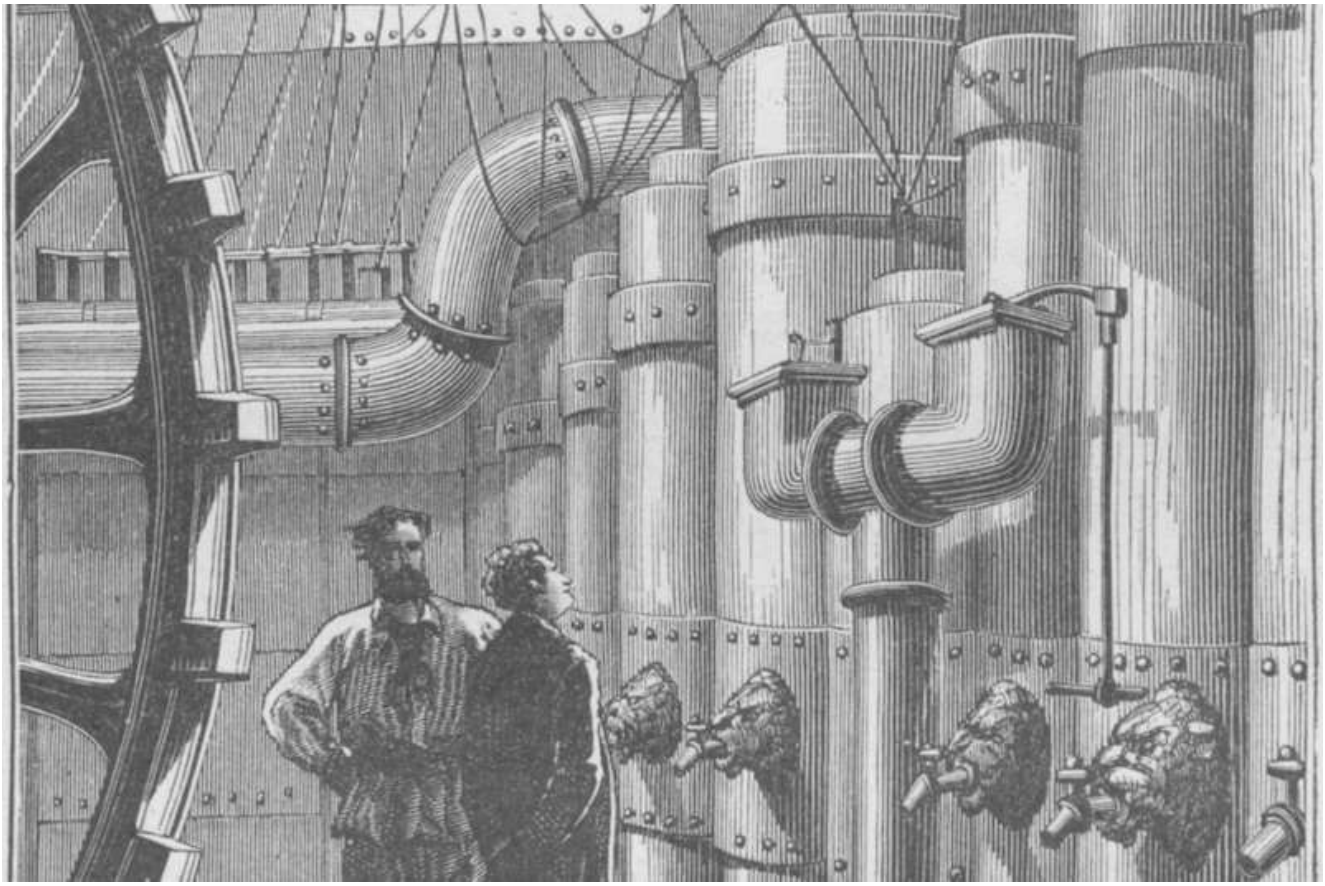
Explorer ses documents de travail avec les méthodes de la Retrieval-Augmented Generation : créer, démonter et mettre en œuvre une application Web complète

Voir les logs de : IA

13 Octobre 2025 • Stéphane Pouyllau • Moteur de recherche, Intelligence artificielle, Retrieval-Augmented Generation, RAG, NLP, LLM, Ollama, Streamlit, Application Web, Système d'information documentaire



« La chambre des machines nettement éclairée », 1867, illustration de Alphonse de Neuville & Riou pour le roman « Vingt mille lieues sous les mers » de Jules Verne, gravée par Hildibrand. J. Hetzel et Cie, « Bibliothèque d'Éducation et de Récréation, Voyages extraordinaires ». In-8°. BnF, département de la Réserve des livres rares, Rés. M-Y2-999

« Le dessin est le croquis d'un lecteur qui essaie de bien comprendre son *auteur* »
Marcello Vitali-Rosati à propos de Léonard de Vinci dans *C'est la matière qui pense*, Ed.

PURH, 2025.

En deçà des outils haut de gamme de *Pré-target RAG*, par ailleurs travaillés au sein du HN Lab, les briques permettant de déployer une application de découverte de documents et de données sur les techniques de la *Retrieval-Augmented Generation* (RAG) se multiplient fortement depuis quelques mois. Ce billet présente une expérimentation du HN Lab autour d'une brique de RAG appliquée à des documents de recherche. Il s'inscrit dans la suite du *log* « Explorer ses documents de travail avec les méthodes de la Retrieval-Augmented Generation : construire un agent conversationnel sur ses propres documents ».

Conçue avec la librairie Python *Streamlit* et déployée sur un serveur virtuel d'Huma-Num, cette application illustre, à travers un **démonstrateur volontairement simple**, comment associer les fonctions de recherche dans des documents et interfaçage en langage naturel en s'appuyant sur l'usage de modèles de langue et donc des principes des IA génératives. Le but de cette « brique » (qui n'en ai pas vraiment une en fait) et d'être volontairement simplifier pour en ouvrir le capot dans le but de réfléchir aux usages et aux limites des RAG dans le domaine de la découverte des données (CSV, tabulaires, etc.) et documents (PDF, DOCX, Markdown, ePub, etc.) des Lettres, des sciences humaines et des sciences sociales (LSHS).

Dans le cadre des travaux du HN Lab autour de l'intelligence artificielle et du programme ISIDORE 2030, nous menons plusieurs expérimentations visant à illustrer, à petite échelle, comment certaines technologies peuvent être mises en œuvre concrètement dans le champ des LSHS. L'objectif n'est pas ici de démontrer ou de convaincre, mais simplement d'explorer : montrer comment ces outils fonctionnent, quels services ils peuvent rendre ? Mais aussi et surtout quelles limites et quelles questions ces techniques soulèvent-elles ? Ce billet s'inscrit dans cette démarche de mise en démonstration à travers la présentation d'une application construite avec Streamlit et fondée sur la technique de la *Retrieval-Augmented Generation*. Nous rappelons ici qu'elle combine trois approches (en simplifiant) : Les méthodes traditionnelles des moteurs de recherche (indexation, calcul de similarité, recherche de pertinence), la vectorisation des données et des questions et la génération en langage naturel permise par les grands modèles de langage.

L'application permet d'interroger un ensemble de documents — PDF, Word, LibreOffice, Markdown, CSV. — sans passer par la création d'une base de données dédiée. Il suffit de poser une question en langage naturel, par exemple : « Quels sont les liens entre telles et telles personnes ? », pour obtenir une réponse contextualisée, accompagnée de citations issues directement des textes.

Au-delà de la démonstration et de l'explication du code, il s'agit également de mettre l'accent sur une dimension souvent difficile à présenter mais essentielle : le déploiement et l'hébergement. L'application a en effet été mise en ligne sur un serveur virtuel d'Huma-Num, illustrant la manière dont une telle infrastructure peut être mobilisée

pour tester, partager et valoriser ce type d'outils expérimentaux dans un cadre institutionnel sécurisé. Une partie du README du dépôt GitLab est consacrée à l'installation et au déploiement sur un serveur de type machine virtuelle à l'aide du serveur Web Apache.

Ainsi, cette expérience s'inscrit pleinement dans la philosophie du HN Lab : proposer des démonstrateurs pédagogiques pour les chercheurs, qui permettent de comprendre, par la pratique, les potentialités et les limites des technologies d'intelligence artificielle appliquées aux SHS.

Détail du fonctionnement de l'application

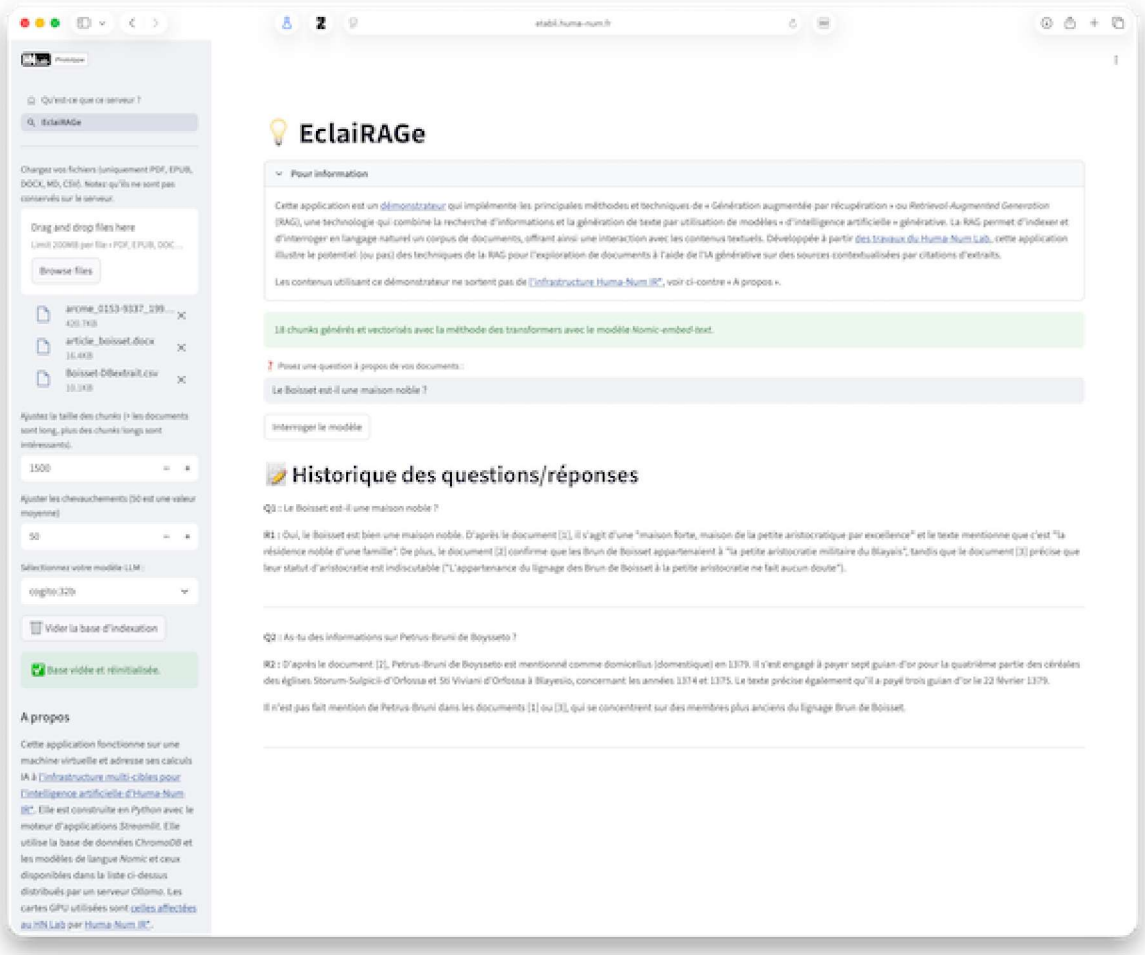
Cette application **Streamlit** qui illustre l'utilisation de la technique de **RAG (*Retrieval-Augmented Generation*)** pour interroger des données et documents textuels (PDF, EPUB, DOCX, Markdown, CSV) via un modèle de langage déployé avec l'aide d'un *juke-box* à modèles de langue (ici un **Ollama** mutualisé fonctionnant dans le cadre de l'infrastructure multicible proposée par Huma-Num IR*). Elle permet d'indexer des corpus, de générer des *embeddings*, puis de poser des questions en langage naturel avec un renvoi aux **extraits sources** utilisés.

Code

Le code est disponible dans le dépôt gitlab du HN Lab.

Démonstration

Cliquez sur l'image ci-dessous pour lancer la démonstration (ce démonstrateur est l'une des versions construites à partir du code proposé. Le code est donc une proposition qui doit être vue comme une boite à outils).



Description technique de l'application

Configuration logicielle requise (au niveau de l'application)

- **Python** 3.11 (ou supérieur)
- **Streamlit** (interface web)
- **ChromaDB** (base vectorielle, stockage persistant)
- **Ollama** (serveur de modèles LLM et embeddings) soit en local, soit via un serveur mutualisé (voir le projet en cours Ilaas).
- Bibliothèques Python : `PyPDF2` , `docx` , `ebooklib` , `bs4` , `scikit-learn` , `pandas` , `chromadb` , `ollama`

Fonctionnalités principales de l'application

- Chargement de documents au format **PDF, EPUB, DOCX, MD, CSV**

- **Découpage en chunks** avec chevauchement paramétrable
- **Indexation vectorielle** via **ChromaDB**
- **Embeddings** générés avec `nomic-embed-text` via un serveur Ollama (ou autre suivant votre serveur Ollama)
- Pose de questions en langage naturel, réponses enrichies par citations d'extraits
- Gestion de l'**historique des questions/réponses**
- Interface interactive développée en **Streamlit**

Utilisation

1. **Uploader vos documents** via la barre latérale (PDF, EPUB, DOCX, MD, CSV)
2. Régler les paramètres :
   - Taille des *chunks* (par défaut 1500 caractères)
   - Chevauchement entre *chunks* (par défaut 50)
   - Choix du modèle LLM (`cogito:32b` , `mistral:7b-instruct` , etc.) selon votre serveur Ollama.
3. Posez une question en langage naturel
4. Consultez la réponse et les **sources citées**
5. Accédez à l'**historique** complet des questions/réponses
6. Les questions/réponses sont ré-injectés dans les questions suivante (voir la question du *Prompt*)

Edition du *Prompt*

Il est possible d'adapter le *Prompt* initial directement dans l'application (voir ligne `prompt`). Il est possible de modifier le nombre de citation de sources via le `k_top` .

Détail des fonctions Python

Extraction de textes

> *Note : le nom des fonctions n'est pas stabilisé du point de vue de la langue. Une mise à jour et une homogénisation sera faite dans les semaines qui viennent.*

- **`extract_text_from_pdf(pdf_file)`** : extrait le texte d'un fichier PDF avec `PyPDF2`
- **`extract_text_from_docx(docx_file)`** : extrait le texte d'un fichier Word `.docx`
- **`extract_text_from_md(md_file)`** : lit le contenu brut d'un fichier Markdown `.md`
- **`extract_text_from_epub(uploaded_file)`** : extrait le texte des fichiers EPUB en utilisant `ebooklib` et `BeautifulSoup`
- **`extract_text_from_csv(uploaded_file)`** : extrait les valeurs d'un tableau de données au format CSV via `Pandas`

Pré-traitement

- `chunk_text(text, size=500, overlap=50)` : découpe un texte en segments (*chunks*) de longueur définie, avec un chevauchement optionnel

Embeddings et vecteurs

- `get_embedding(text: str)` : appelle le serveur Ollama pour générer un embedding avec le modèle `nomic-embed-text`. Il est possible d'utiliser d'autres modèles, suivant les versions de modèles pour *embedding*. Sans rentrer dans les détails pour cette application, la question du choix du modèle d'*embedding* est centrale dans l'opération de vectorisation. Surtout, il faut noter que c'est souvent un angle mort dans les choix possibles ou souhaités par les chercheurs qui utiliseraient ces outils. À la dimension « boite noire » des modèles s'ajoute donc une dimension « angle mort ».

RAG (Question / Réponse)

- `poser_question(question, all_chunks, chunk_sources, embeddings, top_k=3, lignes_par_chunk=5, model_name="cogito:32b")` :
  1. Génère l'embedding de la question
  2. Compare la similarité cosinus avec les chunks vectorisés
  3. Sélectionne les passages les plus proches
  4. Construit un ***prompt***, c'est-à-dire un ensemble d'ordres de commandes pour piloter le modèle de langue autour de :
     - La question posée
     - Le contexte et le fait d'utiliser les extraits découpés des documents et données vectorisées
     - Après la première question posée, l'historique des réponses données

Exemple utilisé ici est :

```
prompt = (
        "Tu es un chercheur ou une chercheure. Voici des extraits de documents numérotés :"
        f"{sources_text}\n\n"
              "Voici l'historique de la conversation :\n\n"
        f"{history_text}\n\n"
        f"En te basant uniquement sur les extraits et l'historique de la conversation, répo
        f"Question : {question}"
    )
```

Le réglage du *Prompt* et donc la façon de le composer, la gestion du nombre de citations à utiliser (`top_k`), la longueur des citations utilisées (`ligne_par_chunk`) permettent de piloter la demande faite au modèle de langue.

1. Interroge le modèle via Ollama et renvoie la réponse, les extraits utilisés et les citations utilisées (et donc ici limité à 3 citations maximum par la valeur du `top_k`)

Gestion de l'interface Streamlit

- `st.session_state.history` : enregistre l'historique des questions/réponses
- **Upload de fichiers** via `st.sidebar.file_uploader` propre à *Streamlit*
- **Paramètres possibles** : taille des chunks, chevauchement, choix du modèle LLM
- **Affichage des réponses** avec les sources citées et l'historique des Q/R

Références & crédits

- Développé dans le cadre du Huma-Num Lab par Stéphane Pouyllau
- Travaux de recherche : Antoine de Sacy (de 2022 à 2024), Adam Faci, Léa Maronet, Emile Provendier

Ressources

- Explorer ses documents de travail avec les méthodes de la Retrieval-Augmented Generation : construire un agent conversationnel sur ses propres documents, 2025
- Note sur l'expérience de l'IA au sein de l'Huma-Num Lab, 2025
- Hackathon I - Retrieval-Augmented Generation en SHS, 2024
- Quels usages du "Retrieval-augmented generation" en SHS ?, 2024
- Travaux sur les IA au sein du HN Lab, 2023

- Pour information complémentaire : la documentation de l'infrastructure multi-cible pour l'IA d'Huma-Num IR*