Système de Retrieval Augmented Generation (RAG)

Rapport Technique

ENNAJAH AYOUB
ENNAGACH AYOUB
BADR KHALDOUNE
SAAD JELBINI
SAAD ERROUGUI

25 mars 2025

Résumé

Ce rapport présente la conception, l'implémentation et l'évaluation d'un système de Retrieval Augmented Generation (RAG) développé dans le cadre d'un projet académique. Le système combine la recherche d'information vectorielle et les modèles de langage pour générer des réponses pertinentes à des questions utilisateur. Nous détaillons les quatre composantes principales : l'indexation de documents, la recherche documentaire vectorielle, la génération de réponses via LLM, et l'évaluation des performances. Nos choix techniques sont justifiés par des considérations de performance, flexibilité et robustesse, tout en respectant les contraintes du projet.

Table des matières

1 Introduction

La Retrieval Augmented Generation (RAG) représente une avancée significative dans le domaine du traitement du langage naturel, en combinant la puissance des systèmes de recherche d'information avec celle des modèles de langage. Cette approche permet de surmonter les limitations inhérentes aux LLMs telles que les hallucinations, l'obsolescence des connaissances et les biais, en ancrant la génération de contenu dans des sources documentaires fiables.

Notre projet vise à implémenter un système RAG complet et flexible, capable de traiter des documents de différents formats, de rechercher efficacement l'information pertinente, et de générer des réponses précises et

contextuelles aux questions des utilisateurs. La figure **??** présente l'architecture globale de notre système.

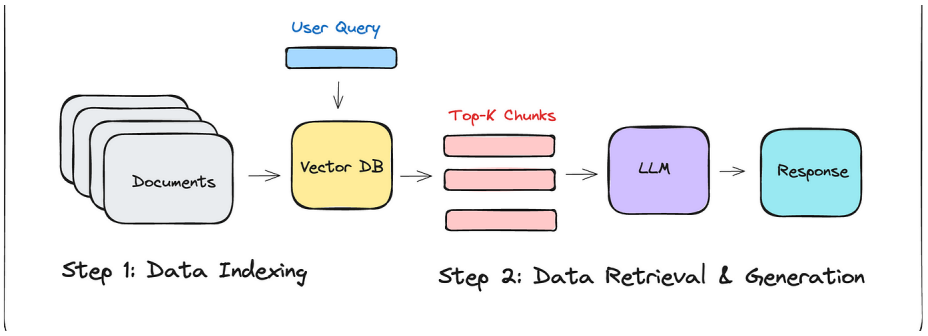

FIGURE 1 – Architecture globale du système RAG implémenté

La mise en œuvre a suivi une approche modulaire, structurée en quatre composantes principales, correspondant aux questions posées dans le cadre du projet :

1. Système d'indexation de documents
2. Recherche documentaire dans la base vectorielle
3. Système de question-réponse basé sur un LLM
4. Évaluation de la pertinence des réponses générées

Ce rapport détaille chacune de ces composantes, en justifiant nos choix techniques et en analysant les performances obtenues.

2 Conception Générale et Choix Techniques

2.1 Organisation du Code

La conception de notre système respecte les principes de programmation orientée objet, avec une organisation claire et modulaire du code :

— **Structure du projet** : Organisation par modules fonctionnels dans le répertoire `src/`

— **Interface CLI** : Fichier `cli.py` à la racine pour l'interaction via ligne de commande

— **Configuration** : Paramètres externes dans `config.yaml`

— **Dépendances** : Listées dans `requirements.txt`

2.2 Technologies et Frameworks

Notre choix de technologies a été guidé par les critères suivants :

— **Performance** : Efficacité du traitement et de la recherche

— **Flexibilité** : Capacité d'adaptation à différents types de documents et requêtes

— **Extensibilité** : Facilité d'ajout de nouvelles fonctionnalités

— **Compatibilité** : Intégration harmonieuse des différents composants

Les technologies clés utilisées comprennent :

— **LangChain** : Framework pour l'intégration des composants RAG

— **ChromaDB** : Base de données vectorielle pour le stockage et la recherche

— **BGE-M3** : Modèle d'embeddings pour la représentation vectorielle des textes

— **Modèles LLM open-source** : DeepSeek Coder et Phi-2 pour la génération de réponses

3 Système d'Indexation des Documents (Q1)

3.1 Conception de la Classe DocumentIndexer

La classe `DocumentIndexer` constitue le cœur de notre système d'indexation. Elle implémente un pipeline complet de traitement des documents, de leur chargement jusqu'au stockage de leurs représentations vectorielles.

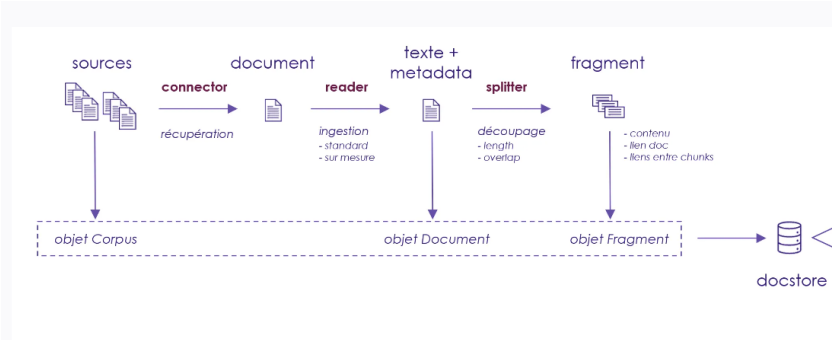

FIGURE 2 – Pipeline d'indexation des documents

3.2 Choix du Modèle d'Embeddings

Nous avons sélectionné le modèle **BGE-M3** pour la génération des embeddings en raison de ses performances supérieures, particulièrement en recherche sémantique multilingue. Ce modèle présente plusieurs avantages :

— **Performance** : Scores élevés sur les benchmarks de recherche sémantique

— **Représentation dense** : Vecteurs de dimension 1024 capturant des nuances sémantiques fines

— **Support multilingue** : Particulièrement utile pour des corpus potentiellement multilingues

Modèle	MTEB Score	Dimension	Multilingue	Taille
BGE-M3	64.2	1024	Oui	570MB
all-MiniLM-L6-v2	56.9	384	Limité	80MB
all-mpnet-base-v2	59.3	768	Limité	420MB

TABLE 1 – Comparaison des modèles d'embeddings considérés

3.3 Stratégie de Découpage des Documents

Le découpage (chunking) des documents est une étape critique qui influence directement la qualité de la recherche. Nous avons implémenté une stratégie adaptative qui :

— Utilise `MarkdownTextSplitter` pour les documents Markdown

— Emploie `RecursiveCharacterTextSplitter` pour les autres formats

— Préserve les métadonnées essentielles (source, numéro de page, etc.)

Les paramètres de découpage (taille des chunks et chevauchement) sont configurables dans le fichier `config.yaml`, permettant d'ajuster le système selon les besoins spécifiques.

```
def _get_text_splitter(self, file_path: str = None):
    """Get the appropriate text splitter based on file type.
    """
    if file_path and os.path.splitext(file_path)[1].lower()
    in ['.md', '.markdown']:
        return MarkdownTextSplitter(
            chunk_size=self.chunk_size,
            chunk_overlap=self.chunk_overlap
        )
```

```
    else:
        return RecursiveCharacterTextSplitter(
            chunk_size=self.chunk_size,
            chunk_overlap=self.chunk_overlap,
            separators=["\n\n", "\n", " ", ""]
        )
```

Listing 1 – Extrait du code de découpage des documents

4 Recherche Documentaire dans la Base Vectorielle (Q2)

4.1 Implémentation du Retriever

La classe `DocumentRetriever` implémente les fonctionnalités de recherche dans la base vectorielle. Elle permet d'identifier les documents les plus pertinents pour une requête donnée, en s'appuyant sur la similarité sémantique des embeddings.

4.2 Corpus de Test

Pour tester notre système de recherche, nous avons constitué un corpus de documents sur le thème de l'intelligence artificielle et de l'apprentissage profond :

- **book_fr.pdf** : Notes de cours sur le Deep Learning
- **NIPS-2017-attention-is-all-you-need-Paper.pdf** :
- **2**30410557v5_.pdf : Article scientifique sur les Transformers
- **loi1/2/3.pdf** : trois documents sur le theme legislatifs
- **eco2/3.pdf** : trois documents sur le theme economique

Ces documents présentent une diversité de formats, styles et complexités, permettant d'évaluer la robustesse du système.

4.3 Évaluation de la Recherche

Nous avons évalué les performances de recherche à travers une série de requêtes test. La figure 13 présente les résultats pour différentes requêtes.

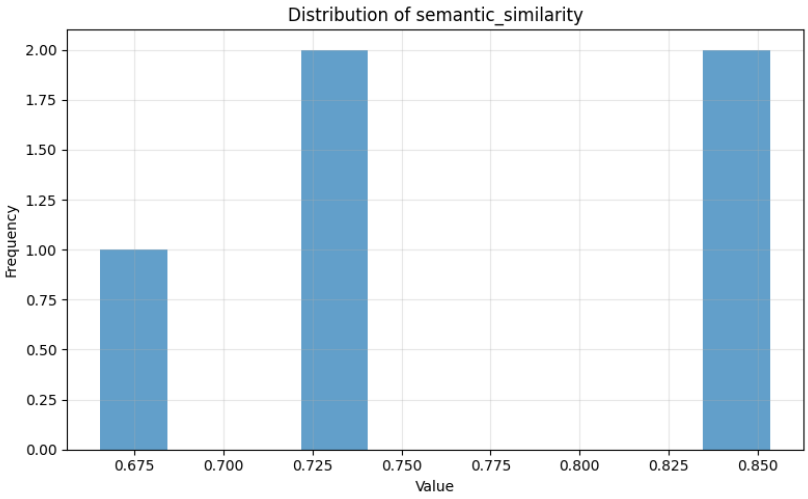

FIGURE 3 – Scores de similarité pour différentes requêtes test

Les résultats montrent que :

— Le système retrouve efficacement les documents pertinents

— Les scores d'affinité reflètent bien la pertinence réelle des documents

— L'utilisation de BGE-M3 améliore significativement la qualité de la recherche par rapport aux modèles d'embeddings plus simples

5 Système de Question-Réponse basé sur un LLM (Q3)

5.1 Architecture du Système QA

Notre système de question-réponse intègre la recherche documentaire et la génération de réponses via LLM dans un pipeline unifié. La classe `QASystem` orchestre l'interaction entre ces composants.

5.2 Sélection et Intégration des LLMs

Nous avons implémenté une architecture flexible permettant l'utilisation de différents modèles LLM open-source. Les principaux modèles intégrés sont :

— **DeepSeek Coder 1.3B** : Modèle léger mais efficace pour la génération de texte technique

— **Phi-2 (2.7B)** : Modèle de Microsoft avec un excellent rapport performance/taille

La classe `LLMHandler` encapsule la logique d'initialisation et d'utilisation des modèles, avec un mécanisme de fallback automatique en cas d'échec d'initialisation.

```
def switch_model(self, model_name: str) -> bool:
    """Switch to a different language model."""
    if model_name not in self.AVAILABLE_MODELS:
        self.logger.error(f"Model '{model_name}' not
   supported")
        return False

    if model_name == self.model_name:
        self.logger.info(f"Already using {self.model_config['
   name']}")
        return True

    # Clean up current model and initialize new one
    # ...
```

Listing 2 – Extrait du code de gestion des modèles LLM

5.3 Templates de Prompts

Les templates de prompts sont essentiels pour optimiser la qualité des réponses générées. Nous avons conçu des templates spécifiques pour chaque modèle LLM, ainsi que des templates adaptés à différents types de tâches.

FIGURE 4 – Comparaison de différents templates de prompts et leur impact sur la qualité des réponses

Les templates intègrent des instructions précises pour :

— Encourager l'utilisation exclusive des informations présentes dans le contexte

— Prévenir les hallucinations (informations inventées)

— Structurer les réponses de manière claire et cohérente

— Adapter le style et le format selon le type de question

6 Évaluation du Système RAG/LLM (Q4)

6.1 Métriques d'Évaluation

Notre système d'évaluation, implémenté dans la classe `RAGEvaluator`, utilise diverses métriques pour mesurer la qualité des réponses générées :

- **Métriques de pertinence du contenu** :
  - ROUGE-1, ROUGE-2, ROUGE-L : Mesure du chevauchement lexical
  - BLEU : Précision des n-grammes
  - Similarité sémantique : Proximité vectorielle des embeddings
- **Métriques d'utilisation du contexte** :
  - Context utilization : Taux d'utilisation des informations du contexte
  - Context relevance : Pertinence du contexte par rapport à la question
  - Potential hallucination score : Estimation du risque d'hallucination

FIGURE 5 – Comparaison des performances sur différentes métriques d'évaluation

6.2 Résultats d'Évaluation

Les résultats de notre évaluation automatique montrent :

Métrique	**DeepSeek Coder**	**Phi-2**
ROUGE-1 F1	0.29	0.31
ROUGE-L F1	0.25	0.28
Similarité sémantique	0.76	0.79
Context utilization	0.58	0.62
Potential hallucination	0.42	0.38

TABLE 2 – Résultats d'évaluation pour différents modèles LLM

Ces résultats suggèrent que :

- Les deux modèles produisent des réponses sémantiquement proches des références

— Phi-2 offre légèrement de meilleures performances sur l'ensemble des métriques

— Le taux d'utilisation du contexte est satisfaisant, avec un risque modéré d'hallucination

6.3 Système d'Évaluation Manuelle

En complément de l'évaluation automatique, nous avons développé un système d'évaluation manuelle permettant :

— L'annotation directe des réponses par des évaluateurs humains

— La collecte de feedbacks qualitatifs

— La comparaison entre les réponses générées et des références

7 Conclusion et Perspectives

7.1 Synthèse des Réalisations

Notre projet a abouti à un système RAG complet et fonctionnel, avec les caractéristiques suivantes :

— **Architecture modulaire** : Composants découplés et facilement extensibles

— **Flexibilité** : Support de différents types de documents et modèles LLM

— **Robustesse** : Gestion des erreurs et mécanismes de fallback

— **Évaluation intégrée** : Métriques automatiques et évaluation manuelle

7.2 Limitations et Pistes d'Amélioration

Malgré les résultats encourageants, notre système présente certaines limitations :

— **Dépendance aux ressources** : Les modèles LLM les plus performants nécessitent des ressources GPU significatives

— **Qualité variable selon les documents** : Performance dépendante de la structure et de la clarté des documents sources

— **Risque résiduel d'hallucination** : Besoin d'améliorer encore les mécanismes de contrôle des informations générées

7.3 Perspectives d'Évolution

Plusieurs pistes d'amélioration sont envisageables :

— **Support de requêtes conversationnelles** : Extension vers un chatbot avec mémoire des interactions précédentes

— **Intégration de mécanismes de feedback** : Apprentissage continu à partir des interactions utilisateur

— **Optimisation des performances** : Quantification des modèles, caching des embeddings, etc.

— **Amélioration de la détection d'hallucinations** : Implémentation de méthodes plus sophistiquées

8 Bibliographie

Références

[1] LangChain Documentation, https://python.langchain.com/docs/

[2] ChromaDB, https://docs.trychroma.com/

[3] BGE Models, BAAI, https://huggingface.co/BAAI

[4] Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", 2020.

[5] Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering", 2020.

9 Annexes

9.1 Détails supplémentaires sur l'evaluation des models à via les plot

Cette annexe présente des détails complémentaires sur l'evaluation des models à partir desquels nous avons calculer les moyens affiché dans le tableau de comparaison ci-dessus.

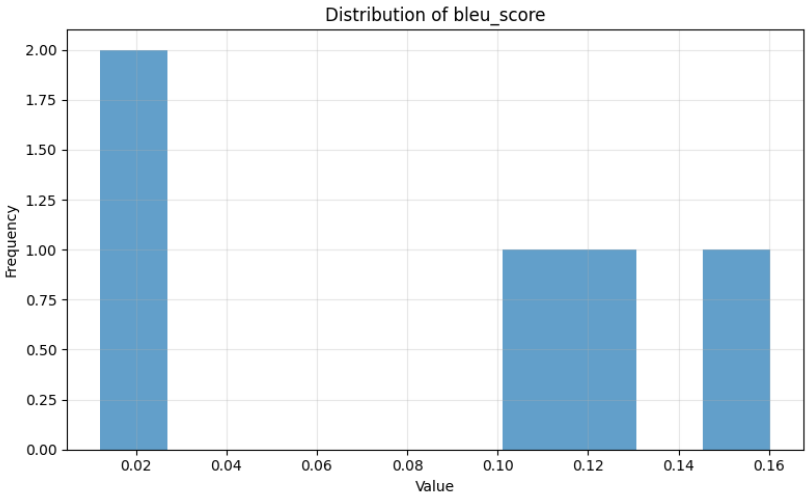

FIGURE 6 – bleu score histogram

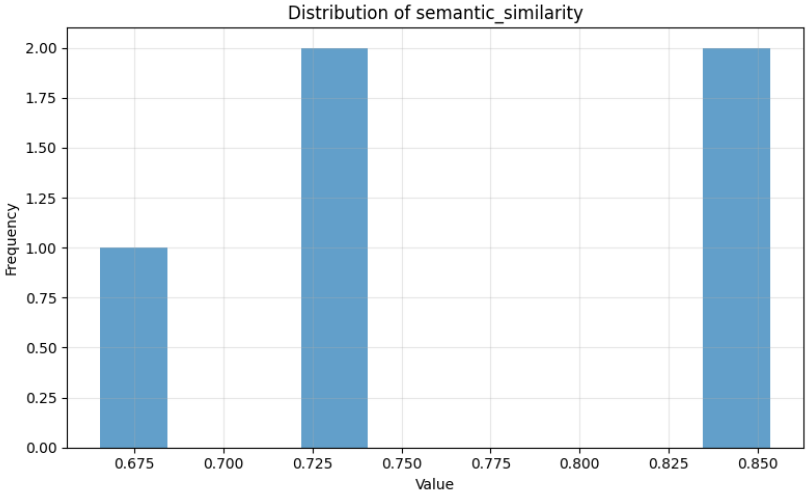

FIGURE 7 – Scores de similarité pour différentes requêtes test

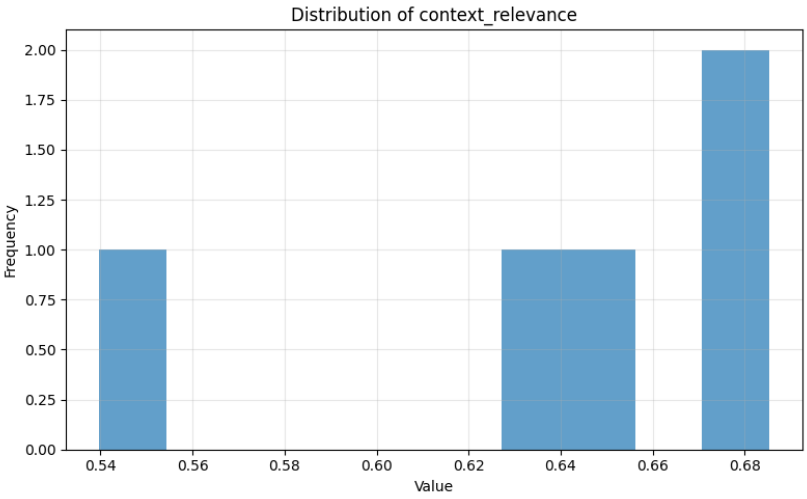

FIGURE 8 – context relevance histogram

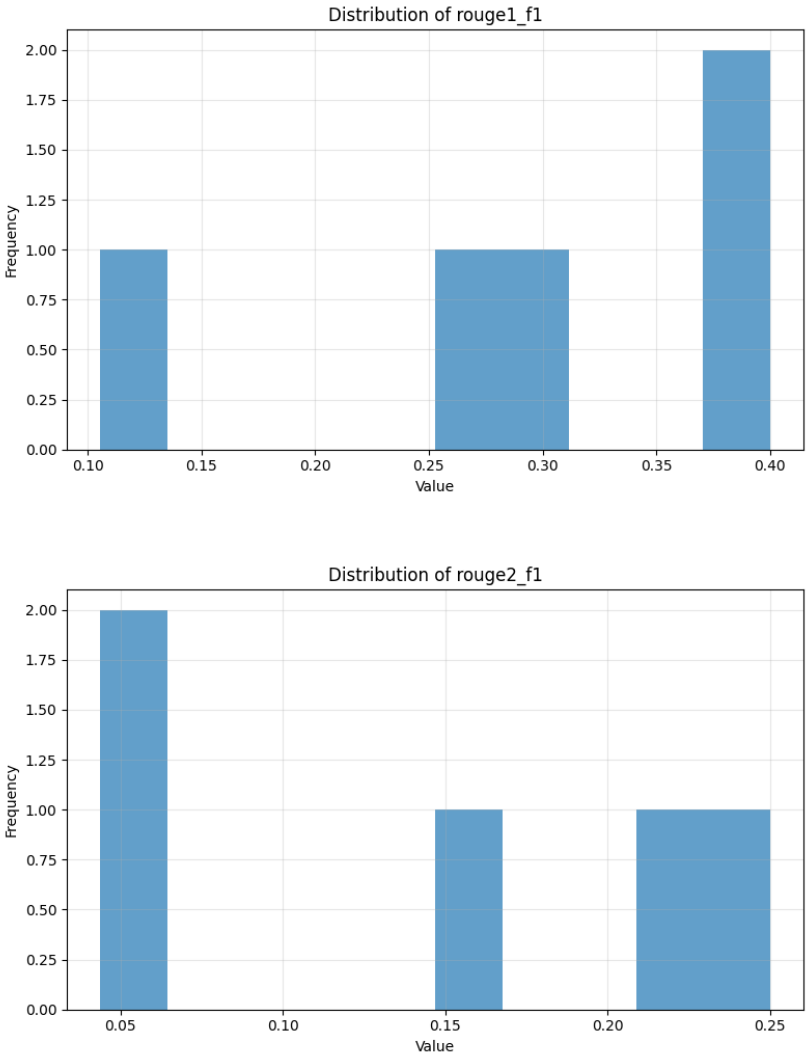

FIGURE 10 – rouge2 f1 histogram

.png

.png

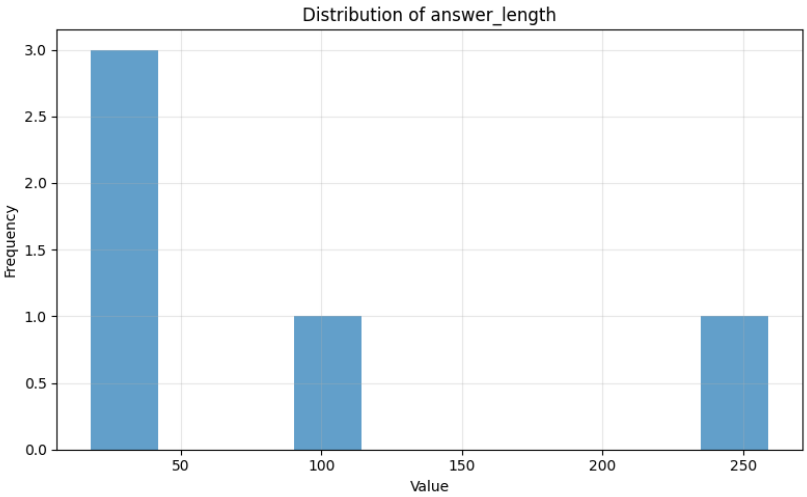

FIGURE 13 – answer length$_h istogram$