

## **TP : Retrieval Augmented Generation (RAG)**



**Réalisé par :**

DERRA Aguirata  
DIESSONGO Sadath  
GUEYE Halimatu's Sahdiya  
NIAMPA Soumaïla  
MAHDAOUI Rania

**Professeur :**

Imad LAKIM

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Architecture et Choix Techniques</b>	<b>3</b>
2.1	Architecture . . . . .	3
2.2	Choix techniques . . . . .	4
2.2.1	fitz (PyMuPDF) . . . . .	4
2.2.2	spacy pour le NLP . . . . .	4
2.2.3	sentence_transformers (Embeddings) . . . . .	5
2.2.4	langchain (Schema, Vectorstores) . . . . .	5
2.2.5	HuggingFaceEmbeddings et InferenceClient . . . . .	5
2.2.6	FAISS (Facebook AI Similarity Search) . . . . .	5
2.2.7	Chunking sémantique pour diviser les documents en unités logiques . .	5
<b>3</b>	<b>Implémentation &amp; Fonctionnalités</b>	<b>6</b>
3.1	Gestion des données . . . . .	6
3.2	Indexation des documents (Chunking) . . . . .	6
3.3	Recherche documentaire (Embedding - Retrieval) . . . . .	7
3.4	Système de question-réponse (Vectorshore) . . . . .	7
3.5	Analyse des composantes du système . . . . .	8
3.6	Évaluation du système . . . . .	10
3.6.1	Résultats des métriques . . . . .	11
3.6.2	Interprétation des résultats . . . . .	11
<b>4</b>	<b>Perspectives et améliorations</b>	<b>11</b>
<b>5</b>	<b>Conclusion</b>	<b>12</b>

## Table des figures

1	Architecture de la solution . . . . .	4
2	Page de configuration du système . . . . .	8
3	État du système . . . . .	9
4	Gestion des documents . . . . .	10
5	Génération de réponse . . . . .	10

# 1 Introduction

Dans le cadre de ce projet, nous avons conçu et implémenté un système de Retrieval Augmented Generation (RAG), combinant à la fois l'indexation de documents et l'utilisation de modèles de langage pour répondre à des requêtes utilisateur. Le système a été développé pour traiter une variété de documents, tels que des textes liés à la biologie, la géologie, le droit marocain et des rapports financiers de Danone. L'objectif principal était de permettre la récupération d'informations pertinentes à partir de cette base documentaire.

Le projet a été structuré autour de plusieurs étapes clés : la mise en place d'un système d'indexation des documents, l'implémentation d'un processus de recherche documentaire pour interroger efficacement la base vectorielle, et la mise en place d'un système de question-réponse exploitant un LLM open-source. Nous avons utilisé GitHub pour la gestion du projet, assurant ainsi un suivi rigoureux des développements et des versions de notre code source, tout en permettant à notre code reviewer, M. Imad, de suivre l'avancement du travail.

Tout au long de ce projet, une attention particulière a été portée sur la qualité du code, avec une programmation orientée objet (POO) et l'utilisation de technologies comme LangChain pour l'implémentation des fonctionnalités RAG. Ce rapport résume notre travail, justifie nos choix techniques et explique les hypothèses prises lors de la conception du système.

## 2 Architecture et Choix Techniques

L'architecture de notre solution repose sur une approche hybride combinant le traitement sémantique des documents, la recherche vectorielle et l'utilisation d'un modèle de langage pour fournir des réponses contextuelles précises, tout en s'appuyant sur des technologies flexibles et évolutives pour optimiser le processus d'extraction et de génération de contenu.

### 2.1 Architecture

L'architecture de la solution repose sur plusieurs composants clés interagissant pour traiter, analyser et fournir des réponses précises à partir de documents. Cette architecture est construite autour de modèles de traitement du langage naturel (NLP) et de techniques de récupération d'information (RAG - Retrieval-Augmented Generation), permettant de diviser le processus en étapes logiques : le chunking, l'embedding, la récupération et le stockage dans un vector store. Le système est conçu pour ingérer des documents au format PDF, en extraire leur contenu de manière sémantique, puis transformer ce contenu en vecteurs d'empreintes numériques (embeddings). Ces vecteurs sont stockés dans une base de données, permettant une récupération rapide et précise en réponse aux requêtes des utilisateurs. En outre, un modèle de langage est utilisé pour générer des réponses à partir du contexte extrait, tout en intégrant les résultats d'un processus de recherche dans le vector store.

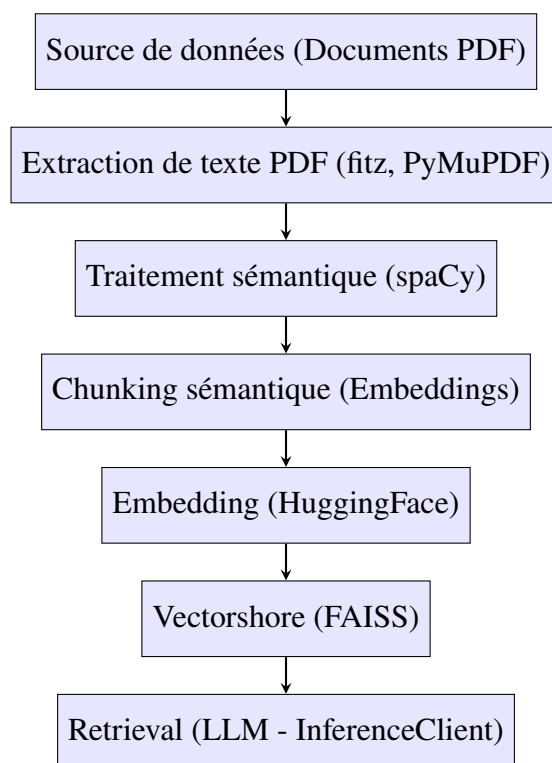


FIGURE 1 – Architecture de la solution

## 2.2 Choix techniques

Dans cette section, nous allons justifier les choix techniques qui ont été faits lors de la conception de l'architecture de notre solution. Chaque décision technique a été prise en fonction des besoins spécifiques du projet, de la nature des données à traiter et des objectifs à atteindre. Chaque choix est expliqué en détail pour mettre en lumière son rôle et son impact sur l'efficacité globale du système.

### 2.2.1 `fitz` (PyMuPDF)

La bibliothèque `fitz` (PyMuPDF) est choisie pour extraire le texte des fichiers PDF. `fitz` est une bibliothèque très performante qui permet de manipuler les documents PDF de manière rapide et efficace. Elle est capable de lire des fichiers PDF structurés et de convertir leur contenu en texte brut pour un traitement ultérieur. D'autres bibliothèques comme `PyPDF2` ou `pdfminer` sont également disponibles, mais `fitz` est préférée ici en raison de sa rapidité et de sa capacité à extraire avec précision le texte, même lorsque le PDF contient des éléments graphiques complexes.

### 2.2.2 `spacy` pour le NLP

`spacy` est une bibliothèque de traitement du langage naturel (NLP) robuste et rapide. Elle permet de traiter des textes, de les analyser linguistiquement et de les diviser en unités syntaxiques (telles que des phrases ou des entités nommées). Elle est largement utilisée pour la création de systèmes NLP professionnels et est privilégiée dans ce projet en raison de ses performances et de sa facilité d'intégration dans des flux de travail complexes. D'autres bibliothèques comme `nltk` ou `TextBlob` pourraient également être utilisées, mais `spacy` offre une meilleure scalabilité et des outils avancés pour l'analyse sémantique.

### 2.2.3 sentence\_transformers (Embeddings)

La bibliothèque `sentence_transformers` est utilisée pour transformer des phrases en embeddings vectoriels. Les embeddings sont des représentations numériques d'un texte dans un espace vectoriel. Cela permet de comparer sémantiquement différentes phrases ou textes. Cette bibliothèque est basée sur des modèles pré-entraînés comme BERT, RoBERTa, et DistilBERT, qui sont parmi les meilleurs modèles pour la création d'embeddings. En utilisant cette bibliothèque, nous pouvons comparer les similarités entre différentes phrases et trouver des relations sémantiques profondes, ce qui est essentiel pour la segmentation et la recherche de similarités dans de grands volumes de texte.

### 2.2.4 langchain (Schema, Vectorstores)

`langchain` est un framework conçu pour intégrer des modèles de langage avec des bases de données et des systèmes de stockage de vecteurs. Il permet de stocker des documents sous forme de *chunks* (blocs de texte) et de créer des *vector stores* pour effectuer des recherches efficaces. Le choix de `langchain` se justifie par sa flexibilité et son intégration facile avec différentes sources de données et modèles de langage, ainsi que sa capacité à gérer de grands volumes de données textuelles de manière évolutive. D'autres solutions comme Haystack ou FAISS peuvent aussi être utilisées, mais `langchain` offre un cadre plus flexible et bien adapté aux besoins de ce projet.

### 2.2.5 HuggingFaceEmbeddings et InferenceClient

Les classes `HuggingFaceEmbeddings` et `InferenceClient` de `langchain_community` permettent de charger des modèles d'embeddings depuis la plateforme Hugging Face et d'effectuer des inférences via une API. Hugging Face est l'un des leaders dans le domaine des modèles pré-entraînés pour le NLP, et leur utilisation permet de bénéficier de modèles de pointe pour le traitement du langage naturel, tout en simplifiant l'intégration avec des systèmes externes. D'autres alternatives comme OpenAI ou Google Cloud AI peuvent être envisagées, mais Hugging Face est un choix populaire pour sa grande communauté et la richesse de ses modèles pré-entraînés.

### 2.2.6 FAISS (Facebook AI Similarity Search)

FAISS est utilisé pour créer un *vector store*, un index de vecteurs permettant de réaliser des recherches efficaces basées sur des similarités. Cela permet de rechercher rapidement des documents ou des morceaux de texte similaires, en utilisant des algorithmes d'indexation adaptés aux grands ensembles de données. Le choix de FAISS se justifie par sa rapidité et sa capacité à gérer de grandes quantités de données en mémoire. Il existe d'autres outils comme Annoy ou HNSW, mais FAISS reste l'un des meilleurs choix pour des recherches de similarité à grande échelle.

### 2.2.7 Chunking sémantique pour diviser les documents en unités logiques

Le *semantic chunking* est une étape cruciale pour diviser un texte volumineux en petites unités qui peuvent être facilement gérées et utilisées pour la recherche. L'algorithme repose sur la similarité des embeddings pour regrouper des phrases similaires et les organiser en chunks logiques. Cette approche permet de conserver le contexte nécessaire pour répondre de manière cohérente aux questions posées. D'autres techniques de chunking, comme le *chunking* basé sur la syntaxe ou la distance lexicale, pourraient être envisagées, mais le *semantic chunking* est

plus adapté à des tâches de recherche sémantique et permet de conserver une compréhension contextuelle fine des documents.

## 3 Implémentation & Fonctionnalités

Afin d'assurer une gestion efficace de nos documents et d'optimiser le processus de traitement, nous avons choisi de structurer nos fichiers de manière systématique. Cette organisation nous permet de faciliter l'accès, le traitement et la recherche d'informations pertinentes au sein des documents.

- **data/** : Contient les documents sources utilisés pour l'indexation et la recherche documentaire.
  - **biologie/**
  - **geologie/**
  - **droit\_marocain/**
  - **rapport\_finance\_danone/**
- **src/**
  - **evaluation/**
    - **evaluation.py**
  - **principal.py** : Gère l'ensemble des fonctionnalités.
- **README.md**
- **cli.py** : Permet d'exécuter le projet depuis le terminal via une ligne de commande.
- **config.yaml**
- **requirements.txt** : Liste des dépendances Python nécessaires au projet.

L'implémentation repose sur la création d'une classe **RAGUEUR** dans le fichier **principal.py**, qui structure les différentes étapes du projet sous forme de méthodes dédiées. Chaque méthode correspond à une phase spécifique du pipeline de gestion et d'exploitation des documents, permettant ainsi une organisation modulaire et évolutive du code. Cette approche facilite la réutilisation du code, l'encapsulation des fonctionnalités et l'amélioration de la lisibilité.

### 3.1 Gestion des données

Ces fonctions permettent de configurer et d'initialiser l'environnement du projet.

- \_\_init\_\_(self, config\_path=None)**
  - Charge la configuration depuis le fichier **config.yaml** via **\_load\_config**.
  - Initialise un modèle NLP (**spaCy**).
  - Initialise un modèle d'embeddings (**HuggingFaceEmbeddings**).
  - Initialise un client LLM (**InferenceClient** de Hugging Face).
- \_load\_config(self, config\_path)**
  - Ouvre le fichier **config.yaml** et charge son contenu.
  - Remplace les variables d'environnement via **\_resolve\_env\_vars**.
  - Applique une configuration par défaut en cas d'échec.
- \_resolve\_env\_vars(self, config)**
  - Vérifie si des variables d'environnement sont utilisées et les remplace par leur valeur système.

### 3.2 Indexation des documents (Chunking)

L'indexation des documents est une étape clé du système RAG. Elle permet de structurer les fichiers textuels afin de faciliter leur exploitation par le moteur de recherche et le modèle de génération de réponses. Le processus débute par le **chargement des documents** depuis un

répertoire spécifié. Chaque fichier est ensuite transformé en texte brut afin de permettre son traitement. L'indexation consiste à extraire le texte des documents PDF et à les découper en segments (chunks).

```
extract_text_from_pdf(self, pdf_path)
— Ouvre un fichier PDF et extrait le texte de chaque page.
— Stocke les métadonnées associées (nom du fichier, source, numéro de page).
— Retourne une liste de dictionnaires contenant le texte et les métadonnées.
semantic_chunking(self, text, metadata)
— Segmente le texte en phrases avec un modèle NLP (spaCy).
— Convertit chaque phrase en vecteur via HuggingFaceEmbeddings.
— Regroupe les phrases ayant une forte similarité cosinus.
— Retourne une liste de documents contenant un chunk de texte et ses métadonnées.
process_pdf(self, pdf_path)
— Utilise extract_text_from_pdf pour extraire le texte.
— Applique semantic_chunking pour segmenter le texte.
— Retourne une liste de chunks prêts à être indexés.
```

### 3.3 Recherche documentaire (Embedding - Retrieval)

La recherche documentaire est une composante essentielle du système RAG, permettant d'extraire rapidement les informations les plus pertinentes à partir des documents indexés.

Chaque document est d'abord converti en une représentation vectorielle à l'aide d'un modèle d'embeddings performant, tel que `all-MiniLM-L6-v2` de Hugging Face. Ces embeddings transforment le texte en vecteurs numériques dans un espace multidimensionnel, facilitant ainsi les comparaisons sémantiques.

Les vecteurs sont ensuite stockés dans une base de données vectorielle spécialisée, ici **FAISS**. Chaque chunk est indexé avec ses métadonnées (nom du fichier, numéro de page, source) pour assurer une traçabilité optimale.

Lorsqu'une requête est formulée, elle est également convertie en embedding, puis comparée aux vecteurs existants. Une mesure de similarité, telle que la **cosine similarity**, permet d'identifier immédiatement les passages les plus pertinents.

Seuls les documents les plus proches du contexte de la requête sont renvoyés avec un score d'affinité, garantissant une recherche précise et fiable. Ce processus assure une **extraction rapide, cohérente et contextuellement pertinente** des informations nécessaires à la génération de réponses par le LLM.

```
get_vector_store(self, chunks)
— Prend une liste de chunks de texte et leurs métadonnées.
— Convertit chaque chunk en vecteur d'embedding.
— Stocke ces vecteurs dans FAISS pour la recherche rapide.
update_vector_store(self)
— Parcourt le répertoire contenant les fichiers PDF.
— Applique process_pdf à chaque fichier.
— Met à jour la base FAISS avec les nouveaux chunks.
```

### 3.4 Système de question-réponse (Vectorshore)

Le système de question-réponse que nous avons développé repose sur une architecture **Vector Store** utilisant FAISS, un moteur de recherche performant permettant de récupérer les documents les plus pertinents en réponse à une question spécifique.



## Fonctionnalités :

- **Chargement de la configuration** (config.yaml) pour définir les modèles et paramètres.
- **Extraction de texte depuis des PDF** avec PyMuPDF (Fitz).
- **Segmentation sémantique des textes** en utilisant spaCy et Sentence Transformers.
- **Création d'un vector store** avec FAISS pour la recherche de documents similaires.
- **Génération de réponses** via un LLM (Llama 3, Mistral 7B, etc.) en s'appuyant sur les documents récupérés.

## Comment ça marche ?

1. **Ajout de documents PDF** → Extraction du texte → Segmentation en chunks → Indexation dans FAISS.
2. **Lorsqu'une question est posée :**
  - Recherche des passages pertinents via FAISS.
  - Construction d'un prompt avec ces passages.
  - Envoi au **LLM** pour générer la réponse.
3. **La réponse est accompagnée des sources** (nom du fichier, numéro de page).

## Avantages :

- Réponses précises basées sur des documents spécifiques.
- Modèle de recherche efficace avec FAISS pour retrouver l'information pertinente.
- Génération naturelle des réponses via un LLM avancé.

## 3.5 Analyse des composantes du système

Etat du système   Gestion des documents   Génération de réponses   **Configuration**

### Configuration du système

	Paramètre	Valeur
0	data_dir	data/droit_Marocain
1	do_sample	True
2	embeddings_model	all-MiniLM-L6-v2
3	llm_model	mistralai/Mistral-7B-Instruct-v0.3
4	max_tokens	500
5	nlp_model	en_core_web_trf
6	repetition_penalty	1.1
7	similarity_threshold	0.6
8	temperature	0.45

### Modifier la configuration

Attention : La modification de la configuration nécessite un redémarrage du système pour prendre effet.

FIGURE 2 – Page de configuration du système


Cette page affiche les paramètres techniques du système RAG :

- Le répertoire de données est `data/droit_Marocain`
- Un échantillonnage est activé (`do_sample : True`)
- Le modèle d'embeddings utilisé est `all-MiniLM-L6-v2`
- Le modèle LLM est `mistralai/Mistral-7B-Instruct-v0.3`
- La limite de tokens est fixée à 500
- Le modèle NLP est `en_core_web_trf`
- D'autres paramètres techniques comme `repetition_penalty` (1.1), `similarity_threshold` (0.6) et `temperature` (0.45) sont également configurés
- Un avertissement indique qu'un redémarrage est nécessaire après modification des paramètres

État du système Gestion des documents Génération de réponses Configuration

## État du système RAG

### Informations générales

	Métrique	Valeur
0	Vectorstore initialisé	 Oui
1	Documents traités	1
2	Chunks créés	292
3	Dernière mise à jour	2025-03-16 12:14:29

Rafraîchir l'état du système

### Visualisation

Ratio Documents/Chunks

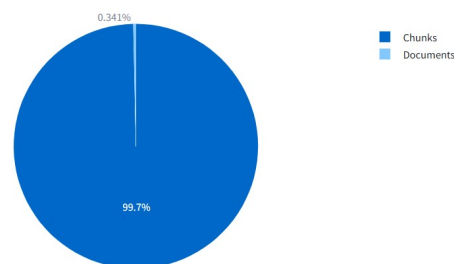


FIGURE 3 – État du système

Cette page montre l'état actuel du système avec des informations générales et une visualisation.

Le **vectorstore** est initialisé, 1 document a été traité, générant 292 chunks (fragments).

La dernière mise à jour date d'aujourd'hui (16 mars 2025) à 12 :14 :29.

Un graphique circulaire montre le ratio entre documents et chunks, avec 99,7% de chunks contre 0,341% de documents, ce qui est logique puisqu'un seul document a généré de nombreux fragments.

Cette page permet de télécharger et gérer des documents. On voit une zone de glisser-déposer pour l'ajout de fichiers PDF (limite de 200 Mo par fichier).

Un seul document existe actuellement dans le système : `Cours Immuno 2019-2020.pdf` d'une taille de 479,45 Ko.

Un bouton "*Mettre à jour le vectorstore*" permet probablement de retraiter les documents pour mettre à jour la base de connaissances.

État du système **Gestion des documents** Génération de réponses Configuration

## Gestion des documents

### Télécharger des documents

Télécharger un fichier PDF

Drag and drop file here  
Limit 200MB per file • PDF

Browse files

### Documents existants

	Nom du fichier	Taille (ko)
0	Cours Immuno 2019-2020.pdf	479.45

### Actions

Mettre à jour le vectorstore

RAGUEUR - Système RAG sur PDF

FIGURE 4 – Gestion des documents

État du système Gestion des documents **Génération de réponses** Configuration

## Génération de réponses

Posez votre question

Quel est le rôle des lymphocytes T dans l'immunité ?

Générer une réponse

### Réponse:

Les lymphocytes T jouent un rôle important dans l'immunité adaptative en reconnaissant les antigènes présents sur les cellules étrangères ou pathogènes, et en activant ou en inhibant les réactions immunitaires appropriées.

### Sources:

- Document: Cours Immuno 2019-2020.pdf, Page: 10
- Document: Cours Immuno 2019-2020.pdf, Page: 16

FIGURE 5 – Génération de réponse

Cette page montre l'interface de question-réponse. Une question a été posée : *"Quel est le rôle des lymphocytes T dans l'immunité ?"*.

Le système a généré une réponse concise expliquant leur rôle dans l'immunité adaptative.

Les sources utilisées sont clairement identifiées comme provenant du document *Cours Immuno 2019-2020.pdf*, pages 10 et 16, démontrant la traçabilité des informations.

### 3.6 Évaluation du système

Pour l'évaluation avec RAGAS, nous avons tout d'abord préparé un jeu de données structuré comprenant des paires question-réponse, des passages récupérés par notre système, et des réponses de référence que nous avons produites. Ce processus implique la sélection de questions représentatives, l'extraction des passages correspondants via notre système de récupération, et notre formulation manuelle de réponses de référence (ground truth). Ces données ont été formatées selon les exigences de RAGAS dans un fichier JSON (*ragas\_dataset\_eval.json*), trans-

formées en dataset Hugging Face, puis soumises à l'évaluation.

Plusieurs métriques clés ont été utilisées : **fidélité**, **pertinence des réponses** et **rappel du contexte**.

Les données sont d'abord extraites d'un fichier JSON et converties en un format compatible avec Ragas, avec des éléments essentiels comme la question, la réponse générée, le contexte et la vérité terrain. Ces données sont ensuite intégrées dans un **dataset Hugging Face** pour une évaluation fluide et efficace.

Les métriques sont appliquées à chaque entrée du dataset :

- **Fidélité** : évalue si la réponse reflète correctement la vérité terrain.
- **Pertinence des réponses** : mesure la correspondance entre la réponse et la question posée.
- **Rappel du contexte** : vérifie si le modèle utilise le contexte pertinent pour formuler sa réponse.

Les résultats sont ensuite analysés et sauvegardés dans un fichier CSV pour permettre une exploitation ultérieure. Ce processus assure une **évaluation précise, robuste et mesurable** des performances du modèle.

### 3.6.1 Résultats des métriques

Voici les résultats obtenus :

- Faithfulness (Fidélité - 0,6333) : Ce score de 63,33% indique une fidélité modérée, suggérant qu'environ un tiers des réponses contiennent potentiellement des hallucinations ou des informations non présentes dans les contextes fournis.
- Answer Relevancy (Pertinence - 0,7955) : Notre score de 79,55% représente la meilleure performance parmi les trois métriques, démontrant que les réponses produites sont généralement bien alignées avec les requêtes des utilisateurs.
- Context Recall (Rappel de contexte - 0,1000) : Le score très faible de 10% révèle une déficience majeure dans notre composant de récupération, qui ne parvient à identifier qu'une fraction minime des passages pertinents.

### 3.6.2 Interprétation des résultats

Ces résultats indiquent que notre système RAG souffre principalement d'une récupération de contexte insuffisante, ce qui limite substantiellement ses performances globales. Bien que le modèle génère des réponses relativement pertinentes avec les informations dont il dispose, la base de connaissances exploitée est trop restreinte.

L'amélioration prioritaire devrait porter sur l'optimisation de notre stratégie de récupération de passages, potentiellement par l'ajustement des embeddings, l'augmentation du nombre de passages récupérés ou la révision de notre algorithme de similarité. Des améliorations secondaires pourraient viser à renforcer la fidélité des réponses en optimisant les prompts pour encourager le modèle à s'appuyer plus strictement sur les données fournies.

## 4 Perspectives et améliorations

Le projet de conception et d'implémentation du système RAG a permis de développer une solution robuste pour la récupération d'informations et la génération de réponses contextuelles à partir de documents variés. Toutefois, plusieurs axes d'amélioration peuvent être envisagés pour optimiser et étendre les capacités du système :

## **Amélioration de la gestion des documents volumineux**

Actuellement, le système traite efficacement des documents de taille modérée, mais la gestion de documents particulièrement volumineux pourrait être optimisée. Une solution pourrait consister à mettre en place un mécanisme de traitement par lot ou de découpage dynamique des documents en unités plus petites, afin d'éviter des problèmes de mémoire ou de latence lors de l'analyse de fichiers de grande taille.

## **Amélioration du processus de Chunking**

Bien que le chunking sémantique offre une solution efficace pour diviser les documents en unités logiques, il pourrait être enrichi par des techniques plus avancées de segmentation contextuelle. L'intégration d'algorithmes de segmentation dynamique en fonction des requêtes de l'utilisateur permettrait de mieux adapter le découpage en fonction des informations recherchées, réduisant ainsi le nombre de résultats erronés.

## **Optimisation des embeddings**

Le modèle utilisé pour générer les embeddings (sentence-transformers) pourrait être optimisé par le fine-tuning de modèles préexistants sur des corpus spécifiques, afin d'améliorer la précision des réponses dans des domaines spécialisés (par exemple, biologie ou droit marocain). L'utilisation de modèles multilingues pourrait aussi enrichir le système pour traiter des documents dans d'autres langues, en particulier pour des applications internationales.

## **Intégration de systèmes de feedback utilisateur**

Le système pourrait bénéficier d'une fonctionnalité de feedback utilisateur permettant d'ajuster en temps réel les résultats fournis. En intégrant un mécanisme d'apprentissage supervisé basé sur les corrections apportées par les utilisateurs, il serait possible d'améliorer les résultats au fur et à mesure de l'utilisation du système.

# **5 Conclusion**

En conclusion, ce projet de développement d'un système de Retrieval Augmented Generation (RAG) a permis de concevoir une solution robuste pour la gestion et la récupération d'informations à partir de documents variés. L'intégration de techniques avancées telles que l'indexation sémantique, l'utilisation de modèles de langage, et la mise en place de processus de recherche vectorielle a permis de répondre efficacement aux exigences de notre projet. Les choix techniques, notamment l'utilisation de bibliothèques comme `fitz`, `spaCy`, et `langchain`, ont été essentiels pour garantir une gestion optimale des documents et des requêtes utilisateur.

L'architecture mise en place, combinant une approche hybride de traitement sémantique et de recherche vectorielle, a prouvé sa pertinence en termes de performance et de scalabilité, notamment pour la gestion de grands volumes de données textuelles. Le système RAG développé a démontré son efficacité dans l'extraction d'informations pertinentes et la génération de réponses contextuelles, répondant ainsi aux besoins exprimés par le projet.

Enfin, ce travail a permis d'acquérir une expérience significative dans l'intégration de technologies modernes et flexibles pour le traitement du langage naturel et la gestion documentaire. Les résultats obtenus ouvrent la voie à des améliorations futures, notamment en termes de précision des réponses et de gestion de données encore plus volumineuses.