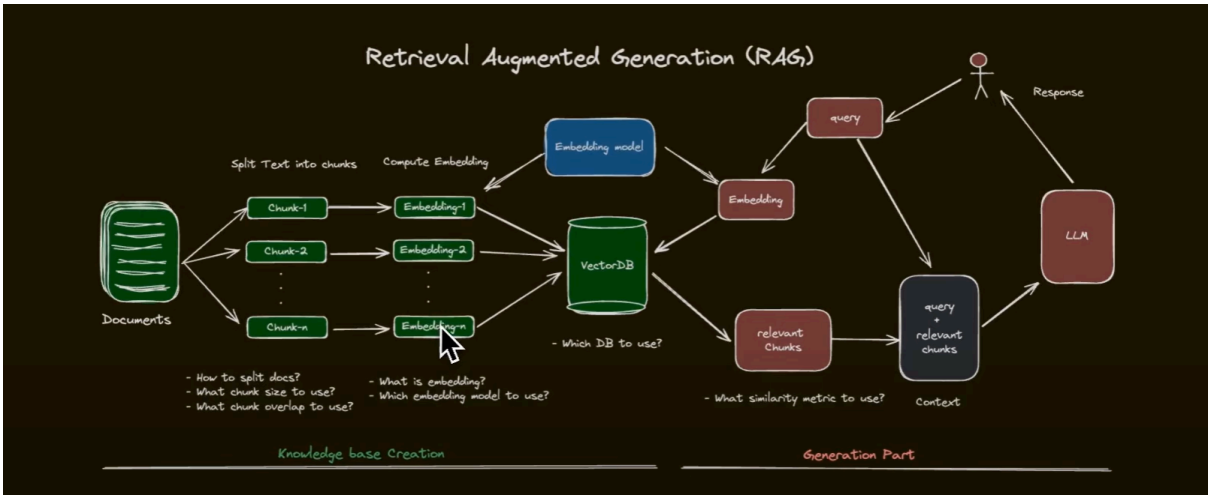
RAPPORT NLP

Implémentation d'un chatbot avec un système de RAG



IA A
REN Simon
AGNESA Ritchy
GUENNEAU Romain
HASSANI Abdellah

2024/2025

Introduction

Dans ce projet, nous nous concentrons sur l'implémentation d'un chatbot utilisant des mécanismes de RAG avancés. Plus précisément, nous explorons plusieurs techniques de pointe telles que le late chunking, ColBERTv2, la recherche hybride (hybrid search) et le reranking. Ces techniques peuvent être appliquées à différents datasets, notamment dans les domaines du cinéma et du droit, pour évaluer leur impact sur la performance du système de retrieval et, par extension, sur la qualité des réponses générées.

Pour le modèle de langage utilisé (LLM), nous avons choisi d'utiliser l'API de Mistral qui est gratuite et open source mais également car la puissance de calcul locale dont nous disposions était très limitée, cela nous a permis de mieux nous concentrer sur les différentes techniques de RAG .

1.Benchmark des méthodes de RAG

1.1 Les RAG

Le Retrieval-Augmented Generation (RAG) est une approche qui combine la recherche d'information (retrieval) et la génération de texte (generation) pour améliorer les réponses des chatbots. Elle permet de retrouver des informations pertinentes à partir de vastes corpus de données et de les intégrer dans la génération de réponses, augmentant ainsi la précision et la pertinence des interactions. Cette méthode est particulièrement utile pour des applications nécessitant des réponses factuelles et contextuellement appropriées. Il existe de nombreuses méthodes de RAG différentes, chacune avec ses avantages.

1.2 Late Chunking

- **Principe** :
  - Contrairement à un découpage précoce des documents (early chunking), le late chunking consiste à diviser le contenu des documents en morceaux **plus tard dans le pipeline** de récupération, plutôt qu'avant l'indexation.
  - Ce mécanisme permet de garder une granularité fine tout en adaptant le découpage en fonction de la requête ou des besoins spécifiques d'un modèle.
- **Avantages** :
  - Réduction de la perte d'information contextuelle grâce à un traitement global des documents avant le découpage.
  - Amélioration des résultats pour des tâches où le contexte global est crucial (comme les questions nécessitant des informations inter-paragraphes).
- **Applications** :
  - Scénarios avec documents complexes où des dépendances inter-sections sont importantes.

1.3 ColBERTv2

- **Principe** :
  - **ColBERTv2** (Contextualized Late Interaction Over BERT) est une version améliorée de ColBERT, un modèle de recherche dense qui combine l'efficacité des approches d'encodage dense avec des interactions tardives entre les représentations de la requête et des documents.
  - ColBERTv2 encode séparément les requêtes et les documents à l'aide de BERT mais effectue une interaction fine au niveau des tokens lors de l'étape de recherche.
  - Les indexes sont stockés à l'emplacement : .ragatouille/colbert/indexes/
- **Avantages** :
  - Performances proches de l'état de l'art sur des benchmarks de recherche dense.

    - Convient aux bases de données volumineuses grâce à son efficacité computationnelle.
- **Applications** :
    - Recherche dense dans des systèmes où la qualité des correspondances textuelles doit être très élevée.

1.4 Hybrid Search

- **Principe** :
    - La recherche hybride combine deux types d'approches :
        - **Recherche vectorielle dense** : utilise des représentations vectorielles continues pour capturer le sens sémantique global des textes.
        - **Recherche lexicale sparse** : basée sur des techniques traditionnelles comme TF-IDF ou BM25 pour capturer des correspondances lexicales exactes.
    - En intégrant ces deux méthodes, la recherche hybride maximise la couverture des résultats pertinents.
- **Avantages** :
    - Précision accrue, car elle compense les limitations respectives de chaque méthode :
        - La recherche dense peut manquer de correspondances lexicales précises.
        - La recherche sparse peut échouer à capter le sens sémantique implicite.
    - Résultats plus robustes, en particulier pour des requêtes complexes ou ambiguës.
- **Applications** :
    - Recherche dans des bases documentaires où la diversité des formulations est élevée.
    - Questions ouvertes ou reformulées de manière inhabituelle.

1.5 Reranking

- **Principe** :
    - Le reranking consiste à réorganiser une liste initiale de documents ou de passages obtenus à l'aide d'une méthode de recherche primaire (dense ou sparse).
    - Utilise souvent des modèles plus complexes ou coûteux comme **BERT** ou ses variantes pour recalculer la pertinence de chaque item.
- **Étapes** :
    - **Étape initiale** : Une méthode rapide génère une liste courte de candidats (top-k retrieval).
    - **Reranking** : Un modèle sophistiqué réévalue les scores de pertinence en tenant compte du contexte global et des nuances lexicales.
- **Avantages** :
    - Amélioration significative de la précision des résultats finaux.

    - Exploitation des capacités de modèles pré entraînés pour mieux comprendre les relations sémantiques.
- **Applications** :
    - Systèmes de recherche nécessitant des résultats hautement précis (moteurs de recherche, assistants virtuels).
    - Cas où le premier niveau de récupération ne suffit pas à répondre précisément à la requête.

2. Méthodologie de travail

Après s'être renseignés sur les différentes méthodes de RAG, nous avons défini l'objectif de notre projet : construire une application de chatbot intégrant les différentes méthodes de RAG avec la possibilité de les utiliser sur n'importe quel fichier.

Nous nous sommes répartis le travail de la manière suivante :

- Romain travaillait sur le développement et l'architecture de l'application avec la bibliothèque streamlit pour créer une interface de discussion avec un llm sans méthode de RAG au départ.
- Ritchy devait implémenter sur un notebook google colab les méthodes de late chunking et hybrid search.
- Simon devait implémenter de son côté la méthode de rerank.
- Abdellah devait implémenter la méthode ColBERTv2.

Nous avons fait des réunions régulièrement (au maximum tous les deux jours) pour discuter de l'avancée de chacun et discuter des différentes étapes à suivre.

Les différentes tâches réparties ont finalement été mises en commun pour créer l'application avec les méthodes de RAG.

3. Étapes clés du projet

A. Les premières implémentations

Afin de faciliter la prise en main et l'expérimentation des différentes méthodes de RAG, nous les avons d'abord implémentées séparément sur des environnements cloud (Google Colab). Pour cela, nous nous sommes concentrés sur un seul des datasets proposés au départ : Wikipedia Movie Plots. Ce dernier étant riche et volumineux, il constituait une bonne première approche pour tester les techniques de RAG.

Ainsi, nous avons implémenté le Reranking pour améliorer les réponses générées par le LLM avec l'API de mistral "mistral-large-latest". Pour le modèle d'embedding, nous

avons pu essayé divers modèles comme “mistral-embed” mais nous avons finalement opté pour “all-MiniLM-L6-v2” qui s’est avéré être le plus adapté.

L'intégration de ColBERTv2, basé sur le modèle pré entraîné "colbert-ir/colbertv2.0" de la librairie Ragatouille, a permis une recherche dense efficace, grâce à l'utilisation d'embeddings binaires compressés et de méthodes de scoring parallélisées, offrant donc des résultats de grande qualité avec des informations riches. Cependant, cette méthode exige des ressources de calcul significatives, ce qui peut poser des défis dans le traitement de très grands datasets.

Pour étendre davantage les possibilités du système, nous avons combiné le Hybrid Search au Late Chunking. Cette combinaison exploite la puissance des méthodes lexicales (BM25) et sémantiques (modèle d’embedding “all-MiniLM-L6-v2”) pour le retrieval tout en permettant de traiter efficacement les documents longs grâce au Late Chunking.

B. Le développement de l’application

Pour la création du chatbot, nous avons dans un premier temps essayé de faire tourner un LLM en local (Mistral 7B) en utilisant la bibliothèque streamlit afin de communiquer avec l’API. L’implémentation était correcte mais assez lente dans l’exécution car le modèle est lourd et ne tourne pas très bien sur des cartes graphiques anciennes voire encore pire sur des ordinateurs sans carte. Nous sommes donc passés sur l’API de Mistral donnant même accès au modèle 70B encore plus performant.

L’interface comprend une boîte de dialogue pour envoyer une requête à la bdd puis au LLM, un historique des réponses, un bouton pour upload le dataset à fournir au système, un bouton pour lancer l’embedding des documents ainsi que 3 boutons pour sélectionner le modèle RAG à utiliser.

Une fois que le chatbot avec le LLM était fonctionnel, nous avons essayé d’implémenter une solution d’embedding avec chromaDB qui s’est avérée très lourde en calcul et qui a donc été abandonnée au profit d’autres solutions.

C. Finalisation de l’application

Après des premiers résultats concluants sur Colab, nous avons décidé d’implémenter les différentes méthodes directement dans l’application finale. Cette phase a impliqué l’intégration des techniques de retrieval, de traitement des données et de génération dans une interface utilisateur conviviale, permettant des interactions intuitives avec le système RAG.

L’application combine ainsi les méthodes explorées tout au long du projet sélectionnables via des boutons dans l’interface, permettant de tester différents systèmes de RAG et de

traiter des requêtes sur des datasets plus ou moins volumineux. Les dataset peuvent être upload grâce à un bouton.

Pour simplifier le déploiement et l'utilisation de l'application, un script colab a également été créé, automatisant l'ensemble du processus de lancement. Il permet notamment de cloner le dépôt Github, installe les dépendances et lance l'application en local grâce à ngrok.

4. Idées d’amélioration

Pour finir, voici quelques idées pour améliorer notre projet et développer un chatbot avec des systèmes de RAG avancés qui soit performant et pratique.

1. **Upload multiple de documents** : Nous aimerions ajouter une fonctionnalité pour charger plusieurs documents simultanément et donner différentes ressources au LLM pour générer ses réponses. Cela éviterait également aux utilisateurs de répéter le processus d'upload pour chaque fichier, rendant l'utilisation plus rapide et moins fastidieuse.

2. **Support des PDF non interactifs** : Nous souhaiterions intégrer la reconnaissance optique de caractères (OCR) pour permettre l'extraction de texte à partir de PDF scannés. Cette fonctionnalité aiderait à traiter des documents qui sont actuellement inutilisables par notre application.

3. **Test approfondi des RAG** : Nous voudrions mettre en place un protocole de test détaillé pour évaluer la précision de notre générateur de réponses. Cela nous permettrait de détecter les faiblesses de notre système et de déterminer quelles sont les méthodes les plus efficaces. Nous pourrions par exemple créer une batterie de questions auxquelles le chatbot devra répondre et faire évaluer ces réponses par des retours utilisateurs (note, remarque…).

4. **Pré-chargement des modèles comme ColBERT** : Pour réduire le temps d'attente lors de la première utilisation, il faudrait implémenter un système de pré-chargement des modèles. Cela devrait considérablement diminuer le délai avant l'utilisation effective de l'application.

5. **Interface modulable** : Il serait également souhaitable de travailler sur une interface utilisateur plus flexible, où les utilisateurs pourraient personnaliser l'organisation avec différentes sessions et chats ou avoir également la possibilité de modifier les documents dans l’interface avant de les fournir au LLM. Cette modification rendrait l'application plus intuitive et adaptée à différents types d'utilisateurs.

Ces changements sont destinés à améliorer l'expérience globale, à rendre l'application plus utile et à répondre mieux aux attentes des utilisateurs.