Réalisation d’une application de type RAG

**      **

Certification : Développeur d’application en Intelligence Artificiel

Marseille – Nice

Bloc de compétences 2 - E2

Intégrer des modèles et des services d’intelligence artificielle

LOMBARDI Joachim

Table des matières

[A3. Compétence : Intégration de services d'intelligence artificielle préexistants 3](#_Toc193101035)

[C6. Veille Technologique et Réglementaire 3](#_Toc193101036)

[C7. Identification et Benchmark de Services d'IA 4](#_Toc193101037)

[A. Choix moteurs de recherche 4](#_Toc193101038)

[B. Utilisation d’elasticsearch 7](#_Toc193101039)

[C. Génération de la réponse 10](#_Toc193101040)

[C8. Paramétrage et Intégration d'un Service d'IA 13](#_Toc193101041)

[Utilisation du LLM 13](#_Toc193101042)

[Conclusion 15](#_Toc193101043)

# A3. Compétence : Intégration de services d'intelligence artificielle préexistants

## C6. Veille Technologique et Réglementaire

La veille a été réalisée au fur et à mesure des besoins du projet et est présente avant l’utilisation de chaque technologie. Les sources utilisées sont principalement les sites des fournisseurs d'outils, ainsi que les sites les plus utilisés par les développeurs comme Hugging Face, GitHub, Google (pour la recherche), StackOverflow ou medium.

**1. Hugging Face**

Hugging Face est une plateforme spécialisée dans les technologies de traitement du langage naturel (NLP) et l'intelligence artificielle. Elle propose une large gamme de modèles pré-entrainés pour diverses tâches, telles que la traduction, la génération de texte, l’analyse de sentiments, la classification, etc. Hugging Face héberge également un hub de modèles où les chercheurs et développeurs peuvent partager et découvrir des modèles d'IA. Leur bibliothèque Transformers est l'une des plus populaires pour le traitement du langage avec des modèles comme GPT et BERT.

**2. GitHub**

GitHub est une plateforme de développement collaboratif qui permet aux développeurs de gérer et de partager leur code source. C’est l'un des sites les plus utilisés pour les projets open-source. Il permet de créer des répertoires (repos) pour organiser les projets, d'utiliser des branches pour le développement parallèle, de soumettre des pull requests pour proposer des modifications, et d'avoir un suivi des problèmes et bugs. GitHub intègre également des fonctionnalités de CI/CD (intégration et livraison continues), d'hébergement de sites web (via GitHub Pages) et une gestion de version (via Git).

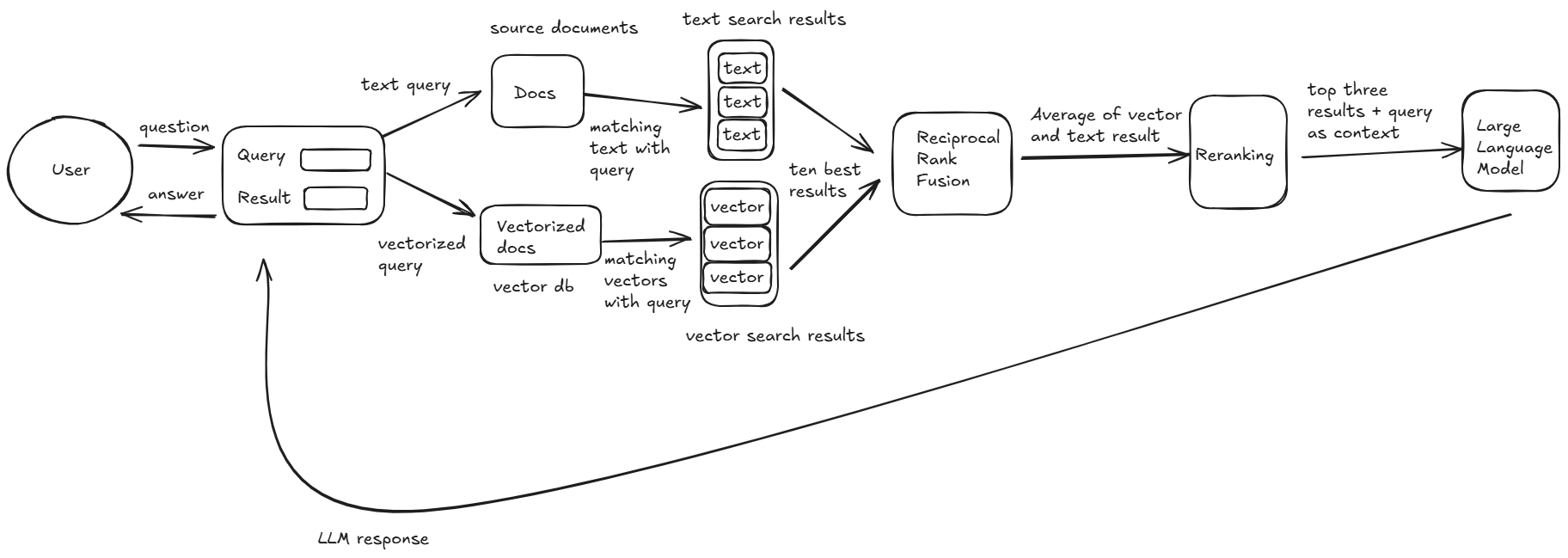
**3. StackOverflow**

StackOverflow est un site de questions et réponses destiné aux développeurs. C’est l'une des plus grandes communautés en ligne pour poser des questions techniques liées à la programmation et au développement de logiciels. Les utilisateurs peuvent poser des questions, fournir des réponses, et voter pour les meilleures réponses. Le site couvre une large gamme de sujets allant des langages de programmation, frameworks et bibliothèques aux meilleures pratiques en développement logiciel, outils de débogage et concepts informatiques.

**4. Medium**

Medium est une plateforme de publication en ligne qui permet à ses utilisateurs de publier des articles, des histoires, des essais et des idées. Medium est largement utilisé par les professionnels et experts pour partager des réflexions sur des sujets divers, y compris la technologie, l'entrepreneuriat, l'innovation, la culture, etc.

Voici le schéma de mon RAG :



*Figure 1. Schéma du RAG*

L’objectif est de réaliser un RAG (Retrieval Augmented Generation), à savoir des recherches sur des documents privés (retrieval) et une réponse synthétisée à partir des meilleurs résultats (génération augmentée à l’aide d’un prompt).

Pour ce faire, il faut un moteur de recherche pour parcourir la base de données et un LLM (Large Language Model) pour synthétiser les réponses. Il faut également choisir un moteur d’inférence pour utiliser le LLM en local.

Les trois principaux moteurs de recherche sont Elasticsearch, SolR et Faiss (Facebook AI Similarity Search).

## C7. Identification et Benchmark de Services d'IA

### Choix moteurs de recherche

#### Elasticsearch

Elasticsearch est un moteur de recherche distribué et open-source basé sur Apache Lucene. Il permet d'indexer et de rechercher des documents à grande échelle avec des fonctionnalités avancées de recherche en texte intégral.

**Avantages :**

* **Flexibilité et Scalabilité** : Elasticsearch est conçu pour gérer de grandes quantités de données et peut évoluer horizontalement (articles, des images, des vidéos…), ce qui en fait un choix idéal pour des systèmes à grande échelle.
* **Indexation et Recherche rapide** : Grâce à son architecture basée sur **Apache Lucene** (bibliothèque open-source de recherche en texte intégral écrite en Java. Elle fournit des fonctionnalités puissantes et flexibles pour l'indexation et la recherche dans de grands ensembles de données non structurées.), Elasticsearch est extrêmement rapide pour la recherche en texte intégral et peut gérer des requêtes complexes efficacement.
* **Facilité d'Intégration** : Il dispose de nombreuses bibliothèques et plugins qui facilitent son intégration avec des frameworks comme Django, Flask, et d'autres outils populaires dans l'écosystème Python.
* **Recherches Complexes** : Supporte des requêtes complexes, comme des agrégations et des filtres, ce qui permet de personnaliser la recherche pour des cas d'utilisation avancés dans un RAG.
* **Support des Recherches Full-text et Vectorielles** : Elasticsearch a intégré le support des recherches par similarité vectorielle (recherche dense), ce qui le rend compatible avec les systèmes RAG qui utilisent des embeddings comme BERT ou T5.

**Inconvénients :**

* **Complexité de Configuration et Maintenance** : Bien que son installation soit relativement simple, sa configuration et sa maintenance peuvent devenir complexes, notamment lors de l’indexation de très grandes bases de données ou d’optimisation des performances.
* **Gestion des Embeddings** : Bien que le support de la recherche vectorielle ait été amélioré, il reste plus complexe à configurer que dans d'autres moteurs spécialisés comme FAISS.
* **Utilisation des Ressources** : Elasticsearch peut être assez gourmand en ressources (mémoire et processeur) lorsqu’il est utilisé à grande échelle avec des données complexes.

#### SolR

Solr est un autre moteur de recherche open-source basé sur Apache Lucene. Il offre des fonctionnalités puissantes pour la recherche en texte intégral et est particulièrement adapté aux environnements d'entreprise.

**Avantages :**

* **Puissant et Flexible** : Solr est un moteur de recherche robuste et flexible qui offre de nombreuses fonctionnalités avancées, comme le faceting (Subdivise les résultats de recherche en facettes, qui sont des catégories ou des critères spécifiques permettant de filtrer ou d’affiner les résultats. Ex : couleur, prix, la marque, catégorie...), la recherche géospatiale, la gestion des données structurées et non structurées, etc.
* **Évolutivité** : Comme Elasticsearch, Solr est conçu pour gérer de grandes quantités de données et peut être déployé sur plusieurs nœuds pour offrir une scalabilité horizontale.
* **Recherches Complexes** : Solr offre un très bon support pour les recherches complexes et permet des personnalisations détaillées des requêtes.
* **Moteur Mature** : Solr est un projet plus ancien que Elasticsearch, donc il est très mature, stable et bien documenté. Il bénéficie également d'une communauté active.

**Inconvénients :**

* **Configuration Complexe** : Solr nécessite une configuration plus approfondie et peut être plus difficile à paramétrer que Elasticsearch. Par exemple, il peut y avoir des défis pour intégrer des embeddings de modèles pré-entraînés pour la recherche par similarité sémantique.
* **Moins Optimisé pour les Embeddings** : Solr est traditionnellement un moteur de recherche basé sur des termes, et bien que des solutions aient été mises en place pour l'intégration des vecteurs, elles ne sont pas aussi simples et intuitives que dans FAISS ou même Elasticsearch.
* **Moins d'Intégration avec Python** : L'intégration avec l'écosystème Python n'est pas aussi fluide qu'avec Elasticsearch, bien qu'il existe des bibliothèques comme pysolr.

#### FAISS

FAISS est une bibliothèque développée par Facebook pour la recherche de similarité haute performance. Elle est optimisée pour les recherches dans des espaces vectoriels à grande échelle.

**Avantages :**

* **Spécialisation pour la Recherche de Similarité de Vecteurs** : FAISS est optimisé pour effectuer des recherches dans de grands espaces de vecteurs (embeddings), ce qui est essentiel dans les systèmes RAG où la recherche de passages ou de documents pertinents repose sur des similarités vectorielles.
* **Haute Performance** : FAISS est très rapide et conçu pour gérer de grands volumes de données, avec des index hautement optimisés pour la recherche en temps réel.
* **Prise en charge de la recherche dense** : FAISS est particulièrement bien adapté à la recherche dense basée sur des embeddings, et il propose de nombreuses méthodes pour améliorer la précision et la rapidité des recherches.
* **Support des GPU** : FAISS permet l'utilisation de GPUs pour accélérer la recherche et l'indexation, ce qui est particulièrement utile pour les systèmes RAG traitant de grandes quantités de données.

**Inconvénients :**

* **Moins de Fonctionnalités de Recherche Traditionnelle** : FAISS se concentre exclusivement sur la recherche de similarité vectorielle, donc il ne prend pas en charge la recherche classique (basée sur des termes) comme Elasticsearch et Solr. Pour des systèmes nécessitant des recherches complexes avec des filtres ou des agrégations, il peut être limité.
* **Gestion Complexe des Embeddings** : FAISS nécessite que les utilisateurs gèrent eux-mêmes l'indexation et l'organisation des embeddings. Bien que très performant, il peut exiger un certain niveau d'expertise pour l'implémentation et l'optimisation.
* **Pas de Fonctionnalités de Recherche Textuelle Complètes** : Par rapport à Elasticsearch et Solr, FAISS n'est pas conçu pour effectuer des recherches en texte intégral, ce qui peut limiter sa polyvalence dans des systèmes RAG où les documents sont souvent de grande taille et ont des métadonnées complexes.

En résumé, Elasticsearch gère à la fois les recherches vectorielles (complexe avec SolR) et les recherche textuel (pas conçu pour ça) de plus, il s'intègre bien à python et Django, ses inconvénients sont la maintenance et les ressources ce qui pour le projet qui est limité ne pose donc pas de problème. C’est également ce moteur de recherche que l’on utilise en entreprise.

### B. Utilisation d’elasticsearch

Pour faire des recherches avec elasticsearch, j’utilise un mode de recherche hybride (vectorielle + textuelle) car cela s’avère être la solution la plus efficace.

#### Indexation des documents

Lors de la recherche, on classe les documents par index pour les retrouver plus facilement et ne pas chercher dans toute une base de données s’il y a un nombre de documents tel que les temps de recherche pourrait devenir trop important. Dans notre cas, le nombre de documents était faible, un index suffit amplement.

On définit d’une part notre index et d’autre part ce qu’est un document à savoir les informations que le moteur de recherche va pouvoir utiliser pour faire la recherche et pour récupérer les informations sur le document.

Ici, on va effectuer des recherches sur le titre, le résumé et la vectorisation de la concaténation du titre et du résumé. On définit nos champs titre et résumé comme étant ceux du modèle Article et le vecteur comme étant un vecteur dense, c’est-à-dire un vecteur contenant les valeurs significatives du résumé et du titre vectorisés afin de tenir compte du sens du texte.

Pour réaliser la vectorisation, on utilise un modèle capable de donner les résultats les plus significatifs. Le modèle le plus adapté est BERT.

##### BERT (Bidirectional Encoder Representations from Transformers)

BERT, développé par Google AI en 2018, est un modèle de traitement du langage naturel (NLP) basé sur l'architecture Transformer (les Transformers capturent les relations entre tous les mots d'une séquence, indépendamment de leur position.). Contrairement aux modèles unidirectionnels (comme GPT, avant GPT-3), BERT analyse un mot en tenant compte à la fois des mots qui le précèdent et qui le suivent.

**Avantages :**

* **Compréhension contextuelle :** Entraînement bidirectionnel pour une meilleure précision.
* **Performance sur les benchmarks :** BERT a battu des records sur plusieurs benchmarks NLP comme GLUE et SQuAD. (Source : https://arxiv.org/pdf/1810.04805)
* **Polyvalence :** Utilisé dans de nombreuses tâches grâce à son pré-entraînement généraliste.

**Limites :**

* **Taille et complexité :** Les modèles pré-entraînés de BERT sont volumineux, ce qui entraîne des coûts élevés en mémoire et en calcul.
* **Coût énergétique :** Le pré-entraînement et l’utilisation de BERT nécessitent une puissance de calcul significative.
* **Pas adapté aux séquences longues :** Limité à une longueur de séquence de 512 tokens, ce qui peut poser problème pour les documents très longs.

Depuis son lancement BERT a inspiré de nombreuses variantes :

* **RoBERTa :** Optimisation des hyperparamètres pour des performances améliorées.
* **DistilBERT :** Une version plus légère et rapide.
* **BioBERT :** Adapté aux données biomédicales.
* **ALBERT :** Une version compacte de BERT pour réduire les coûts de calcul.

J’utilise une variante entraînée sur les résumés médicaux de PubMed : « microsoft/BiomedNLP-BiomedBERT-base-uncased-abstract »

**Avantages :**

* **Adapté au vocabulaire biomédical :** Le modèle comprend des termes médicaux, pharmacologiques et biologiques mieux qu'un modèle BERT général. Exemple : Il comprend des termes complexes comme *"angiogenèse"*, *"cytokines"*, ou *"miARN"*.
* **Précision dans les tâches biomédicales :** Plus performant que des modèles généralistes comme bert-base-uncased sur des corpus biomédicaux.
* **Flexible :** Peut être ajusté (fine-tuned) pour diverses tâches spécifiques à la recherche biomédicale.

**Limites :**

* **Domaine limité :** Bien qu'excellent pour les textes biomédicaux, il est moins performant sur des textes généraux ou non scientifiques.
* **Données spécifiques nécessaires :** Pour des tâches très spécialisées, un **fine-tuning** supplémentaire sur un sous-ensemble de données biomédicales pertinentes peut être nécessaire.
* **Consommation de ressources :** Comme tout modèle basé sur BERT, il peut nécessiter des ressources significatives pour l'entraînement ou l'inférence.

Ce qui en fait un modèle parfaitement adapté à mon cas car il comprend très bien les terminologies, relations et concepts spécifiques au domaine de la santé et peut donc aider à répondre à des questions techniques dans le domaine de la santé.

##### Création des documents à indexer

La définition du modèle de document à indexer se fait dans *document.py*.

On crée une instance d’index Elasticsearch basé sur le nom d’index et on réalise le mapping. Ici le projet ayant peu de données, on n’a pas besoin d’exécuter des tâches en parallèles (Shards) et l’accessibilité (replicas) qui correspond à une instance d’elasticsearch, comme il y en a qu’une on peut mettre les répliques à 0. La méthode utilisée pour faire la recherche est le KNN.

Le KNN (k-Nearest Neighbors) est un algorithme qui trouve les **k** éléments les plus proches d'un point donné dans un espace vectoriel, basé sur une métrique de distance comme :

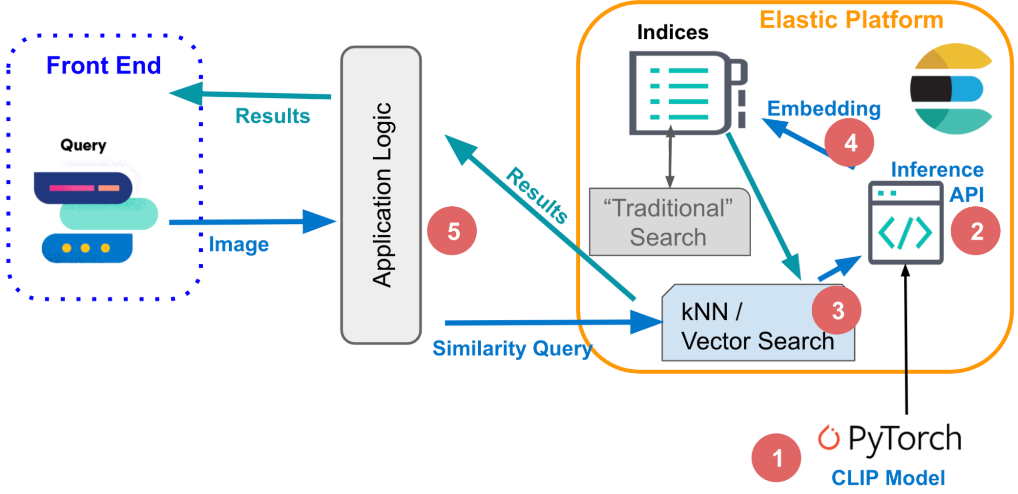
* La distance euclidienne.
* La distance cosinus.
* D'autres distances personnalisées.

On définit ensuite le document :

* + Title
  + Abstract
  + Title\_abstract\_vector

Le document est ensuite rempli dans *commands.py* avec la table Article qui a une méthode pour réaliser les embeddings et l’indexation est réalisée. Le moteur de recherche peut maintenant être utilisé.

##### Recherche des résultats avec une question en langage naturel.



*Figure 2. Schéma fonctionnement elasticsearch : https://www.elastic.co/fr/blog/5-technical-components-image-similarity-search*

Il s’agit de la première partie du RAG. La question est posée dans un formulaire, l’appel à la fonction se trouve donc dans *views.py* et la fonction dans *business\_logic.py.*

A partir des documents importés, on va effectuer une recherche.

Dans un premier temps, on vectorise la question avec le même modèle que pour les articles.

On définit le type de recherche que l’on souhaite, full-text, sémantique, hybride. Le mode hybride étant payant, j’en ai fait un à la main avec une première recherche full text, puis sémantique et une fonction pour assembler les deux.

D’abord, Elasticsearch va récupérer les 10 résultats les plus pertinents en faisant une recherche KNN parmi 5000 fichiers. Ensuite, il va faire une recherche « multi\_match », c’est-à-dire qu’il va rechercher la correspondance dans les champs titre et abstract. On utilise le type « best fields » pour qu’il garde le score plus élevé parmi les champs titre et abstract.

Ceux-ci sont affectés d’un poids 2 pour le titre et 5 pour le résumé ce qui indique le poids dans le calcul du score en cas de match. On récupère les 10 résultats les plus pertinents. On injecte ces résultats dans une fonction qui va itérer sur chacun de ces résultats pour leur affecter un score en fonction de leur rang, plus le rang est bas plus le score sera élevé. Ensuite, on additionne les scores de même rang. On les classe dans l’ordre décroissant. Puis on utilise un Cross-Encoder, pour réordonner les résultats. Le Cross-Encoder va capturer toutes les interactions entre les deux textes (ici, la question et le document) pour en évaluer la pertinence.

J’utilise le modèle : ‘cross-encoder/ms-marco-MiniLM-L-6-v2', développé pour traiter des paires (requête, documents) afin de classer les documents par rapport à une requête. On récupère le document et le score correspondant.

Lorsque l’on a nos résultats, on utilise les ids pour récupérer les articles correspondant dans la base de données et les enregistrer dans un dictionnaire.

### C. Génération de la réponse

#### Choix du LLM

##### GPT (Generative Pre-trained Transformer)

GPT est une série de modèles créée par Open AI. Ces modèles sont des réseaux neuronaux avancés conçus pour comprendre et générer du texte naturel.

Ils sont très performants pour la compréhension du langage et la génération de texte mais pas open-source.

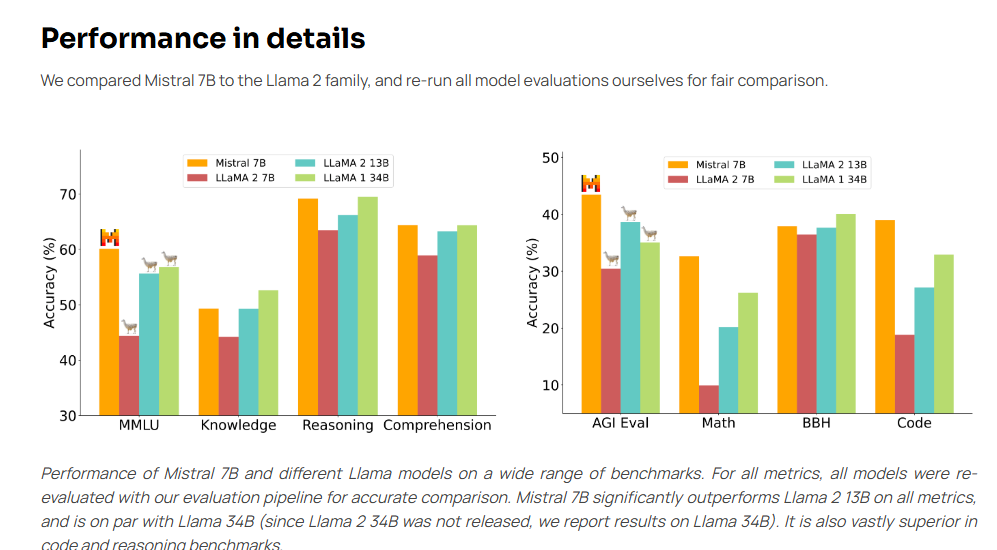
##### Mistral

Mistral est une entreprise de pointe spécialisée dans les modèles de langage de grande taille (LLM). Fondée en 2023 par des experts en intelligence artificielle, elle vise à offrir des solutions performantes, ouvertes et transparentes pour les applications basées sur le traitement du langage naturel (NLP).

Mistral propose des modèles open-source équivalents à GPT.

##### Llama

LLaMA est une famille de modèles de langage développée par Meta AI (anciennement Facebook AI) datant de 2023.



*Figure 3. Performance de mistral. (source :* [*https://mistral.ai/en/news/announcing-mistral-7b*](https://mistral.ai/en/news/announcing-mistral-7b)*)*

Afin de générer une réponse, j’utilise un LLM ici Mistral modèle 7B.

Le modèle Mistral 7b est basé sur la famille des Transformers comme GPT. Il a 7 milliards de paramètres d’où son nom. Ci-dessus une évaluation de ses performances issues du site mistral.

Il est plus performant que d’autres modèles open-source tels que LLama, ce n’est pas le plus performant mais il offre un bon compromis en rapport de performances/consommation de mémoire.

Afin de le faire tourner en local, j’utilise le moteur d’inférence Ollama. Un moteur d'inférence est un composant logiciel ou système spécialisé conçu pour exécuter des modèles d'intelligence artificielle (IA) ou d'apprentissage automatique (ML) afin de produire des résultats ou des prédictions à partir de données d'entrée.

Compétence validée : Identifier des services d’intelligence artificielle préexistants à partir de l’expression de besoin en fonctionnalités d’intelligence artificielle, en réalisant un benchmark de services existants et en analysant leurs caractéristiques pour formaliser une ou plusieurs recommandations de services adaptés au besoin.

#### Choix du moteur d’inférence

Il existe deux principaux moteurs d’inférence Ollama et VLLM.

##### Ollama

Ollama est conçu pour être un framework d'optimisation de modèles de langage qui permet de déployer des modèles LLM sur des serveurs locaux. Il s’adresse aux entreprises cherchant à intégrer des LLM sur leurs serveurs locaux tout en garantissant la sécurité et la confidentialité des données.

L'un des principaux avantages d'Ollama est qu'il permet un contrôle total des données, car les utilisateurs peuvent exécuter des modèles sur leurs propres serveurs locaux. Cela garantit que les données sensibles ne quittent jamais l'infrastructure interne.

##### VLLM

VLLM (ou Very Large Language Models) est une bibliothèque open-source optimisée pour l'exécution et l'inférence de modèles de langage de grande taille (LLMs) tels que GPT-3 et GPT-4.

VLLM est une bibliothèque puissante et flexible pour l'exécution de modèles de langage de grande taille dans des environnements cloud ou de centres de données.

La gestion de la mémoire est optimisée pour traiter de très grands modèles avec des ressources limitées, grâce à des techniques comme la quantification des poids des modèles et l'utilisation de memory-mapped files.

Etant donné que le projet se fait en local et que la taille des données est modérée, Ollama est donc plus adapté au projet.

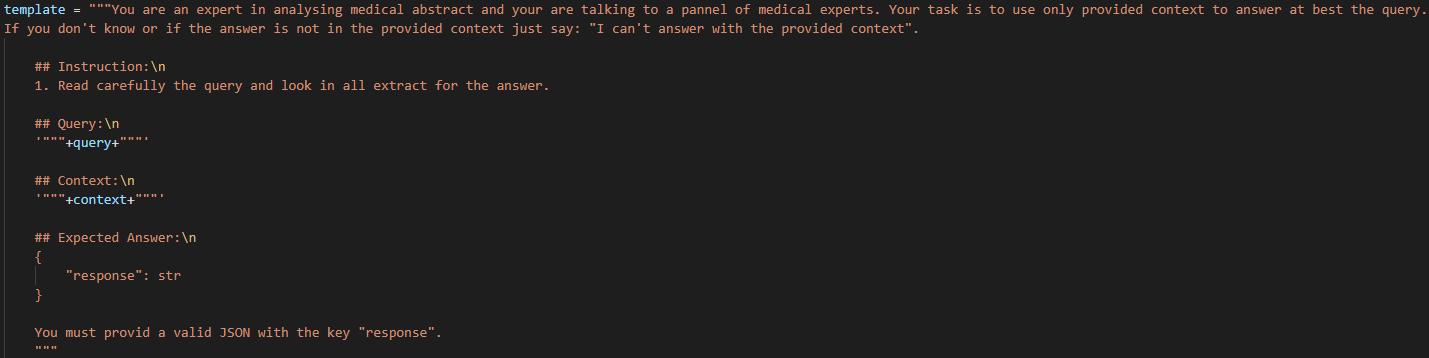
## C8. Paramétrage et Intégration d'un Service d'IA

### Utilisation du LLM

Maintenant que nous avons déterminé le LLM et son moteur d’inférence, nous allons l’augmenter à l’aide d’un prompt. Ce prompt contient les éléments suivants :

* Un persona : représente une configuration ou une personnalité simulée que le modèle incarne pour répondre aux besoins d'un utilisateur spécifique ou d'une application particulière. Cela peut inclure son style de communication, ses connaissances, ou sa perspective.
* Des instructions : Les différentes tâche que le modèle devra réaliser pour arriver aux résultats, et la gestion des cas particuliers (ex : si la réponse ne se trouve pas dans les documents.
* La query : La question posée.
* Le context : Les articles retournés par elasticsearch servant à répondre à la question.
* Le format de la réponse : un dictionnaire avec une clé « réponse » et comme valeur la réponse du LLM sous forme de chaîne de caractère.

Voici le prompt :



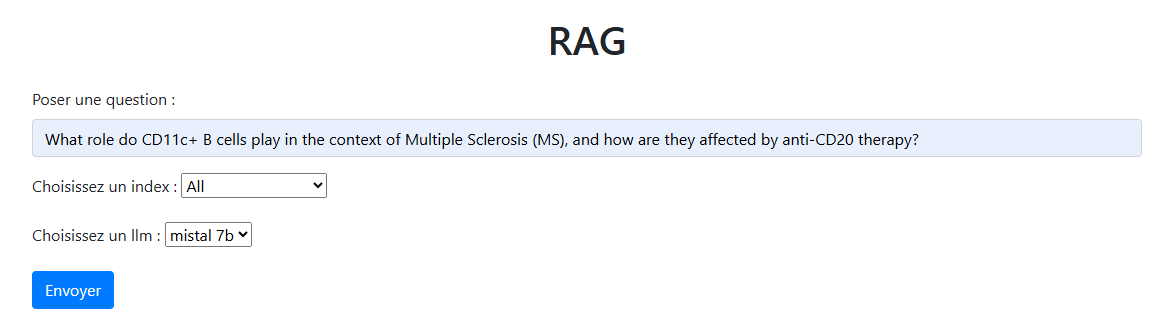
*Figure 4. Prompt de génération*

Ensuite, on paramètre le LLM avec les éléments suivants :

* Le modèle : ici mistral pour Mistral 7b
* Le message : Un dictionnaire où l’on passe le rôle à savoir la personne qui envoie le message (assistant si c’est le LLM, system pour définir les comportements généraux du LLM, user si c’est un message envoyé par un utilisateur), ici comme on lui passe un Template avec le message de l’utilisateur, on choisit « user ».
* Stream : si on veut que le message apparaisse au fur et à mesure.
* Le format : pour définir le format de la réponse ici, un JSON.
* Les options : on choisit une valeur fixe de seed pour que le résultat soit toujours les même d’un utilisateur à l’autre et une température qui mesure le niveau de créativité entre 0 et 1. (0 pas de créativité et 1 très imaginatif). Comme nous interrogeons des document la température est à 0.

On va réaliser la requête sur le conteneur ollama avec le port 11434 en lui passant les paramètres ci-dessus.

Le LLM est appelé dans la vue rag\_articles via la fonction handle\_rag\_pipeline. Lors de la soumission de la requête par l’utilisateur, le moteur de recherche va effectuer une recherche sur les documents et le LLM va synthétiser le résultat en langage naturel.

* Figure 5. Ecran du RAG*

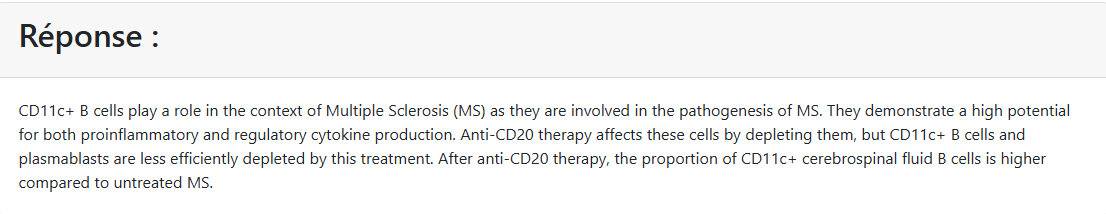


Figure 6. Réponse générée

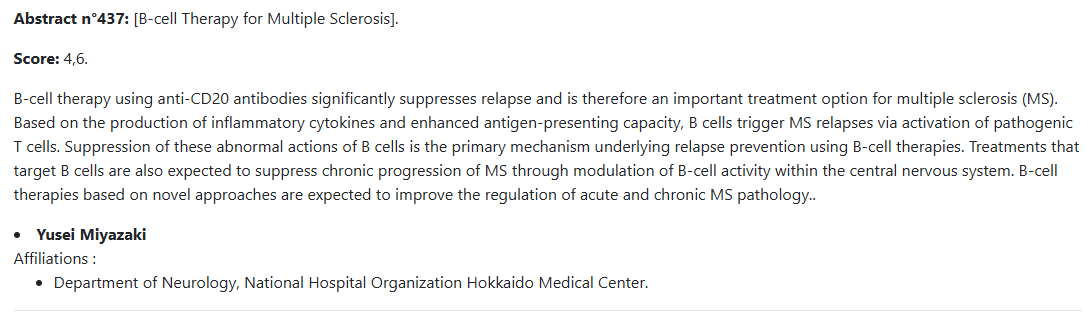


Figure 7. Document utilisé pour la réponse

Compétence validée : Paramétrer un service d’intelligence artificielle en suivant sa documentation technique et en respectant les spécifications du projet, afin de permettre l’intégration des connecteurs du service dans le système d’information.

Développer une API exposant un modèle d’intelligence artificielle en utilisant l’architecture REST pour permettre l’interaction entre le modèle et les autres composants du projet.

Intégrer l’API d’un modèle ou d’un service d’intelligence artificielle dans une application, en respectant les spécifications du projet et les normes d’accessibilité en vigueur, à l’aide de la documentation technique de l’API, afin de créer les fonctionnalités d’intelligence artificielle de l’application.

# Conclusion

Nous avons maintenant identifié tous les éléments pour effectuer le RAG :

* Le moteur de recherche
* Le LLM
* Les paramétrages :
  + Indexation
  + Vectorisation
  + Mode de recherche
  + Nombre d’entrées
  + Nombre de sorties
  + Trie des résultats
  + Prompt
  + Température
  + …

Nous allons pouvoir intégrer notre RAG au projet.