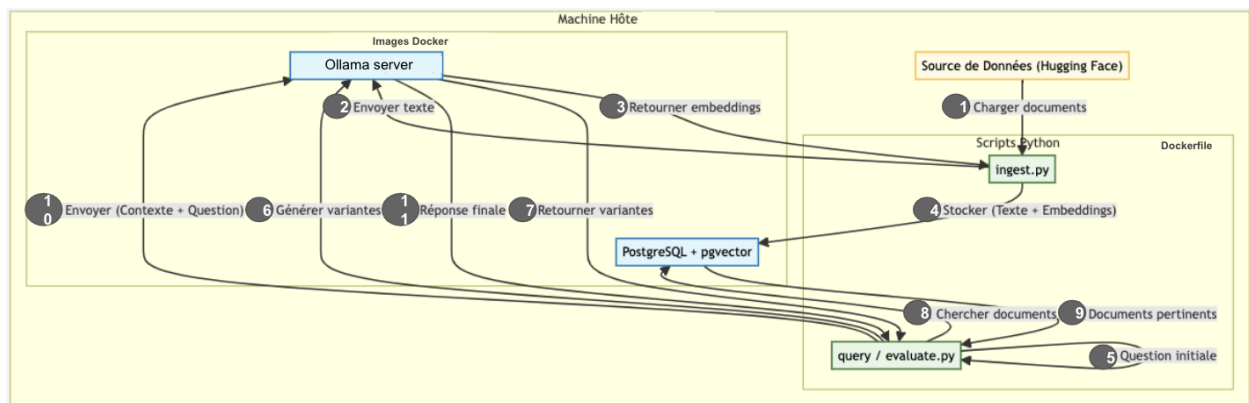
## III. Scripts commentés et workflow de traitement

Le projet est constitué de fichiers de configuration Docker et des scripts de l'application Python. L'application python est constitué de trois scripts :

- [ingest.py](#) qui crée la base de connaissance du système RAG. Il charge des documents (dans notre exemple depuis un dataset spécialisé pour les RAGs, chargé avec Hugging Face), les découpe en chunks et les stocke dans une base de données vectorielle PostgreSQL avec pgvector.
- [query.py](#) qui permet d'interroger le RAG. Il récupère des informations pertinentes depuis la base de données vectorielle pour contextualiser la question de l'utilisateur avant de la soumettre au LLM.
- [evaluate.py](#) qui a pour but d'évaluer la bonne prise en compte de la base de connaissance RAG par le LLM.

### A. Workflow de traitement

*Schéma du workflow (cas du multi-query retriever) :*



## B. Configuration Docker

Le fichier `docker-compose.yml` définit et orchestre les différents services de notre architecture RAG. Il permet de démarrer, arrêter et gérer l'ensemble de l'environnement de développement d'un seul coup. Les différents services sont les suivants :

- *serveur Ollama pour les LLM* : je spécifie tout d'abord l'image Docker à importer, celle officielle `ollama/ollama`. Puis je donne un nom au conteneur qui servira pour les commandes `docker exec`. Puis je mappe le port 11434 du conteneur (celui par défaut) au port 11434 de ma machine. Ensuite je monte le volume `ollama_data` pour assurer la persistance des modèles Ollama téléchargés. Puis je connecte ce service au réseau Docker `rag_network`, ce qui permet la communication entre les services au sein du réseau. Je définis ensuite les variables d'environnement `OLLAMA_LOAD_TIMEOUT` et `OLLAMA_KEEP_ALIVE` qui permettent d'améliorer la robustesse du service, notamment pour les gros modèles comme Gemma. Elles augmentent les délais d'Ollama pour le chargement et le maintien en mémoire des modèles. Enfin je configure un "healthcheck" qui vérifie si le service est bien prêt. Avec `test: ["CMD-SHELL", "ollama list || exit 1"]`, j'exécute "ollama list" à l'intérieur du conteneur. Si cette commande réussit, le conteneur est considéré comme *healthy*. Puis la `command: serve` assure que le serveur Ollama est lancé.
- *base de données PostgreSQL avec l'extension pgvector* : le process est similaire à celui d'Ollama. L'image "pgvector/pgvector:pg16" me permet d'avoir l'extension pgvector déjà activée. Les variables d'environnement configurent les identifiants de la base de données lors de l'initialisation. Concernant le healthcheck, l'utilitaire "pg\_isready" permet de vérifier que la base de données est accessible avec les identifiants spécifiés.
- *application RAG* : pour ce service, j'ai dû build l'image à partir d'un Dockerfile. Pour ce faire, je spécifie le chemin vers le répertoire contenant le Dockerfile avec `context: ./rag_app` puis j'indique le nom du Dockerfile dans le contexte avec `dockerfile: Dockerfile`. Ce service n'expose pas de port car il s'agit d'un script qui est exécuté à la demande via la commande `docker compose exec`, et non d'un serveur web permanent. Si je souhaite exposer une API dans le futur, il faudra que j'ajoute ce mapping 8000:8000. Concernant les variables d'environnement, elles permettent ici au script Python de se connecter aux services Ollama et PostgreSQL. Elles spécifient également les modèles à utiliser. Le volume monté permet de modifier le code Python sur ma machine et de l'exécuter immédiatement dans le conteneur sans avoir à reconstruire l'image Docker. Spécifier la connexion networks permet à l'app d'accéder aux autres services. Enfin j'ai ajouté des dépendances entre les services : la partie `depends_on` spécifie que ce service ne doit démarrer que si les services dépendants (ollama et postgres) sont *healthy*. Cela assure que la base de données et le LLM sont prêts avant que l'application ne tente de les utiliser.
- Je finis par déclarer les volumes et le réseau. `driver: bridge` spécifie que je veux utiliser le driver de réseau par défaut de Docker, pour créer un réseau isolé où les conteneurs peuvent communiquer entre eux par leurs noms de service.

## C. Script d'ingestion

Le script `ingest.py` constitue la phase de préparation de notre base de connaissance.

Le workflow d'ingestion est le suivant :

1. Chargement de la base de connaissance : le script lit les documents bruts à partir d'une source : ici, un dataset Hugging Face.
2. Découpage en chunks : Les documents longs sont divisés en plus petits segments de texte ("chunks"). Cette étape est importante car les modèles d'embeddings ont une limite de tokens, et des chunks plus petits permettent une recherche sémantique plus précise.
3. Génération d'embeddings : chaque chunk est envoyé à l'API d'Ollama qui charge le moteur d'embedding "nomic-embed-text" pour être converti en embedding, les vecteurs numériques. Ces vecteurs représentent le sens sémantique du texte.
4. Stockage vectoriel : Les chunks de base et leurs embeddings sont stockés dans la base de données PostgreSQL avec pgvector (module Postgre qui permet le stockage et la manipulation d'objets vectoriels).

Explication du code :

- *1. configurations initiales* : les variables sont récupérées de l'environnement du conteneur qui sont définies dans le fichier `docker-compose.yml`. Elles doivent correspondre aux services `postgres_db` et `ollama_server`. L'URL de connexion à la base de données est de format standard pour se connecter à PostgreSQL via `psycopg2` et il utilise ces variables. C'est là que les chunks et leurs embeddings sont stockés. Concernant le modèle d'embedding, j'ai ajouté à `OLLAMA_BASE_URL` la méthode `.strip()` pour supprimer tout espace blanc accidentel qui pourrait invalider l'URL (ce qui m'est arrivé lors des tests).
- *2. chargement des données* : j'utilise `HuggingFaceDatasetLoader`, une classe de `LangChain` qui simplifie le chargement de datasets depuis le Hub d'Hugging Face. Pour l'exercice, j'ai limité le chargement à un échantillon de 100 documents. Mais pour un passage en production, il faudrait charger l'ensemble du dataset.
- *3. chunking* : j'utilise `RecursiveCharacterTextSplitter`, un diviseur de texte de `LangChain`. Il divise le texte de manière récursive en essayant de préserver au maximum la sémantique grâce à une liste de séparateurs (paragraphe, phrases, mots, etc.). Le paramètre `chunk_overlap` sert à préserver le contexte lorsque des informations importantes se trouvent à la frontière entre deux chunks.
- *4. création des embeddings et stockage vectoriels* : j'ai ajouté ici un bloc de debug qui a servi à résoudre l'erreur "Failed to parse" en faisant une requête HTTP directe à l'API d'Ollama et en affichant la réponse brute. J'ai ainsi découvert que le modèle "gemma3:4b", que j'avais choisi d'utiliser à la fois comme LLM et modèle d'embedding, ne supportait pas les embeddings, et ai donc changé pour "nomic-embed-text". Ici j'utilise la classe `OllamaEmbeddings` de `LangChain` qui interagit avec l'API d'Ollama. J'utilise également la méthode `from_documents` de `PGVector` qui me permet directement de me connecter à la base de données

### Les logs de l'ingestion :

### D. Script de requête du RAG

Le workflow de requête est le suivant :

1. Requête de l'utilisateur : Une question est posée par l'utilisateur.
2. Génération de l'embedding de la requête : la question de l'utilisateur est convertie en embedding en utilisant le même modèle (nomic-embed-text).
3. Recherche de similarité = "retrieval" : cet embedding est ensuite utilisé pour interroger la base de données pgvector afin de trouver les chunks de la base de connaissance qui ont les embeddings les plus similaires (c'est-à-dire sémantiquement proches) à celui de la question de l'utilisateur.
4. Construction du contexte : les chunks les plus pertinents sont récupérés et assemblés pour former un "contexte" pour le LLM.
5. Génération de la réponse : le contexte récupéré, combiné à la question de l'utilisateur, est envoyé au LLM gemma3. Celui-ci utilise le contexte pour générer une



réponse informée, évitant ainsi les hallucinations et fournissant des réponses plus pertinentes basées sur la base de connaissance.

Explication du code :

- *1. configurations initiales* : même chose que pour l'ingestion. La seule différence est qu'en plus du modèle d'embedding, j'utilise un LLM "gemma3:4b" qui génère la réponse.
- *2. initialisation des composants LangChain* : j'utilise `OllamaEmbeddings` pour convertir la question de l'utilisateur en embedding. Je crée également un "retriever" à partir du vector store pour récupérer les documents pertinents. `{"k": 5}` signifie qu'on récupère les 5 chunks les plus pertinents, c'est-à-dire les plus similaires à la question.
- *3. définition du prompt template* : à cette étape j'utilise `PromptTemplate` qui permet de créer un prompt structuré qui sera alimenté par le contexte récupéré et la question de l'utilisateur.
- *4. création de la RAG chain* : cette partie permet d'orchestrer le pipeline grâce à LangChain Expression Language. Mon pipeline se lit de la manière suivante : # 1. On prend la question de l'utilisateur // # 2. Le retriever recherche le contexte pertinent. // # 3. On passe la question et le contexte récupéré dans le prompt. // # 4. Le prompt formaté est envoyé au LLM. // # 5. La sortie du LLM est parsée pour obtenir une chaîne de caractères.
- *5. boucle d'itération* : cette boucle permet à l'utilisateur de poser plusieurs questions successivement. La fonction `stream` permet d'afficher la réponse mot par mot.

Les logs de la requête :

```
(venv01) → rag_llm git:(main) ✗ docker compose exec rag-app python query.py
— Configuration du client RAG chargée —
Modèle LLM: gemma3:4b
Modèle d'embedding: nomic-embed-text:latest
Collection PGVector: rag_knowledge_base

Initialisation des composants LangChain
/usr/local/lib/python3.10/site-packages/langchain_core/_api/deprecation.py:139: LangChainPendingDeprecationWarning: This class is pending deprecation and may be removed in a future version. You can swap to using the `PGVector` implementation in `langchain_postgres`. Please read the guidelines in the doc-string of this class to follow prior to migrating as there are some differences between the implementations. See https://github.com/langchain-ai/langchain-postgres for details about the new implementation.
  warn_deprecated()
/usr/local/lib/python3.10/site-packages/langchain_community/vectorstores/pgvector.py:322: LangChainPendingDeprecationWarning: Please use JSONB instead of JSON for metadata. This change will allow for more efficient querying that involves filtering based on metadata. Please note that filtering operators have been changed when using JSONB metadata to be prefixed with a $ sign to avoid name collisions with columns. If you're using an existing database, you will need to create a migration for your metadata column to be JSONB and update your queries to use the new operators.
  warn_deprecated()
Retriever configuré pour chercher les 3 documents les plus pertinents.

✅ Le système RAG est prêt. Posez vos questions.
Tapez 'exit' ou 'quit' pour quitter.

Votre question: why must we recycle concrete ?

Réponse de l'assistant:
Recycling concrete in construction helps in reutilizing the construction waste into new concrete, preserving landfill space, and reducing construction costs.

=====
Votre question: 
```

## E. Script d'évaluation

Le script `evaluate.py` a pour but d'évaluer l'efficacité de notre RAG, en comparant les réponses d'un LLM seul avec celles du même LLM enrichi par le contexte récupéré dans la base de connaissances ingérée. Le résultat est un tableau comparatif au format Markdown qui compare à la fois la pertinence des résultats et le temps pris pour générer la réponse.

Explication du code :

- *Définition des questions de test* : le script démarre par une liste de questions pour évaluer le système dans différents scénarios (si la réponse est dans la base de connaissances, si elle est proche de la base de connaissances, si elle est totalement hors sujet).
- Suite à des erreurs, j'ai ajouté une fonction de débogage qui permet d'afficher le contexte récupéré en lien avec la question. Cela m'a permis de comprendre quels documents sont fournis au LLM pour la génération de la réponse.
- *fonction `setup_chains`* : j'initialise tout d'abord le LLM (avec température = 0 qui réduit l'aléatoire et la créativité du LLM -> important pour l'évaluation). Puis je paramètre la chaîne RAG. Pour cette dernière, j'ai testé et comparé deux types de retriever : le retriever simple avec `k = 5` (qui récupère les 5 chunks les plus pertinents) et un multi-query retriever qui génère plusieurs reformulations de la question originale pour améliorer la pertinence de la recherche dans le vector store. Dans cet exercice, les réponses du RAG sont les mêmes mais le multi-query est plus lent (*voir partie Résultats*). Le multi-query serait probablement plus adapté pour des cas plus complexes. Dans le prompt de la chaîne RAG, je demande au LLM de ne se référer qu'au contexte fourni. Enfin je définis un prompt sans RAG, pour le LLM seul, qui servira de base de comparaison.
- *fonction `run_evaluation`* : cette fonction initialise les deux chaînes (RAG et non RAG) et itère sur chaque question. Elle mesure également le temps que le LLM met pour répondre. Enfin elle stocke les résultats dans `results`.
- enfin la fonction `main` exécute l'évaluation et génère le rapport au format markdown.

## IV. Résultats

Voici le tableau de résultats obtenu après run du script `evaluate.py`.

Ils sont également disponibles au format pdf et markdown dans le repository Github.

Type de question	Question	Réponse avec RAG	Durée RAG multi query	Durée RAG simple retriev	Réponse sans RAG	Durée sans RAG
Dans la base de connaissances	Advise me some podcasts about love and relationships	Here are five love and relationship podcasts mentioned in the context: * Love Is Like A Plant with Sarah May Bates and Ellen Huerta * Relationships! Let's Talk About It with Pripo Teplitsky * Unqualified with Anna Faris * Confessions of a Terrible Husband with Nick Pavlidis * Savage Lovecast with Dan Savage	196.92	89.78	Bien sûr, voici quelques podcasts sur l'amour et les relations, classés par catégories pour vous aider à trouver celui qui correspond le mieux à vos intérêts : - Pour des discussions profondes et introspectives : * <b>Where Should We Begin? with Esther Perel:</b> Esther Perel est une psychologue de la relation renommée. Ce podcast explore des conversations avec des couples sur des problèmes relationnels réels, des stratégies de communication et des perspectives sur l'amour et l'engagement. C'est un podcast très respecté et souvent recommandé. ... - Pour des conversations légères et amusantes : * <b>2 True:</b> Deux amis, Sarah and Emily, discutent de relations, de rencontres, de sexe et de tout ce qui se passe dans le monde des relations. C'est un podcast divertissant et accessible. ... - Pour des perspectives spécifiques : * <b>Love Notes:</b> Ce podcast est axé sur les relations LGBTQ+  J'espère que cette liste vous sera utile ! N'hésitez pas à me dire si vous avez des questions plus spécifiques sur un podcast en particulier.	240.37
Dans la base de connaissances	Why must we recycle concrete in the construction industry?	Recycling concrete in construction helps in reutilizing the construction waste into new concrete, preserving landfill space, and reducing construction costs.	177.47	76.57	We must recycle concrete in the construction industry for several key reasons: * <b>Resource Conservation:</b> Concrete is a massive resource, primarily composed of limestone, shale, and clay. Recycling reduces the demand for extracting these virgin materials, which involves significant environmental disruption like quarrying and land degradation. * <b>Reduced Landfill Waste:</b> Concrete is one of the most abundant materials in landfills globally. Recycling diverts a huge volume of this waste, extending the lifespan of landfills and reducing the associated problems (methane emissions, groundwater contamination, etc.). * <b>Energy Savings:</b> Producing new concrete from raw materials is an energy-intensive process. Reusing concrete significantly reduces the energy required, lowering the carbon footprint of construction projects. * <b>Carbon Sequestration:</b> Concrete naturally absorbs carbon dioxide during its formation. Recycling concrete allows us to retain this captured carbon, rather than releasing it back into the atmosphere when new concrete is produced. * <b>Aggregate Supply:</b> Recycled concrete can be crushed and used as aggregate in new concrete mixes, further reducing the need for natural aggregate extraction.  In short, recycling concrete is a sustainable practice that benefits the environment, conserves resources, and reduces the overall impact of the construction industry.	77.48
Dans la base de connaissances	How many followers has the most followed person on Instagram?	Selena Gomez has 72.5 million followers.	70.16	80.39	As of today, November 2, 2023, the most followed person on Instagram is Cristiano Ronaldo, with approximately 606 million followers	11.03

Sujet proche mais hors base	What are the fashion trends for next fall?	Some trendy short hairstyles for fall include a short bob for women with curly hair, a short straight hairstyle that suits many face shapes, a light blonde pixie haircut, a classic bob haircut, a straight short bob with the same length, a playful layered razor cut, a casual short hairstyle, a trendy short pixie hairstyle, a short pixie mohawk, a razor cut mingled with a clipper cut, an asymmetrical layered cut, and a simple pixie hairstyle.	240.2	122.04	Okay, let's look at what's predicted for Fall 2024 fashion trends! Based on runway shows and trend forecasting reports, here's a breakdown of what's expected: Overall Themes: * Quiet Luxury Continues (But with a Twist): The emphasis on understated elegance and high-quality materials isn't going anywhere, but it's becoming more playful and less rigidly formal. Think luxurious fabrics with a touch of personality. * Return to Romance: Softness and femininity are making a strong comeback. * Utility & Workwear Influence: Practicality and functionality are still relevant, but it's moving beyond purely utilitarian looks. Specific Trends: * Color Palettes: * Deep Reds & Burgundies: A major focus –think rich, saturated reds, from cherry to oxblood. ... Important Note: Trends are constantly evolving, and individual style will always be key. These are predictions based on current forecasts, but personal taste and experimentation are always encouraged!	199.41
Totalement hors base (culture générale)	Who was the first First Lady of the United States?	The provided context does not contain information about the first First Lady of the United States. It discusses various topics including barstools, Franklin Templeton Investments, AUNE, and the location of a tannery.	216.66	90.68 w/ small diff : "I am sorry, but the provided context does not contain information about the first First Lady of the United States"	Martha Washington was the first First Lady of the United States. She was the wife of George Washington, the first President, and served in that role from 1789 until his death in 1799.	14.32
Totalement hors base (créatif)	What is the meaning of life ?	The provided documents do not contain an answer to the question "What is the meaning of life?". They discuss topics such as the pursuit of property, the role of women, interdisciplinary research, and end-of-life care.	126.58	92.55 w/ small diff : The provided documents do not contain information about the meaning of life. They focus on topics such as end-of-life care, the role of women, and the symbolism of gemstones.	That's a question that has puzzled philosophers and individuals for centuries! There isn't a single, universally accepted answer to the meaning of life. Here's a breakdown of different perspectives: * Nihilism: This philosophy suggests that life has no inherent meaning or purpose. ... Ultimately, the meaning of life is often what you make it. It's a deeply personal question with no right or wrong answer. It can evolve over time as you grow and experience the world.	96.48

Ces résultats démontrent que le RAG a bien été pris en compte par le LLM. Notre application fonctionne. Les réponses aux questions présentes dans la base de connaissances sont précises et concises, contrairement au LLM seul qui génère des réponses trop détaillées, avec par moment quelques éléments d'hallucinations. Par exemple, en effectuant une recherche approfondie sur le web, j'ai pu réaliser que le podcast Love Notes existe. Il en existe même plusieurs du même nom. Mais aucun ne traite des relations LGBTQ+.

Voici dans la base de données les entrées correspondantes aux questions posées. Cela nous permet de vérifier que le RAG a bien trouvé le document le plus proche sémantiquement de la question posée. Pour rappel, seul le contexte est ingéré et embeddé. Le LLM n'a pas connaissance des questions et réponses présentes dans le dataset.

### Question 1 : “Advise me some podcasts about love and relationships”

context	question	answer
string · lengths 4.72k→5.61k 13.8%	string · lengths 48→79 47.5%	string · lengths 243→484 25.6%
<p>5 Great Podcasts on Love and Relationships/</p> <p>As much as you'd like to think that you can handle your love and relationship problems on your own like a champ, it never hurts to get advice (or even an entirely different perspective) from people who have already been there, done that, and set up an appropriately themed podcast to talk about it. But with the sheer number of love and relationship podcasts out there, you just can't help but wonder: Which ones among them should you be listening to? And frankly, does it even matter? Seriously, how many different ways can people actually talk about such a universal and all-encompassing emotion as love? As it turns out, quite a lot – and these five love and relationship podcasts are among the best and most unique ones out there.</p> <p>Love Is Like A Plant with Sarah May Bates and Ellen Huerta</p> <p>What do you get when you combine two experts on love and heartbreak in one podcast? An informative and in-depth conversation about planting the seeds of love in your life and making it last, no matter what kinds of challenges come your way.</p>	<p>What are the five love and relationship podcasts mentioned in the context?</p>	<p>The five love and relationship podcasts mentioned in the context are "Love Is Like A Plant" with Sarah May Bates and Ellen Huerta, "Relationships! Let's Talk About It" with Pripo Teplitsky, "Unqualified" with Anna Faris, "Confessions of a Terrible Husband" with Nick Pavlidis, and "Savage Lovecast" with Dan Savage.</p>

### Question 2 : “Why must we recycle concrete in the construction industry?”

context	question	answer
string · lengths 5.61k→6.49k 8.6%	string · lengths 48→79 47.5%	string · lengths 2→243 63%
<p>construction of concrete crushing recycling in verona</p> <p>Construction Waste Recycling concrete-crusher.org</p> <p>Construction Waste Recycling Technology equipments to process construction waste into reutilized concrete Construction Waste Recycling; Concrete Crusher;</p> <p>Crushed Concrete Aggregate Concrete Construction</p> <p>Crushing old concrete to make aggregate for new concrete is increasing in popularity for executive director of the Construction Materials Recycling</p> <p>construction of concrete crushing recycling in la rioja 2</p> <p>Construction of concrete crushing recycling in La Rioja. Concrete crusher; Recycling. construction waste; construction of concrete crushing recycling in</p> <p>Recycling Concrete How to Recycle Concrete The Recycling concrete from demolition project can result in Recycling of concrete pavement is a removing and crushing concrete from an existing pavement</p>	<p>What is the purpose of recycling concrete in construction?</p>	<p>Recycling concrete in construction helps in reutilizing the construction waste into new concrete, preserving landfill space, and reducing construction costs.</p>

### Question 3 : “How many followers has the most followed person on Instagram?”

context	question	answer
string · lengths 1.16k→2.05k 16.3%	string · lengths 48→79 47.5%	string · lengths 2→243 63%
<p>SELENA GOMEZ MOST FOLLOWED PERSON ON INSTAGRAM 🤔🤔</p> <p>🤔</p> <p>Selena Gomez is now the most Followed Person on Instagram with 72.5 million followers (as of this posting) .</p> <p>Former Disney star, Selena Gomez became the most followed person on Instagram, surpassing her best friend, Taylor Swift .</p> <p>No, Selena Gomez is not the most famous person on the planet. She is not even the most famous person with an Instagram account (that would be the those pesky lil Kardashians / Jenner sisters and Taylor Swift ). Most people know her as either the ex-girlfriend of Justin Bieber, or the girl who was on Witches of Waverly Place, or as a pop star whose songs are always playing in their spin class. She comes across as perfectly appealing in interviews, and has been famous for nearly a decade now, and has carved out a very successful career for herself. But none of this quite explains why she would be the most followed person on arguably the most popular social network.</p>	<p>Who is the most followed person on Instagram according to the context?</p>	<p>Selena Gomez is the most followed person on Instagram.</p>

Enfin Question 4 (proche mais s'éloigne tout de même du contexte) :  
 "What are the fashion trends for next fall?"

context	question	answer
string · lengths 2.94k↔3.83k 17.5%	string · lengths 17↔48 6.8%	string · lengths 243↔484 25.6%
<p>Here we have collected some of the trendy short hairstyles for fall. You will see that there are a lot of different splendid hairstyles to choose if you want to create short hair which always looks cool and smashing.</p> <p>For women with curly hair, it is a great idea to create a short bob which looks fabulous yet needs a curl encouraging gel or mousse can to avoid trickiness.</p> <p>Ina addition, the short straight hairstyle also looks fabulous and suit many face shapes. Besides, you can add or leave no bangs according to the hairstyle and your facial shape.</p> <p>Short Fall Hairstyles Trends: Curly Hair /Via Short Fall Hairstyles Trends: Curly Hair</p> <p>The short curly hairstyle is full of fun and charm. To add definition to the hair, you can apply a water-based or silicone gel when the hair is still wet, and let it dry naturally.</p>	<p>What are some trendy short hairstyles for fall?</p>	<p>Some trendy short hairstyles for fall include a short bob for women with curly hair, a short straight hairstyle that suits many face shapes, a light blonde pixie haircut, a classic bob haircut, a straight short bob with the same length, a playful layered razor cut, a casual short hairstyle, a trendy short pixie hairstyle, a short pixie mohawk, a razor cut mingled with a clipper cut, an asymmetrical layered cut, and a simple pixie hairstyle.</p>

On remarque que pour le sujet proche, le LLM ne fait pas la différence entre le fait que l'utilisateur demande des tendances vestimentaires et les informations contenues dans la base de données sur les coupes de cheveux. Cela pourrait être amélioré dans une V2. Le LLM pourrait préciser qu'il donne une réponse présente dans le contexte fourni mais qu'il lui manque une partie de la réponse.

Pour les sujets totalement hors base, le LLM suit les instructions données dans le prompt et ne s'avance pas sur une réponse. Il donne seulement les thématiques proches sémantiquement qu'il a trouvé lors de sa recherche. Ce comportement serait fortement conseillé dans un contexte d'entreprise où les hallucinations des LLMs seraient problématiques.

Enfin j'ai fait tourner le script deux fois : avec un simple retriever et avec un multi-query retriever afin de comparer les résultats. Mon objectif était de vérifier si le multi-query apporterait des réponses plus qualitatives. Les réponses pour les questions en base de connaissance étaient exactement les mêmes. J'ai observé quelques variations pour les questions totalement hors base mais le comportement du LLM était le même ("je ne sais pas d'après le contexte fourni"). En revanche, les temps d'inférence diffèrent fortement. Le multi-query retriever (qui reformule la question de l'utilisateur sous différentes formes) est deux fois plus lent que le retriever simple.

En conclusion, pour un cas simple comme celui de l'exercice, le simple retriever fonctionne parfaitement et est le plus rapide pour répondre. Cependant si la base de connaissances se complexifiait (pour une application future), il pourrait être utile de passer par le multi-query retriever.

Je vous remercie pour votre lecture attentive.