Réalisation d’une application de type RAG

**      **

Certification : Développeur d’application en Intelligence Artificiel

Marseille – Nice

Bloc de compétences 2 - E3

Intégrer des modèles et des services d’intelligence artificielle

LOMBARDI Joachim

Table des matières

[A4. Compétence : Intégration d'un modèle ou service d'IA 2](#_Toc192421974)

[C9. Développement d'une API REST pour un Modèle d'IA 2](#_Toc192421975)

[1) Les vues basées sur des fonctions (Function-Based Views - FBV) 2](#_Toc192421976)

[2) Les vues basées sur des classes (Class-Based Views - CBV) 3](#_Toc192421977)

[C10. Intégration de l'API dans une Application 4](#_Toc192421978)

[A5. Compétence : Facilitation du déploiement avec une approche MLOps 5](#_Toc192421979)

[C11. Monitorage d'un Modèle d'IA 5](#_Toc192421980)

[I. Monitoring 5](#_Toc192421981)

[II. Logs 7](#_Toc192421982)

[C12. Tests Automatisés d'un Modèle d'IA 8](#_Toc192421983)

[Choix du mode d’évaluation 8](#_Toc192421984)

[C13. Création d'une Chaîne de Livraison Continue 12](#_Toc192421985)

[Outils de versionnement et CI/CD 12](#_Toc192421986)

[Conclusion 15](#_Toc192421987)

# A4. Compétence : Intégration d'un modèle ou service d'IA

## C9. Développement d'une API REST pour un Modèle d'IA

Il existe deux types de script pour les vues dans Django :

### Les vues basées sur des fonctions (Function-Based Views - FBV)

Les vues basées sur des fonctions sont des fonctions Python ordinaires qui prennent une requête (Http Request) en paramètre.

Avantages :

* Faciles à lire et à comprendre.
* Idéal pour des besoins simples sans comportement complexe.
* Pas de surcharge avec des concepts avancés.

Inconvénients :

* Deviennent difficiles à maintenir lorsqu'elles deviennent complexes.
* Manquent de modularité et de réutilisabilité pour des comportements similaires.

### Les vues basées sur des classes (Class-Based Views - CBV)

Les vues basées sur des classes sont des classes Python qui héritent des classes utilitaires fournies par Django pour gérer différentes tâches courantes (comme les requêtes HTTP GET ou POST).

Avantages :

* Permet une meilleure organisation pour les projets complexes.
* Plus facile d'ajouter ou de modifier des fonctionnalités spécifiques grâce à l’héritage.
* Supportent des mixins pour réutiliser le code (exemple : LoginRequiredMixin).

Inconvénients :

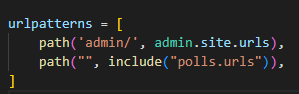
* Plus complexes à comprendre et à utiliser.
* Nécessitent une bonne connaissance des concepts orientés objet.

Ainsi chaque fonction présentes dans *views.py* est associée à une URL spécifique présente dans le fichier *urls.py* dans le dossier de l’application où sont stockées toutes les urls des différentes pages du projet.



*Figure 1. polls/urls.py*

Pour trouver l’url général de l’api, il faut aller dans le fichier *urls.py* où se trouve le fichier *settings.py.*



*Figure 2. pubmed\_analyze/urls.py*

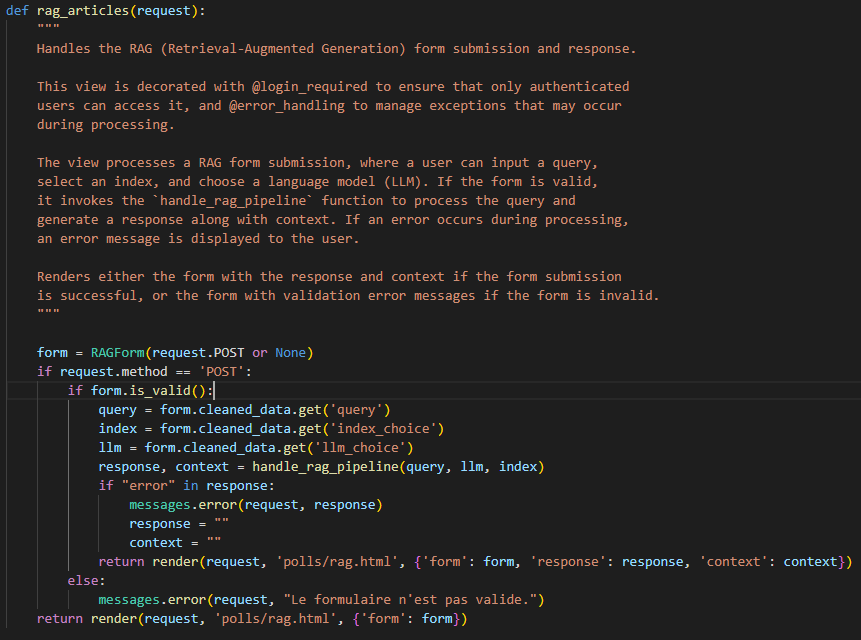
Les urls des différentes pages se concatènent avec l’url générale.

A chaque fonction présente dans *views.py* est associé un fichier HTML où est défini le design de la page. Les formulaires du CRUD après validation renvoient tous vers la page d'affichage des articles de la base de données.

Les formulaires sont créés dans *forms.py* en récupérant les modèles et en utilisant leurs attributs, on définit aussi dans forms des méthodes de sauvegarde. Ils sont ensuite importés dans les HTML correspondant.

## C10. Intégration de l'API dans une Application

L’API est intégrée dans l’application via la vue « rag\_articles »



*Figure 3. Vue gérant le RAG*

La vue fait appel à la fonction de monitoring « handle\_rag\_pipeline » qui elle-même fait appel au moteur de recherche et au LLM. Le résultat est envoyé vers la page HTML.

# A5. Compétence : Facilitation du déploiement avec une approche MLOps

## C11. Monitorage d'un Modèle d'IA

### Monitoring

Afin de réaliser le monitoring de l'application, j'utilise la plateforme Grafana.

#### Grafana

Grafana est une plateforme open-source de surveillance et de visualisation de données en temps réel qui permet de collecter, analyser et présenter des métriques issues de diverses sources dans des tableaux de bord interactifs. Il offre une interface de visualisation (graphiques, tableaux, diagrammes) et s’intègre avec des outils comme : Prometheus (surveillance des systèmes), Elasticsearch, InfluxDB (base de données pour séries temporelles), MySQL/PostgreSQL, Cloud (AWS, Google Cloud, Azure). Il permet également de configurer des alertes en fonction de seuils et de les notifier via emails ou messageries.

Avantages

* Interface intuitive : Facile à prendre en main, même pour des utilisateurs non techniques.

Tableaux de bord attractifs et interactifs.

* Open-source et extensible : gratuit dans sa version de base. Communauté active qui développe des plugins et fournit du support.
* Alertes avancées : Détection proactive des problèmes avant qu'ils ne deviennent critiques.
* Flexibilité des sources de données : Prend en charge un large éventail de sources, ce qui en fait une solution polyvalente.

Inconvénients

* Courbe d'apprentissage : Nécessite un peu de temps pour maîtriser les fonctionnalités avancées (alertes, intégrations complexes).
* Dépendance aux sources de données : Ne collecte pas de données directement ; il s'appuie sur des outils comme Prometheus, Elasticsearch, ou d'autres systèmes.
* Gestion des performances : Avec des tableaux de bord très complexes et des sources de données volumineuses, les performances peuvent en souffrir.

Afin de collecter des métriques, j’utilise la plateforme Prometheus.

#### Prometheus

Prometheus est une plateforme de monitoring open-source conçue pour collecter, stocker, et analyser des métriques en temps réel.

Chaque métrique est identifiée par un nom et des étiquettes.

Le langage de requête est PromQL, c’est lui qui permet d’extraire et d’analyser les métriques.

Prometheus stocke lui-même ses métriques. Il possède un système d’alerte basé sur les métriques collectées.

Avantages

* Optimisé pour les métriques : Haute performance pour des milliers de points de données par seconde.
* Écosystème riche : Large communauté et support pour Kubernetes et d'autres environnements cloud.
* Alerting natif : Alertmanager permet des alertes complexes sans outils tiers.

Limites

* Pas adapté aux logs ou données textuelles : Prometheus gère uniquement des métriques numériques.
* Rétention des données limitée : Pas conçu pour conserver les métriques sur de longues périodes.
* Scraping pull-based uniquement : Nécessite que les services exposent des endpoints.

Pour surveiller la performance et l’utilisation de mon application RAG, j’ai défini les métriques suivantes :

* *rag\_pipeline\_latency :* Il est appliqué en tant que décorateur de ma fonction RAG et mesure le temps total d’exécution de mon pipeline RAG.
* *search\_latency :* Mesure le temps total d’exécution de la recherche Elasticsearch.
* *llm\_latency :* Mesure le temps total d'exécution du LLM.
* *rag\_requests\_total :* Compte le nombre total de requêtes reçues par le pipeline.
* *rag\_errors\_total :* Compte le nombre total d'erreurs rencontrées dans le pipeline.

Les résultats sont visibles dans Grafana sous forme de graphiques en ajoutant les métriques.

J’utilise uptime kuma pour m’assurer que le conteneur fonctionne bien.

#### Uptime Kuma

Uptime Kuma est un outil open-source de surveillance de services.

Voici ses principales caractéristiques :

* Surveillance variée :
  + HTTP(s), TCP, UDP, Ping, DNS, Push, Docker et plus.
* Interface utilisateur moderne : Interface claire et personnalisable.
* Notifications intégrées : Support pour Telegram, Discord, Slack, email, etc.
* Auto-hébergement : Facile à déployer via Docker ou manuellement.
* Tableau de bord complet : Pour voir les temps de réponse et les incidents.

J’ai configuré Uptime Kuma pour qu’il surveille l’état du conteneur django et envoie un mail sur mon adresse s’il y a déconnection du conteneur.

Le monitoring est réalisé lors de l’utilisation du RAG et peut être visualisé via les onglets UpTime Kuma et Grafana.

Le détail des métriques et leur résultats sont indiqués dans le rapport E5.

Compétence visée :

* Monitorer un modèle d’intelligence artificielle à partir des métriques courantes et spécifiques au projet, en intégrant les outils de collecte, d’alerte et de restitution des données du monitorage pour permettre l’amélioration du modèle de façon itérative.

* Assurer le maintien en condition opérationnelle d’une application d’intelligence artificielle.

### Logs

Afin de détecter les éventuelles erreurs lors de l’exécution de l’application, j’ai implanté un mécanisme de gestion de ces dernières qui fonctionne de la manière suivante pour l’exécution de chaque fonction :

* Capturer l’erreur : évite de faire planter l’application.
* Journaliser l’erreur : On enregistre le message de l’exception puis la trace complète pour comprendre pourquoi l’erreur s’est produite.
* Envoyer un mail avec les détails de l’erreur : le mail contient le message de l’exception, sa trace.

Afin de réaliser ces étapes je crée un décorateur « @error handling » qui prend la fonction en argument et la retourne si elle fonctionne ou passe l’erreur à la fonction de journalisation si elle échoue.

## C12. Tests Automatisés d'un Modèle d'IA

### Choix du mode d’évaluation

Pour évaluer la qualité d’un RAG, il existe différentes solutions en voici quelques-unes :

* La méthode humaine : Un ou des experts évaluent la qualité de la réponse générée et du des résultats de la recherche.
* LLM as a judge : à l’aide d’un prompt, un LLM évalue la qualité de la réponse et du retrieval (résultats de la recherche). Il évalue la cohérence entre la réponse et la question (pertinence), la réponse par rapport aux documents fournis (fact-checking), les documents fournis par rapport à la question. On peut aussi lui demander de comparer les réponses fournies aux réponses attendues…
* L’évaluation automatique nécessite de connaître la réponse attendue. Elle utilise des métriques quantitatives.

Pour la génération :

* + **ROUGE** : Mesure le chevauchement des n-grams entre la réponse générée et la référence.
  + **BLEU** : Évalue la précision des n-grams entre la sortie et la référence (utilisé surtout pour la traduction).
  + **METEOR** : Prend en compte la synonymie et le stemming pour mesurer la similarité.
  + **BERTScore** : Compare les embeddings des réponses générées et des références pour une évaluation sémantique.

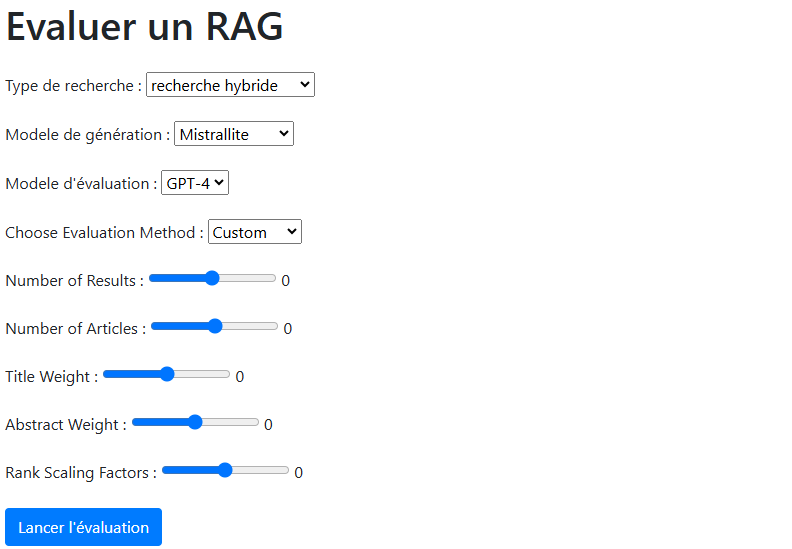
Pour le retrieval :

* + **Recall@k** : Taux de récupération des documents pertinents parmi les kkk premiers résultats.
  + **MRR (Mean Reciprocal Rank)** : Mesure la position du premier document pertinent dans les résultats.
  + **NDCG (Normalized Discounted Cumulative Gain)** : Pondère la pertinence des documents récupérés en fonction de leur position.

Ces métriques peuvent être déterminées à l’aide de bibliothèques (Pyserini pour le retrieval et Hugging Face Transformers pour la génération) ou de FrameWork (Elasticsearch, FAISS…) ou des plateformes (DeepAI, Google EvalAI, OpenAI API, LangChain).

Pour faire l’évaluation j’ai choisi LLM as a judge car je ne connais pas les bons résultats.

L’évaluation du RAG se fait à l’aide d’un LLM différent de celui qui fait la génération ici GPT-4-o car il est plus performant, en effet il a été conçu pour travailler sur des tâches complexes là où mistral 7b est adapté à des environnements limités. GPT-4-o est une variante optimisée de GPT-4, développée par OpenAI. Elle a été conçue pour offrir une performance similaire ou supérieure à GPT-4, tout en étant plus rapide et moins coûteuse en termes de consommation de ressources.



*Figure 5. Ecran d'évaluation du RAG*

#### Paramètres du RAG

Afin de réaliser le RAG le plus performant, je fais varier certains paramètres influençant le retrieval et la génération.

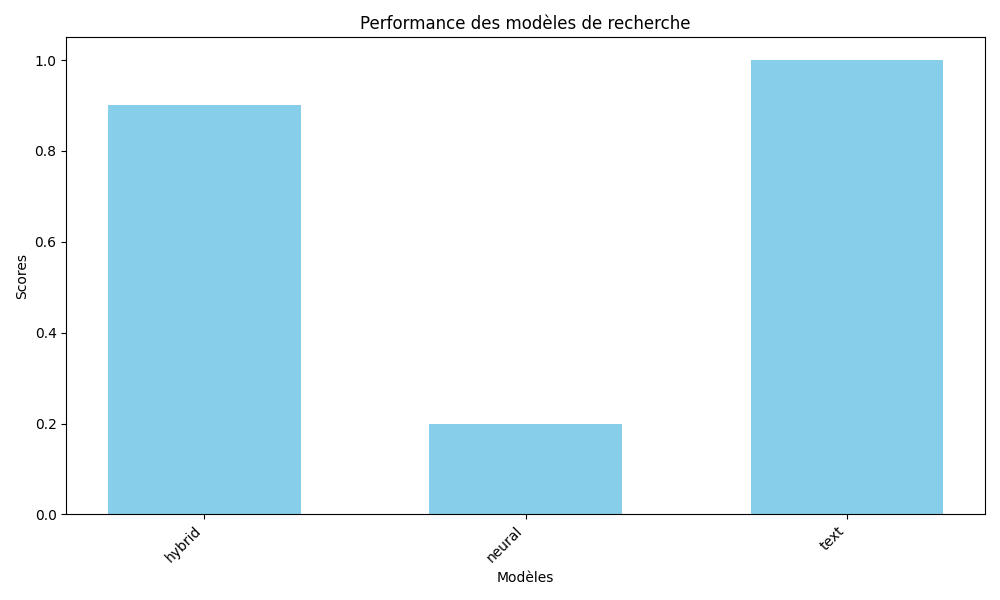
Pour le retrieval :

* Les poids du résumé et du titre
* Le type de recherche : full-text, hybrid, vectorielle.
* Le nombre d’articles que l’on passe pour le re-ranking.
* Le prompt.

Ci-après, le graphique des scores donnés par GPT4-o selon le type de recherche avec le prompt suivant : *« You are an expert in medical abstracts. You will receive two medical abstracts and a query. Your task is to determine which of these abstracts contains the most pertinent information to answer the query.*

*You must return the number of the abstract containing the most relevant information. If the two abstracts contain the same information, return number 1.*

*Please provide a clear reason for your decision.* »



On voit que le type *text* donne les meilleurs résultats et le *neural* les pires. En ce qui me concerne, je suis critique sur ces résultats car la recherche neuronale tient compte du contexte pour classer les résultats ce qui est bien plus efficace qu’une simple recherche par mot clé car les questions nécessitent des réponses détaillées et non un élément de réponse. Pareil pour la recherche hybride elle utilise la combinaison des recherches neuronal et textuel en moyennant les scores donc le score devrait être le plus élevé.

Pour la génération :

* Le prompt.
* Le modèle.

Ci-après, le graphique des scores donnés par GPT4-o selon le modèle Mistral avec le prompt suivant: *« Your task is to score the relevance between a generated answer and the query based on the retrieval in the range between 1 and 5, and please also provide the scoring reason.*

*Your primary focus should be on determining whether the generated answer contains sufficient information to address the given query according to the retrieval.*

*If the generated answer fails to provide enough relevant information or contains excessive extraneous information, then you should reduce the score accordingly.*

*If the generated answer contradicts the retrieval, it will receive a low score of 1-2.*

*For example, for query "Is the sky blue?", the retrieval is "the sky is blue." and the generated answer is "No, the sky is not blue.".*

*In this example, the generated answer contradicts the retrieval by stating that the sky is not blue, when in fact it is blue.*

*This inconsistency would result in a low score of 1-2, and the reason for the low score would reflect the contradiction between the generated answer and the retrieval.*

*Please provide a clear reason for the low score, explaining how the generated answer contradicts the retrieval.*

*Labeling standards are as following:*

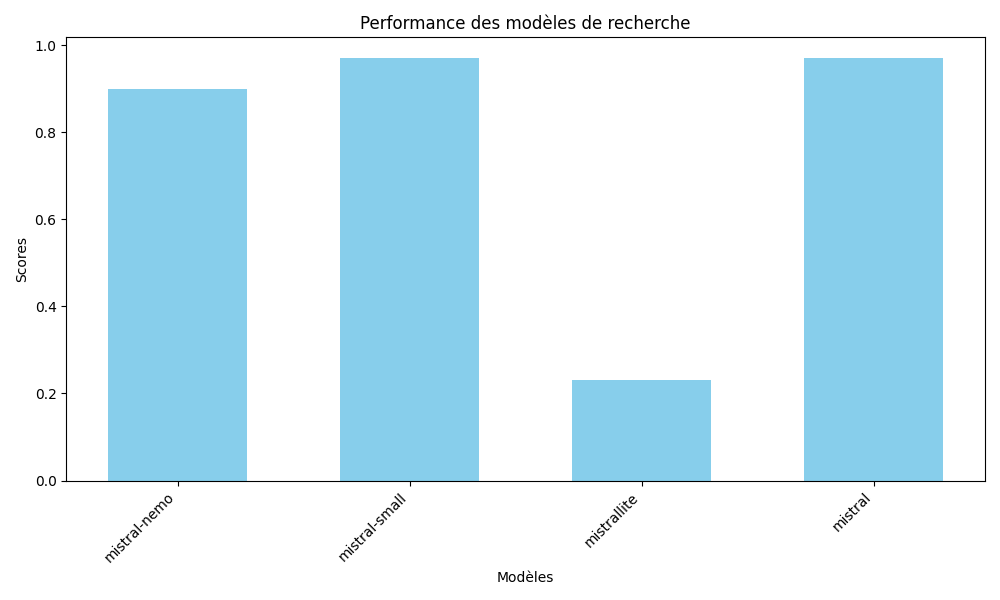
*5 - ideal, should include all information to answer the query comparing to the retrieval，* *and the generated answer is consistent with the retrieval*

*4 - mostly relevant, although it might be a little too narrow or too broad comparing to the retrieval, and the generated answer is consistent with the retrieval*

*3 - somewhat relevant, might be partly helpful but might be hard to read or contain other irrelevant content comparing to the retrieval, and the generated answer is consistent with the retrieval*

*2 - barely relevant, perhaps shown as a last resort comparing to the retrieval, and the generated answer contradicts with the retrieval*

*1 - completely irrelevant, should never be used for answering this query comparing to the retrieval, and the generated answer contradicts with the retrieval*»



Ici, le meilleur modèle est mistral (7 milliards de paramètres) et mistral-small (22 milliards), nemo (12 milliards) et mistralite (7 milliards). Cela correspond à ce à quoi je m’attendais, mais il est certain que la méthode d’évaluation LLM as a judge nécessite un contrôle car elle est bien évidemment sujette à des biais en ce qui concerne le jugement de la réponse.

## C13. Création d'une Chaîne de Livraison Continue

### Outils de versionnement et CI/CD

#### Outils de versionnement

Le versionnement est le processus de gestion des changements dans les fichiers d’un projet, en particulier dans le développement logiciel. Il permet de suivre l'historique des modifications, de collaborer efficacement, et de revenir à des états précédents si nécessaire.

Il existe plusieurs outils de versionnement dont voici les principaux :

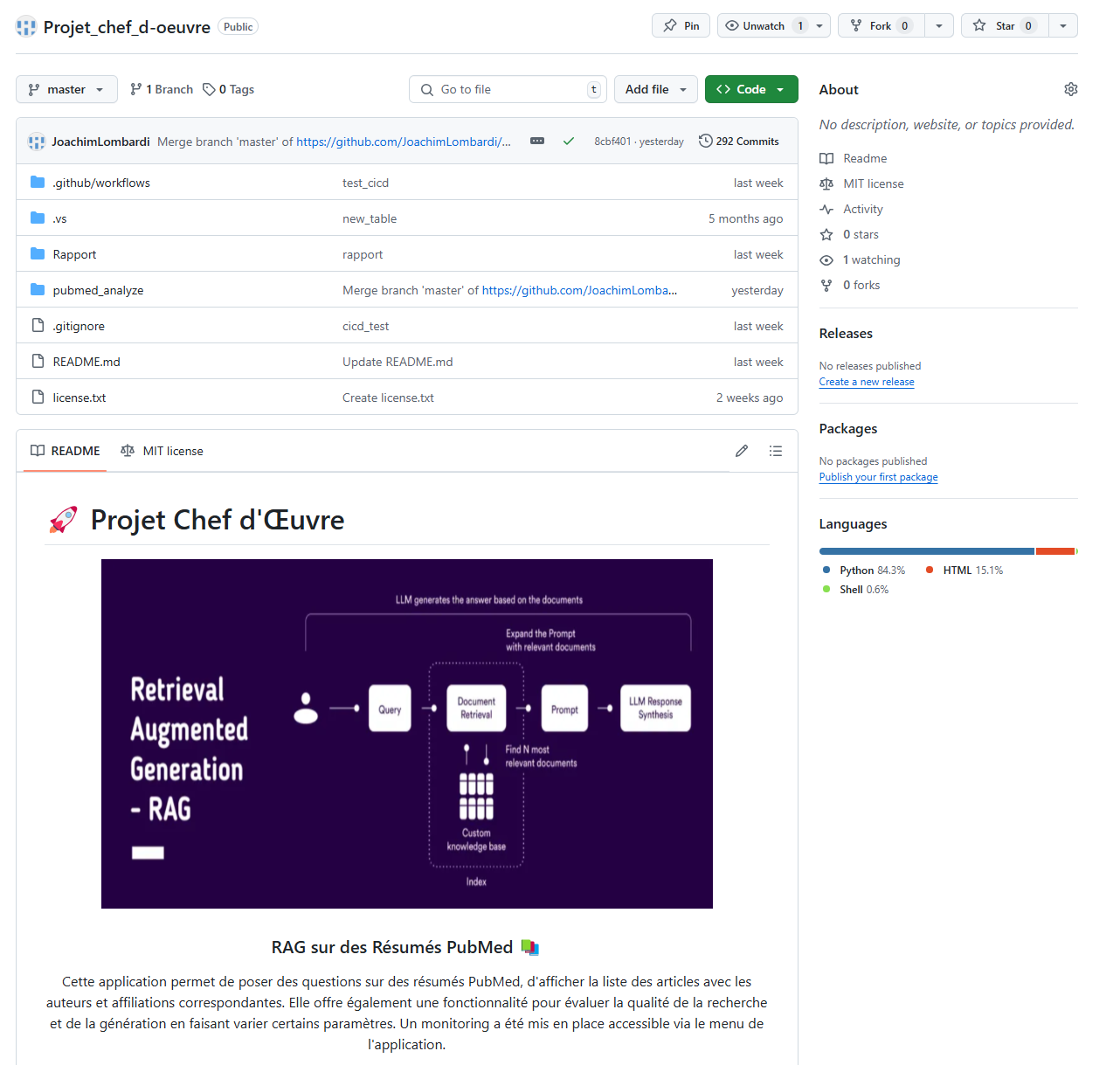
##### Git

Git est un outil de versionnement open source qui permet à plusieurs développeurs de collaborer sur un projet, de créer des branches pour des fonctionnalités spécifiques, et de fusionner les modifications dans la branche principale. (Une branche est une copie indépendante du code source, qui permet de travailler sur des fonctionnalités ou des correctifs sans affecter la version principale du projet.). Il fonctionne localement et permet de synchroniser le code local avec un dépôt distant GitHub.

GitHub est une plateforme d’hébergement de code basée sur Git, permettant aux développeurs de :

* Collaborer sur des projets en équipe.
* Gérer le versionnement avec Git.
* Automatiser des workflows avec GitHub Actions.
* Partager des projets open source ou privés.

Il est utilisé par des millions de développeurs dans le monde et est gratuit.



*Figure 6. Mon projet sur github.*

##### Subversion (SVN)

Subversion est également un système de versionnement mais contrairement à Git, les fichiers sont stockés sur un serveur centralisé, et les développeurs récupèrent une copie locale pour y travailler.

##### Mercurial

Mercurial est un autre système de contrôle de version décentralisé, similaire à Git, connu pour sa simplicité et ses performances. Il est plus intuitif que Git et supporte les grands projets avec des millions de fichiers.

En raison de sa popularité (les plateformes d’hébergement GitHub sont très présentes sur le web) et de mon expérience et du fait que mon entreprise l’utilise j’ai choisi Git.

Compétence validée : Versionnement\* des scripts avec Git et un dépôt Git en ligne.

#### CI/CD

Le CI/CD (Continuous Integration/Continuous Delivery ou Continuous Deployment) est une méthodologie de développement logiciel qui vise à automatiser et optimiser les étapes de construction, test, déploiement, et livraison des applications.

* CI (Continuous Integration) :
  + Processus d'intégration continue.
  + Objectif : intégrer régulièrement (quotidiennement ou plus fréquemment) le code des développeurs dans une branche partagée (généralement main ou develop).
* Avantages :
  + Tests automatisés pour détecter rapidement les bugs.
  + Réduction des conflits entre branches.
* CD (Continuous Delivery) :
  + Processus de livraison continue.
  + Objectif : préparer automatiquement le code pour qu'il puisse être déployé à tout moment.
  + Les étapes incluent : tests automatisés, packaging (build), et déploiement dans un environnement de pré-production ou staging.
* CD (Continuous Deployment) :
  + Extension de la livraison continue.
  + Objectif : déployer automatiquement en production après validation des tests, sans intervention humaine.
* Avantages : déploiements fréquents et rapides.

Il existe de nombreux outils pour réaliser le CI/CD (Jenkins, GitHub Actions, GitLab CI/CD…). J’utilise GitHub Actions car mon code est hébergé sur GitHub.

Ici le CI/CD est utilisé pour réaliser les tests lorsque le code est poussé sur GitHub.

Pour réaliser le CI/CD, on utilise GitHub Actions.

GitHub Actions est une plateforme d'intégration et de livraison continue (CI/CD) intégrée directement à GitHub. Elle permet d'automatiser des workflows (processus) en fonction d'événements tels que des push, pull requests, ou des déclencheurs programmés. Avec GitHub Actions, on peut configurer des pipelines pour tester, construire et déployer des applications.

Les instructions pour le CI/CD se font dans le dossier : «.github/workflows » que l’on va créer via github.

On définit l’action qui va déclencher le workflow et un environnement de travail ici Ubuntu et on clone le dépôt dans l’environnement d’exécution. On installe docker, on crée les conteneurs et enfin on exécute les tests.

Le résultat apparaît dans GitHub Actions sous forme d’un terminal qui affiche la liste des actions effectuées. Si le workflow est réussi, un macaron vert apparaît sur GitHub à côté du workflow.



*Figure 7. cicd réussi sur GitHub*

Compétence validée : Créer une chaîne de livraison continue d’un modèle d’intelligence artificielle en installant les outils et en appliquant les configurations souhaitées, dans le respect du cadre imposé par le projet et dans une approche MLOps\*, pour automatiser les étapes de validation, de test, de packaging\* et de déploiement du modèle.

# Conclusion

Notre RAG est désormais intégré dans le projet, les outils de surveillance (Grafana, Uptime Kuma) et de gestion des erreurs ainsi que leur champ d’application ont été définis. Le mode d’évaluation des outils d’Intelligence Artificielle (LLM as a judge) a également été défini. Enfin le processus d’intégration continu avec les tests automatisés a été mis en place.