

Tutoriel

RAG · LLM local · Provenance

Construire un mini-RAG local et traçable

Relier une recherche de passages à une réponse générée localement, avec citations et refus lorsque le corpus ne suffit pas.

Auteur **Youness Boutyour**

**DURÉE**

60 minutes

NIVEAU

Intermédiaire

FORMAT

Tutoriel pratique

RAG · LLM local · Provenance · publié le 2026-07-12 · hybrid

Objectifs et livrables

Comment poser une question sur une procédure interne et retrouver le passage qui justifie la réponse ? Nous allons construire un mini-RAG local à partir de deux documents, en suivant chaque étape jusqu'à une réponse accompagnée de ses sources.

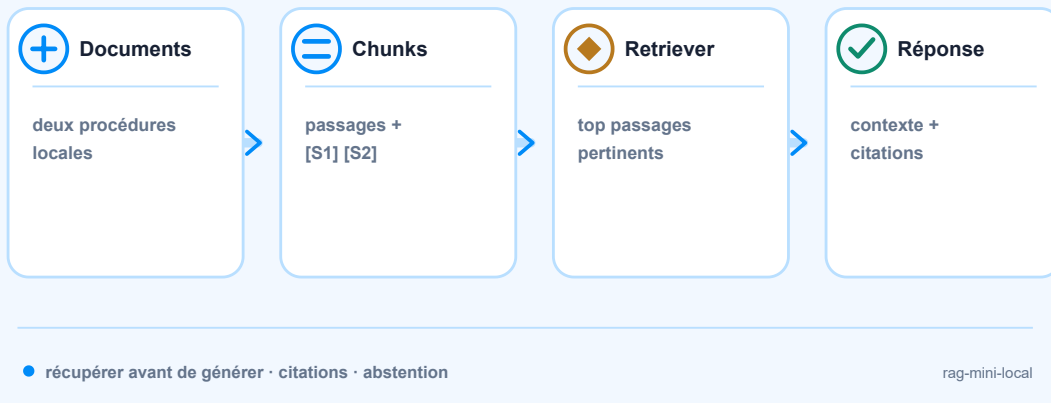
Vous préparerez le corpus et quelques questions, puis vous récupérerez les passages susceptibles de répondre. Nous examinerons ces passages avant de les fournir au modèle local. Leurs identifiants, comme `[S1]`, permettront ensuite de retrouver les éléments cités dans la réponse.

Un essai hors corpus complètera le parcours : il montrera comment signaler que les documents ne contiennent pas l'information demandée. Vous pourrez ainsi comparer un contexte pertinent à un contexte insuffisant et comprendre l'effet de la récupération sur la génération.

Le fichier `outputs/last-answer.json` conservera la question, les sources, le contexte et la réponse. Vous disposerez d'un prototype facile à inspecter, ainsi que d'une trace pour choisir les documents ou les règles de recherche à améliorer.

Alimenter une réponse avec le bon contexte

Schéma du mini-RAG qui indexe des documents, récupère les passages pertinents et refuse une question hors corpus.



La génération reçoit un contexte sélectionné et peut s'abstenir lorsque le corpus ne suffit pas.

Lire le flux avant de coder

Le schéma sépare l'indexation, la récupération et la génération. Les documents deviennent des passages identifiés ; le retriever choisit ceux qui répondent le mieux à la question ; le modèle reçoit uniquement ce contexte. Une citation comme `[S1]` doit toujours pouvoir être reliée à un passage concret.

Le test hors corpus vérifie la capacité d'abstention. Si aucun passage ne soutient la réponse, produire une phrase plausible serait un défaut de contrôle. La trace last-answer.json doit vous permettre de voir la question, les sources retenues, le contexte envoyé et la réponse finale.

1. Préparer l'environnement

Le projet a deux parties indépendantes : la récupération de passages et la génération d'une réponse. Les dépendances installées ici suffisent à préparer la recherche et les appels HTTP du programme ; elles ne démarrent pas un modèle local. Le mode mock permet de vérifier les fichiers et les embranchements du programme avant de configurer ce service.

Gardez les commandes à la racine du dossier, car les scripts recherchent `data/` et écrivent dans `outputs/`. Le contrôle qui liste les documents se lance après leur création à l'étape suivante : une erreur sur le nombre de fichiers à ce stade indique simplement que le corpus n'a pas encore été préparé. L'import des bibliothèques peut, lui, être vérifié immédiatement.

```
Terminal · Bash

mkdir -p rag-local-tracable/{data,src,outputs}
cd rag-local-tracable
python3 -m venv .venv && source .venv/bin/activate
python -m pip install scikit-learn requests
```

Sortie obtenue

```
Sortie / exemple

Successfully installed scikit-learn requests
```

Commande de vérification

```
Terminal · Bash

python -c "import sklearn, requests; print('rag stack ok')"
```

Code de l'étape

```
Python

from pathlib import Path

documents = sorted(Path("data").glob("*.*md"))
```

```
assert len(documents) >= 2
for document in documents:
    print(document.name, document.read_text(encoding="utf-8").splitlines()[0])
```

2. Préparer le corpus et les questions

Les sections séparent des sujets qui répondent à des questions différentes : validation, suivi, retour arrière, accès ou qualité. Cette organisation évite de fournir au générateur deux documents entiers pour une seule question. Le titre du document et celui de la section donnent également un contexte au passage lorsqu'on examine les sources récupérées.

Les deux questions documentées vérifient que la recherche retrouve une information présente ; la troisième vérifie la conduite à tenir lorsqu'elle manque. Une question hors corpus ne doit pas être remplacée par une réponse de culture générale. Gardez ces essais identiques entre mock et génération réelle afin de localiser les différences : sélection des passages d'un côté, rédaction de la réponse de l'autre.

Créez `data/mlops.md`, `data/gouvernance.md` et `data/questions.json`. Chaque document contient plusieurs sections opérationnelles ; ajoutez deux questions documentées et une question dont la réponse n'existe pas.

Code de l'étape

● ● ● Python

```
import json
from pathlib import Path

Path("data/mlops.md").write_text("# MLOps\n\n## Validation\nLes métriques, le contrat des données et les tests de service sont vérifiés avant promotion.\n\n## Suivi\nLa latence, le taux d'erreur et la distribution des prédictions sont comparés à une ligne de base.\n\n## Retour arrière\nL'artefact et la configuration de chaque version sont conservés.\n", encoding="utf-8")
Path("data/gouvernance.md").write_text("# Gouvernance\n\n## Accès\nChaque rôle reçoit uniquement les permissions nécessaires.\n\n## Qualité\nLes valeurs manquantes, les doublons et le schéma sont contrôlés.\n\n## Sources\nChaque réponse cite sa source et chaque table conserve son propriétaire.\n", encoding="utf-8")
Path("data/questions.json").write_text(json.dumps([
    {"question": "Que vérifie la gouvernance ?"},
    {"question": "Que faut-il suivre en MLOps ?"},
    {"question": "Quelle est la recette du couscous ?"},
], ensure_ascii=False, indent=2), encoding="utf-8")
assert len(list(Path("data").glob("*.md"))) == 2
```

Sortie obtenue

Le corpus est lisible et les questions distinguent information connue et inconnue.

Sortie / exemple

```
documents=2 sections=6 readable=true known_queries=2 unknown_query_supported=true
```

Commande de vérification

Terminal · Bash

```
python src/rag.py --mock --check-corpus
```

3. Récupérer et générer avec citations

Le vectoriseur apprend le vocabulaire et les poids sur les passages, puis transforme la question dans ce même espace. Le cosinus permet de les classer ; le seuil élimine les rapprochements trop faibles avant de construire le contexte. Demander trois résultats signifie ici « au plus trois passages au-dessus du seuil », pas « trois sources obligatoires ».

Le seuil `0.45` est une règle de ce petit exemple, pas un niveau de confiance universel. Inspectez le passage retenu lorsqu'une question documentée est refusée et recherchez des correspondances accidentelles lorsqu'une question inconnue reçoit une source. Une citation comme `[S1]` doit désigner un passage fourni au modèle ; demander des citations dans le prompt ne garantit pas à lui seul que chaque affirmation est justifiée.

Récupérez trois passages avec TF-IDF, construisez un contexte `[S1]`, `[S2]`, puis appelez Ollama si `--ollama` est présent. Le prompt interdit toute information absente du contexte.

Sortie obtenue

Sortie / exemple

```
question="Que vérifie la gouvernance ?" sources=1 answer_status=grounded
question="Que faut-il suivre en MLOps ?" sources=1 answer_status=grounded
question="Quelle est la recette du couscous ?" sources=0 answer="information insuffisante"
```

Ajoutez `--ollama` pour appeler l'endpoint `/v1/chat/completions`. La réponse et les sources restent enregistrées dans `outputs/last-answer.json`.

Code de l'étape

Python

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

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 = 0.45
chunks = load_chunks()
vectorizer = TfidfVectorizer(lowercase=True, strip_accents="unicode", stop_words=FRENCH_STOP_WORDS)
matrix = vectorizer.fit_transform([chunk["text"] for chunk in chunks])
scores = cosine_similarity(vectorizer.transform([question]), matrix).ravel()
for index in scores.argsort()[::-1][:3]:
    if scores[index] < MIN_RELEVANCE:
        continue
    print(chunks[index]["source"], round(float(scores[index]), 4))
```

Commande de vérification

Terminal · Bash

```
python src/rag.py --mock --question "Que vérifie la gouvernance ?"
python src/rag.py --mock --question "Que faut-il suivre en MLOps ?"
python src/rag.py --mock --question "Quel est le salaire du directeur ?"
```

4. Inspecter la trace

La trace relie quatre éléments : la question posée, les passages sélectionnés, le contexte transmis et la réponse finale. Commencez par retrouver chaque identifiant cité dans la liste des sources, puis lisez le passage correspondant. Une source correctement numérotée peut être sans rapport avec une affirmation ; la présence d'une citation ne remplace donc pas cette vérification du contenu.

Le fichier `last-answer.json` décrit la dernière exécution et peut être remplacé par la suivante. Consultez-le après chaque question, ou conservez une copie distincte pour comparer plusieurs essais. L'assertion finale détecte une réponse sans aucune source, sauf abstention ; elle ne vérifie ni la fidélité de toutes les phrases ni l'exhaustivité des sources.

Ouvrez `outputs/last-answer.json` et vérifiez la correspondance entre chaque citation et un passage. Comparez les réponses à `data/expected.json`.

Sortie obtenue

La trace est suffisante pour reproduire la réponse sans consulter un log externe.

Sortie / exemple

```
outputs/last-answer.json question sources answer_status
```

Commande de vérification

Terminal · Bash

```
python src/evaluate.py --answers outputs/last-answer.json --expected data/expected.json
python -m json.tool outputs/last-answer.json >/dev/null
```

Code de l'étape

Python

```
payload = {
    "question": question,
    "sources": sources,
    "context": context,
    "answer": answer,
}
Path("outputs/last-answer.json").write_text(
    json.dumps(payload, ensure_ascii=False, indent=2), encoding="utf-8"
)
assert payload["answer"] == "information insuffisante" or payload["sources"]
```

Exemples de sorties

Le mode mock produit deux réponses citées, un refus hors corpus et un fichier JSON complet. Ollama ajoute la génération locale mais ne change pas les sources récupérées.

Dépannage et diagnostics

Si les citations ne correspondent pas au contexte, marquez la réponse invalide. Ne corrigez pas le texte manuellement après génération.

Vérification finale

- ☐ deux documents indexés ;
- ☐ deux réponses citées ;

- ☐ question hors corpus refusée ;
- ☐ trace JSON consultable.

Limites et prolongements

TF-IDF est un baseline. Une version suivante peut remplacer la récupération par des embeddings, puis comparer rappel et précision sur un jeu annoté.

Références

- [Ollama — API generate](#)
- [scikit-learn — TF-IDF](#)
- [Wikimedia Commons — RAG schema.svg \(CC BY-SA 4.0\)](#)