

Mini-série

LLM · RAG · Recherche d'information

Construire un pipeline RAG local et traçable

Construire un prototype RAG reproductible sur un corpus de cours Markdown, retrouver les passages pertinents et produire une réponse qui conserve ses sources.

Auteur **Youness Boutyour**

**DURÉE**

3 épisodes · 45 min /
épisode

NIVEAU

Intermédiaire

FORMAT

Mini-série · 3 épisodes

LLM · RAG · Recherche d'information · publié le 2026-05-03 · hybrid

Objectifs et livrables

Une base documentaire interne contient souvent la réponse recherchée, mais encore faut-il retrouver le bon passage et savoir d'où il vient. Nous allons construire un assistant de recherche sur des documents MLOps, capable de montrer ses sources et, si vous le souhaitez, de faire formuler une réponse par un modèle local.

Le parcours commence par un corpus Markdown accompagné de sa provenance. Nous le découperons en passages, construirons un index TF-IDF et lancerons plusieurs recherches. Cette première méthode rend le classement facile à examiner et fonctionne sans clé API ni GPU. Vous pourrez ensuite ajouter la génération locale en conservant les passages sélectionnés dans le contexte.

Nous prendrons le temps de lire les scores et les sources dans `outputs/last-retrieval.json`. Vous comparerez une recherche réussie à un cas moins pertinent pour comprendre l'effet du vocabulaire et du découpage. Les identifiants et les métadonnées permettront de suivre chaque passage jusqu'au document d'origine.

À la fin, vous aurez un pipeline local inspectable, depuis le corpus jusqu'à la réponse. Celle-ci citera les passages `[S1]`, `[S2]` ou indiquera que l'information est insuffisante. La trace de récupération vous aidera à choisir quoi améliorer avant de changer de modèle.

RÉSULTATS DES ÉTAPES · RAG

01 · CORPUS

```
data/mlops.md
data-management.md

chunks: 4

✓ source + section
provenance conservée
```

02 · RETRIEVAL

```
[S1] Validation 0.62
[S2] MLOps 0.31
[S3] Provenance 0.00

✓ last-retrieval.json
scores inspectables
```

03 · RÉPONSE

```
Réponse [S1][S2]

ou

information
insuffisante

aucune invention
```

Progression de construction du corpus vers la réponse citée.

RÉSULTAT DE RÉFÉRENCE · RETRIEVAL

Question : que vérifier avant de promouvoir un modèle ?

PASSAGES CONSERVÉS DANS LAST-RETRIEVAL.JSON

[S1] mlops.md / Validation score 0.62
[S2] mlops.md / ML0ps score 0.31
[S3] data-management.md score 0.00
sources + scores + question = preuve inspectable

RÉPONSE CONTRÔLÉE

Réponse [S1][S2]
ou
information
insuffisante.

Les scores sont des exemples : conserver les valeurs effectivement produites par votre corpus.

La qualité de la récupération est évaluée avant celle du LLM.

Exemple de sortie traçable avec citations et abstention hors corpus.

Conserver la preuve pendant la recherche

Schéma du RAG local qui relie le corpus, les passages classés, le contexte et la réponse avec citations.



La récupération conserve les identifiants nécessaires pour remonter de la réponse au document source.

Lire le flux avant de coder

Le schéma distingue la récupération de la génération. Le retriever choisit des passages à partir du corpus ; le modèle ne devrait recevoir que ce contexte sélectionné. Les identifiants de source traversent tout le pipeline pour que la réponse puisse être relue document par document.

Le cas hors corpus est un contrôle aussi important que la question réussie. Un score faible ou une absence de passage ne prouve pas que le modèle est incapable de répondre ; cela indique que la base disponible ne justifie pas une réponse. La sortie attendue est alors une abstention explicite, accompagnée de la trace qui permet d'améliorer le corpus ou le découpage.

1. Préparer l'environnement

Commencez par la version sans générateur : elle permet de suivre la préparation des documents et le classement des passages avec les seules dépendances de recherche. Le modèle local sera ajouté après vérification de ces résultats. Cette séparation évite de chercher dans le prompt une erreur qui provient en réalité d'un fichier absent ou d'un contexte mal sélectionné.

Exécutez les commandes depuis le dossier du projet et gardez les versions installées avec le code. Le fichier issu de `pip freeze` décrit cet environnement ; il ne garantit pas une installation identique sur tous les systèmes. Vérifiez l'import des bibliothèques avant de créer les données, puis assurez-vous que les exports sont écrits dans le dossier prévu.

Créez un dossier de travail et un environnement virtuel :

```
Terminal · Bash

mkdir -p rag-local-tracable/{data,src,outputs}
cd rag-local-tracable
python3 -m venv .env
source .env/bin/activate
python -m pip install --upgrade pip
python -m pip install scikit-learn==1.7.1
python -m pip freeze > requirements.lock.txt
```

La commande `freeze` enregistre les versions réellement installées. Sur une autre machine, la reconstruction se fait avec :

```
Terminal · Bash

python3 -m venv .env
source .env/bin/activate
python -m pip install -r requirements.lock.txt
```

Vérifiez l'environnement :

```
Terminal · Bash

python --version
python -c "import sklearn; print('scikit-learn', sklearn.__version__)"
```

Sortie obtenue

Sortie / exemple

```
Python 3.11.x  
scikit-learn 1.7.x
```

Commande de vérification

Conservez `requirements.lock.txt` et la sortie de version. Si la version affichée diffère du fichier de verrouillage, régénérez l'environnement avant de comparer les scores.

2. Créer un corpus avec provenance

Les titres de niveau deux servent de frontières pour les passages. Donnez à chaque section un thème suffisamment autonome pour qu'elle puisse être comprise sans le document entier. Le couple fichier-section permettra ensuite de retrouver une source, tandis que l'identifiant du fragment distinguera plusieurs passages issus du même fichier.

Parcourez les documents avant l'indexation et associez les questions de test aux sections qui devraient y répondre. Cette référence manuelle permettra de repérer une récupération plausible mais hors sujet. Si vous déplacez ou ajoutez une section, les identifiants fondés sur sa position peuvent changer : ne les traitez pas comme des identifiants de contenu immuables.

Créez `data/mlops.md` :

Markdown

MLOps

Le MLOps relie les données, le code, les paramètres, les artefacts et le service dans une chaîne reproductible. Une expérience n'est pas reproductible si son environnement ou ses données ne peuvent pas être reconstruits. Le pipeline conserve donc un identifiant de version pour le jeu d'entraînement et pour le modèle promu.

Validation

Avant de promouvoir un modèle, il faut vérifier le contrat des données, les métriques, les tests de service et les conditions de retour arrière. Une baisse de rappel ou une colonne manquante bloque la promotion jusqu'à résolution.

Observabilité

Le service expose la latence, le taux d'erreur, le volume de prédictions et la distribution des scores. Ces indicateurs sont comparés à une ligne de base pour détecter un changement de comportement après le déploiement.

Retour arrière

Chaque version publiée pointe vers un artefact immuable et une configuration rejouable. En cas de dérive ou d'incident, l'équipe peut restaurer la version précédente sans reconstruire le modèle à partir de données différentes.

Créez aussi `data/data-management.md` :

● ● ● Markdown

Management de la data

La gouvernance des données décrit les responsabilités, les usages autorisés, la qualité visée, les règles d'accès et la durée de conservation. Elle relie un produit de données à son propriétaire, à ses consommateurs et à ses règles de contrôle avant toute réutilisation analytique.

Provenance

Une décision fiable doit pouvoir relier sa donnée source, sa transformation, sa date de production et la personne ou le système responsable. Le catalogue conserve aussi la version du schéma, le niveau de sensibilité et la durée de conservation afin de reconstruire le chemin entre la source et le rapport.

Qualité

Un jeu de données exploitable définit des règles sur les valeurs manquantes, les doublons, les plages numériques et les identifiants. Les contrôles sont exécutés à l'ingestion puis avant chaque publication afin de distinguer une anomalie de source d'une transformation volontaire.

Accès et conservation

Les droits sont accordés selon le besoin d'en connaître et révisés lorsque le propriétaire ou l'usage change. Les données personnelles sont minimisées, chiffrées au repos et supprimées à la fin de la durée de conservation prévue.

Chaque passage doit conserver au minimum le nom du fichier, le titre de section et son numéro. Cette métadonnée sera affichée dans la réponse afin de rendre la provenance inspectable.

Sortie obtenue

● ● ● Terminal · Bash

```
find data -maxdepth 1 -name '*.md' -print | sort
```

Sortie / exemple

```
data/data-management.md
data/mlops.md
```

Commande de vérification

Ouvrez les deux fichiers et vérifiez qu'ils contiennent les sections utilisées dans les questions. Le contrôle porte sur le corpus versionné, pas seulement sur le nombre de fichiers détectés.

3. Implémenter l'indexation et la récupération

Le découpage produit une liste de passages avec leur provenance ; TF-IDF transforme ensuite leurs textes en une matrice. La question est projetée avec ce même vectoriseur pour comparer des coordonnées compatibles. Les scores sont triés, puis filtrés par le seuil : le résultat contient jusqu'à trois passages, éventuellement aucun si le corpus ne fournit pas de correspondance suffisante.

Examinez la récupération avant toute génération : source, section, score et texte doivent raconter la même chose. Le score n'est pas une probabilité de vérité et le seuil n'est pas universel. [last-retrieval.json](#) décrit le dernier appel, pas un historique complet. Une question sans résultat doit conduire à l'abstention, plutôt qu'à l'ajout artificiel du passage le moins mauvais.

Créez `src/rag.py` :

Python

```
from __future__ import annotations

import argparse
import json
import os
import re
import urllib.request
from pathlib import Path

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

ROOT = Path(__file__).resolve().parents[1]
DATA_DIR = ROOT / "data"
OUTPUT_DIR = ROOT / "outputs"
FRENCH_STOP_WORDS = ["a", "à", "au", "aux", "de", "des", "du", "en", "est", "et", "la", "le", "les", "pour",
"que", "quel", "quelle", "quels", "quelles", "un", "une"]
MIN_RELEVANCE = 0.1
```

```

def split_markdown(path: Path) → list[dict[str, str]]:
    text = path.read_text(encoding="utf-8")
    sections = re.split(r"(?m)^## +", text)
    chunks = []
    for index, section in enumerate(sections):
        content = section.strip()
        if not content:
            continue
        lines = content.splitlines()
        title = lines[0].strip("# ") if lines else path.stem
        body = "\n".join(lines[1:]).strip() if len(lines) > 1 else content
        chunks.append({
            "source": path.name,
            "section": title,
            "chunk_id": f"{path.stem}-{index:02d}",
            "text": f"{title}\n{body}",
        })
    return chunks

def load_corpus() → list[dict[str, str]]:
    chunks = []
    for path in sorted(DATA_DIR.glob("*.md")):
        chunks.extend(split_markdown(path))
    if not chunks:
        raise RuntimeError("Le dossier data/ ne contient aucun fichier Markdown.")
    return chunks

def retrieve(question: str, top_k: int = 3) → list[dict[str, str | float]]:
    chunks = load_corpus()
    vectorizer = TfidfVectorizer(lowercase=True, strip_accents="unicode", stop_words=FRENCH_STOP_WORDS)
    matrix = vectorizer.fit_transform([chunk["text"] for chunk in chunks])
    query = vectorizer.transform([question])
    scores = cosine_similarity(query, matrix).ravel()
    ranked = scores.argsort()[::-1][:top_k]
    return [
        {**chunks[index], "score": round(float(scores[index]), 4)}
        for index in ranked
        if scores[index] ≥ MIN_RELEVANCE
    ]

def build_context(results: list[dict[str, str | float]]) → str:
    blocks = []
    for position, result in enumerate(results, start=1):
        blocks.append(
            f"[S{position}] {result['source']} - {result['section']} "
            f"(score={result['score']})\n{result['text']}"
        )
    return "\n\n".join(blocks)

def generate_with_ollama(question: str, context: str) → str:

```

```

model = os.getenv("OLLAMA_MODEL", "llama3.2:3b")
payload = json.dumps({
    "model": model,
    "stream": False,
    "prompt": (
        "Réponds uniquement avec les informations du CONTEXTE. "
        "Si le contexte ne suffit pas, réponds : information insuffisante. "
        "Cite les sources [S1], [S2], etc.\n\n"
        f"CONTEXTE:\n{context}\n\nQUESTION:\n{question}"
    ),
}).encode("utf-8")
request = urllib.request.Request(
    "http://127.0.0.1:11434/api/generate",
    data=payload,
    headers={"Content-Type": "application/json"},
)
with urllib.request.urlopen(request, timeout=120) as response:
    return json.loads(response.read())["response"]

def main() → None:
    parser = argparse.ArgumentParser()
    parser.add_argument("question")
    parser.add_argument("--top-k", type=int, default=3)
    parser.add_argument("--ollama", action="store_true", help="Générer une réponse locale avec Ollama")
    args = parser.parse_args()

    results = retrieve(args.question, max(1, args.top_k))
    context = build_context(results)
    OUTPUT_DIR.mkdir(exist_ok=True)
    (OUTPUT_DIR / "last-retrieval.json").write_text(
        json.dumps({"question": args.question, "results": results}, ensure_ascii=False, indent=2) + "\n",
        encoding="utf-8",
    )

    if not results:
        print("information insuffisante")
        return

    print("Sources récupérées :")
    for position, result in enumerate(results, start=1):
        print(f"[S{position}] {result['source']} / {result['section']} - score {result['score']}")
    print("\nContexte transmis au générateur :\n")
    print(context)
    if args.ollama:
        print("\nRéponse Ollama :\n")
        print(generate_with_ollama(args.question, context))

if __name__ == "__main__":
    main()

```

Lancez une première recherche :

 Terminal · Bash

```
python src/rag.py "Que faut-il vérifier avant de promouvoir un modèle ?"
```

La commande affiche une sortie de récupération ; les scores exacts peuvent varier selon la version de scikit-learn :

 Sortie / exemple

```
Sources récupérées :  
[S1] mlops.md / Validation – score 0.6...
```

Le fichier `outputs/last-retrieval.json` constitue la première trace reproductible : il conserve la question, les passages sélectionnés, les sources et les scores. Ne publiez pas automatiquement ce fichier s'il contient des données privées.

Sortie obtenue

 Terminal · Bash

```
test -s outputs/last-retrieval.json && python -m json.tool outputs/last-retrieval.json >/dev/null
```

 Sortie / exemple

```
Sources récupérées :  
[S1] mlops.md / Validation – score <mesure-locale>  
[S2] mlops.md / MLOps – score <mesure-locale>
```

Commande de vérification

Conservez `outputs/last-retrieval.json` et comparez chaque source affichée au corpus. Une réponse est invalide si elle ne peut pas remonter vers un passage identifié.

4. Ajouter un générateur local facultatif

Le générateur reçoit uniquement le contexte retenu et la consigne de réponse. Il peut reformuler ces informations, mais ne doit pas transformer une source voisine du sujet en justification d'un fait absent. Vérifiez d'abord que le service local répond et que le modèle demandé est disponible, puis ajoutez l'option d'appel sans modifier simultanément les paramètres de récupération.

Comparez la réponse aux passages portant les repères [S1], [S2] et suivants. Chaque repère doit correspondre à une source présente, et son texte doit réellement soutenir l'affirmation associée. Une erreur de connexion au générateur ne remet pas en cause l'indexation déjà vérifiée ; une réponse mal étayée demande, elle, de séparer le diagnostic de récupération du diagnostic de rédaction.

Testez d'abord le retrieveur seul. Ensuite, dans un second terminal :

Terminal · Bash

```
ollama pull llama3.2:3b
OLLAMA_MODEL=llama3.2:3b python src/rag.py "Qu'est-ce que la provenance ?" --ollama
```

Pour garder la reproduction portable, conservez le corpus, le fichier de dépendances, le nom du modèle et les paramètres de récupération (`top_k`, méthode de découpage, date du corpus). Un modèle local ne garantit pas à lui seul la qualité des réponses : les passages et les citations doivent être évalués séparément.

Sortie obtenue

Sortie / exemple

Réponse Ollama :
La décision doit s'appuyer sur [S1] et [S2].

Si le corpus ne contient pas l'information, la sortie est exactement `information insuffisante`.

Commande de vérification

Capturez le terminal montrant la question, les sources et la réponse. Conservez le nom du modèle local et le fichier `last-retrieval.json` ; ne présentez pas une réponse LLM sans ses passages sources.

Vérification finale — évaluer le système

Créez `data/questions.json` :

JSON

```
[
  {"question": "Que relie le MLOps ?", "source": "mlops.md"},
  {"question": "Que décrit la gouvernance ?", "source": "data-management.md"},
  {"question": "Quelle information est absente du corpus ?", "source": null}
]
```

Pour chaque question, vérifiez manuellement :

- le premier passage est-il réellement utile ?
- la source de référence apparaît-elle dans le top-k ?
- le système sait-il signaler l'absence d'information ?
- la réponse distingue-t-elle le texte source de l'interprétation ?

Le tutoriel est réussi lorsque la récupération, la provenance et l'abstention peuvent être examinées indépendamment de la qualité stylistique du LLM.

Exemples de sorties

Pour chaque question, conservez :

Sortie / exemple

question

→ texte exact de la question

outputs/last-retrieval.json

→ passages, sources et scores

réponse

→ citations [S1], [S2] ou abstention

Une exécution réussie doit permettre de remonter de la réponse vers le passage source. Pour la question sans information suffisante, la sortie affichée est `information insuffisante` plutôt qu'une réponse inventée.

Dépannage et diagnostics

Symptôme	Cause probable	Vérification
aucun passage utile	corpus vide, découpage inadéquat ou question hors domaine	inspecter <code>data/</code> et afficher les passages avant d'activer Ollama
scores différents	version de scikit-learn, corpus ou paramètres modifiés	conserver <code>requirements.txt</code> , le hash des fichiers et <code>top_k</code>
réponse sans citation	prompt ou contexte mal transmis au LLM	rejouer sans <code>--ollama</code> , puis inspecter <code>last-retrieval.json</code>
le modèle invente une information	absence de règle d'abstention	imposer la réponse d'insuffisance et ajouter un test négatif

Limites et prolongements

TF-IDF est un point de départ simple, mais il ne comprend pas les synonymes comme un modèle d'embeddings. Pour une version suivante, remplacez le vectoriseur par des embeddings, ajoutez une base vectorielle persistante, filtrez par métadonnées et comparez plusieurs stratégies de découpage. Conservez toutefois la même sortie d'évaluation afin de comparer les versions.

Références

- [scikit-learn — TfidfVectorizer](#)
- [Ollama — documentation officielle](#)