

RAG et embeddings : comprendre et implémenter



Le **RAG (Retrieval-Augmented Generation)** est une technique révolutionnaire qui combine **la récupération d'informations** et **la génération de texte** pour améliorer la précision des modèles de langage (LLM). Les **embeddings** jouent un rôle **central** dans le RAG en permettant de représenter les mots, phrases et documents sous forme de **vecteurs numériques**. Cela permet aux modèles d'IA de comprendre le sens et les relations entre les concepts.

Pourquoi cet article ?

- Comprendre le **fonctionnement du RAG** et son utilité.
- Découvrir le **rôle des embeddings** dans le RAG.
- Apprendre à **implémenter le RAG** avec des exemples concrets.

□ Difficulté : ★★★☆☆ (3/5)

□ **Pédagogique** : Analogies visuelles, exemples concrets, et schémas explicites.

△ **Technique** : Concepts avancés (embeddings, similarité cosinus, bases vectorielles) et manipulation de bibliothèques Python (Sentence Transformers, FAISS, LangChain).

□ **Vocabulaire** : Termes comme **chunking**, **vector store**, **retriever**, **similarité cosinus**, ou **fine-tuning** nécessitent des bases en IA et en traitement des données.

□ Prérequis recommandés

□ **Statistiques/NLP** : Notions de vecteurs, distances, et traitement du langage naturel.

□ **Python intermédiaire** : Manipulation de bibliothèques (LangChain, FAISS, Hugging Face).

□ **Gestion de données** : Prétraitement de texte, segmentation, et stockage vectoriel.

□ **Environnement technique** : Accès à un LLM (local ou API), Ollama installé, GPU recommandé.

1. Qu'est-ce qu'un embedding ?

Un **embedding** est une technique qui permet de **représenter des mots, des phrases ou des documents sous forme de vecteurs numériques**. Ces vecteurs capturent le **sens** et les **relations** entre les éléments qu'ils représentent.

1.1. Analogies

Imaginez que chaque mot est une **étoile dans l'espace**. Les embeddings sont comme des **coordonnées** qui placent chaque étoile (mot) de manière à ce que :

- Les étoiles **proches** ont des **sens similaires**.
- Les étoiles **éloignées** ont des **sens différents**.

Exemple :

- Le mot **“roi”** sera proche de **“reine”** et **“prince”**.
- Le mot **“pomme”** sera proche de **“fruit”** et **“banane”**, mais éloigné de **“roi”**.

1.2. Pourquoi utiliser des embeddings ?

Les ordinateurs ne comprennent pas les mots directement. Ils ont besoin de **nombres** pour traiter l'information. Les embeddings permettent de :

- **Transformer du texte en nombres** que les modèles d'IA peuvent comprendre.
- **Capturer le sens et les relations** entre les mots (ex. : “roi” - “reine” ≈ “homme” - “femme”).
- **Réduire la dimension** des données pour les rendre plus faciles à manipuler.

1.3. Exemple concret

Voici un exemple de mots et leurs embeddings (simplifiés en 2D) :

Mot	Embedding (x, y)
roi	(0.2, 0.8)
reine	(0.3, 0.7)
homme	(0.25, 0.75)
femme	(0.35, 0.65)
pomme	(-0.1, 0.1)
banane	(-0.2, 0.2)

Observations :

- “roi” et “reine” sont proches (sens similaire).
- “homme” et “femme” sont proches de “roi” et “reine” (relations de genre).
- “pomme” et “banane” sont proches (tous deux des fruits), mais éloignés des mots liés à la royauté.

```
      reine (0.3, 0.7)
      /
     /
roi (0.2, 0.8)
  \
   \
    homme (0.25, 0.75)

pomme (-0.1, 0.1) ----- banane (-0.2, 0.2)
```

2. Comment fonctionnent les embeddings ?

Étape 1 : Représentation initiale

Chaque mot est d'abord représenté par un **identifiant unique** (ex. : un nombre). Par exemple :

- “roi” → 1
- “reine” → 2
- “pomme” → 3

Mais cette représentation ne capture **aucun sens** !

Étape 2 : Transformation en vecteurs

Les embeddings transforment chaque mot en un **vecteur** (une liste de nombres). Par exemple :

- “roi” → [0.2, 0.8, -0.5, 0.1, ...]
- “reine” → [0.3, 0.7, -0.4, 0.2, ...]
- “pomme” → [-0.1, 0.1, 0.9, -0.3, ...]

Ces vecteurs sont appris par des modèles d'IA (comme Word2Vec, GloVe, ou BERT) de manière à ce que :

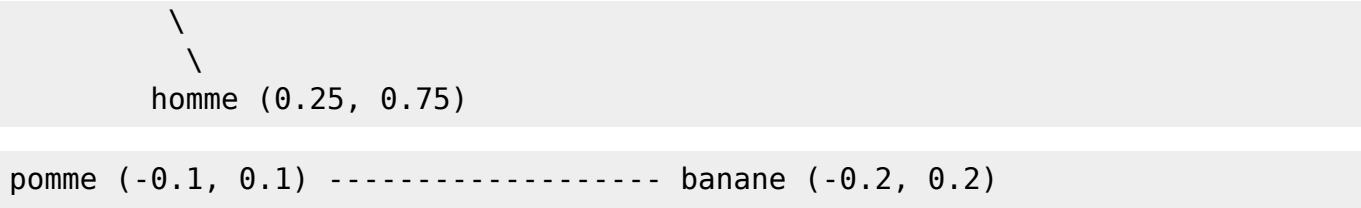
- Les mots **similaires** aient des vecteurs **proches**.
- Les mots **différents** aient des vecteurs **éloignés**.

Étape 3 : Visualisation dans un espace

On peut visualiser ces vecteurs dans un **espace à 2 ou 3 dimensions** (même si en réalité, ils ont souvent 50 à 300 dimensions).

Exemple visuel :

```
      reine (0.3, 0.7)
      /
     /
roi (0.2, 0.8)
```



3. Le RAG : Retrieval-Augmented Generation

Le **RAG** est une technique qui combine :

- **La récupération d'informations** (retrieval) à partir d'une base de données.
- **La génération de texte** (generation) par un modèle de langage (LLM).

Pourquoi le RAG ?

- **Précision** : Réduit les hallucinations des LLM en s'appuyant sur des données externes.
- **Actualité** : Permet d'utiliser des informations récentes ou spécifiques à un domaine.
- **Transparence** : Les sources utilisées peuvent être citées, améliorant la confiance dans les réponses.

4. Étapes du RAG et rôle des embeddings

Étape 1 : Indexation des documents

- **Objectif** : Préparer une base de données de documents pour la recherche.
- **Processus** :
 1. Les documents sont **découpés en morceaux** (chunks).
 2. Chaque chunk est **transformé en un vecteur d'embedding** à l'aide d'un modèle (ex. : Sentence-BERT).
 3. Les vecteurs sont **stockés dans une base de données vectorielle** (ex. : FAISS, Chroma, Weaviate).

Exemple :

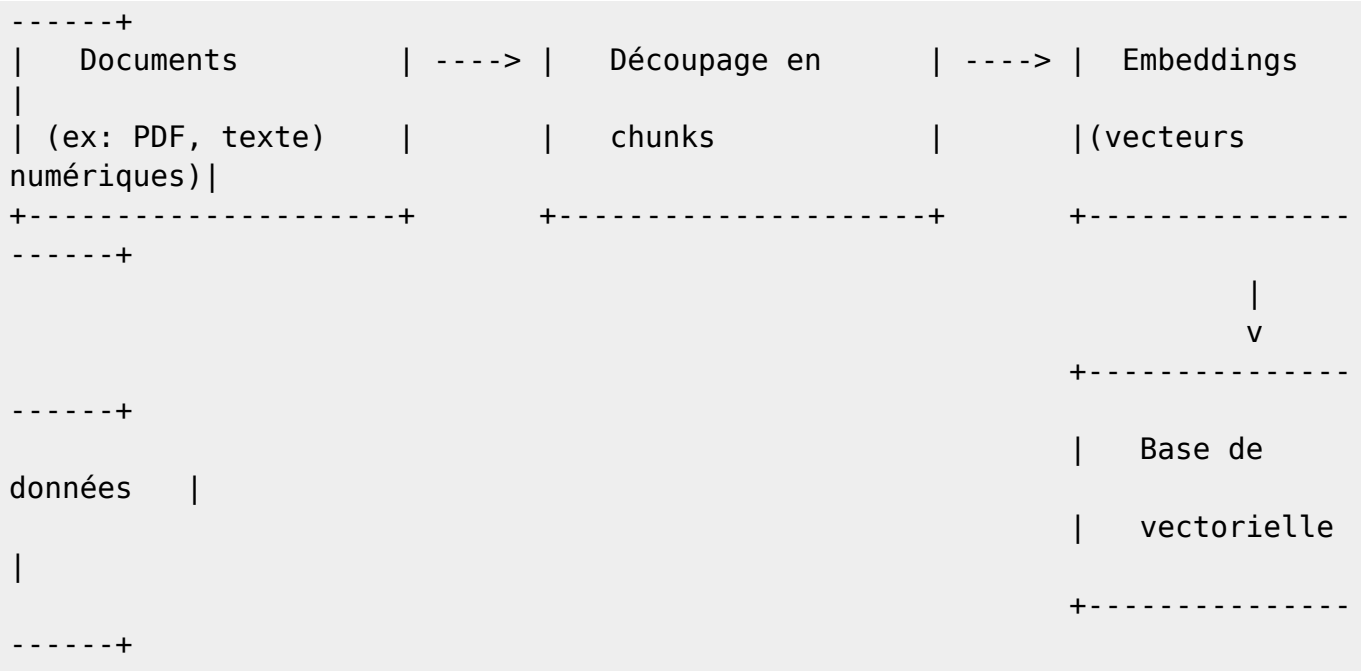
- Document : “Le RAG est une technique qui combine la récupération d'informations et la génération de texte.”
- Chunk : “Le RAG est une technique qui combine la récupération d'informations.”
- Embedding :

[0.12, 0.45, -0.23, 0.67, ...]

(vecteur numérique).

Schéma :





Étape 2 : Récupération des informations (Retrieval)

- **Objectif** : Trouver les chunks les plus pertinents pour répondre à une requête.
- **Processus** :
 1. La **requête de l'utilisateur** est transformée en un vecteur d'embedding.
 2. La base de données vectorielle **compare le vecteur de la requête** avec les vecteurs des chunks.
 3. Les chunks **les plus proches** (en termes de similarité cosinus) sont sélectionnés.

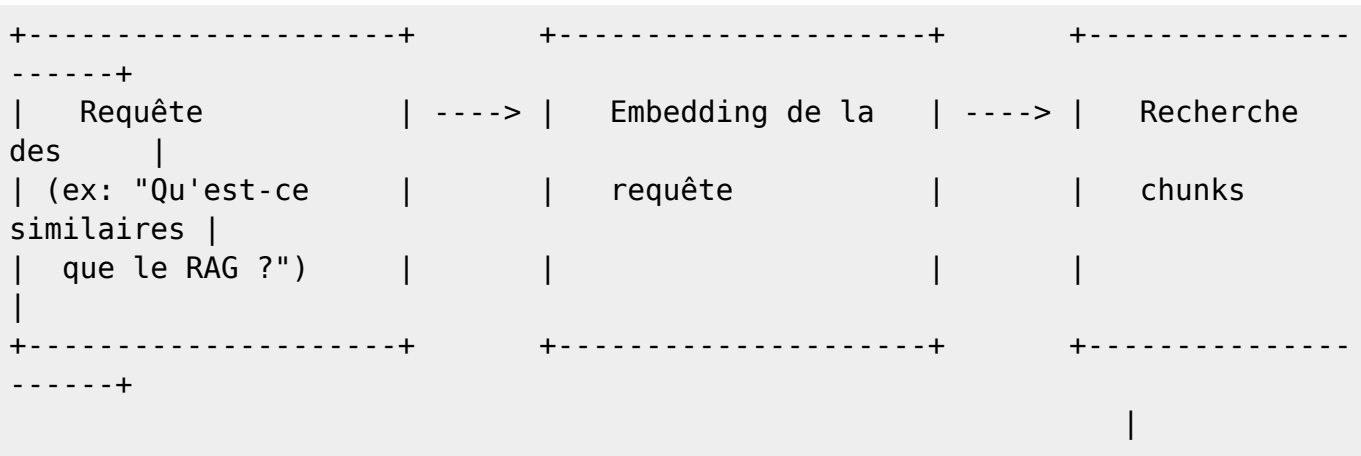
Exemple :

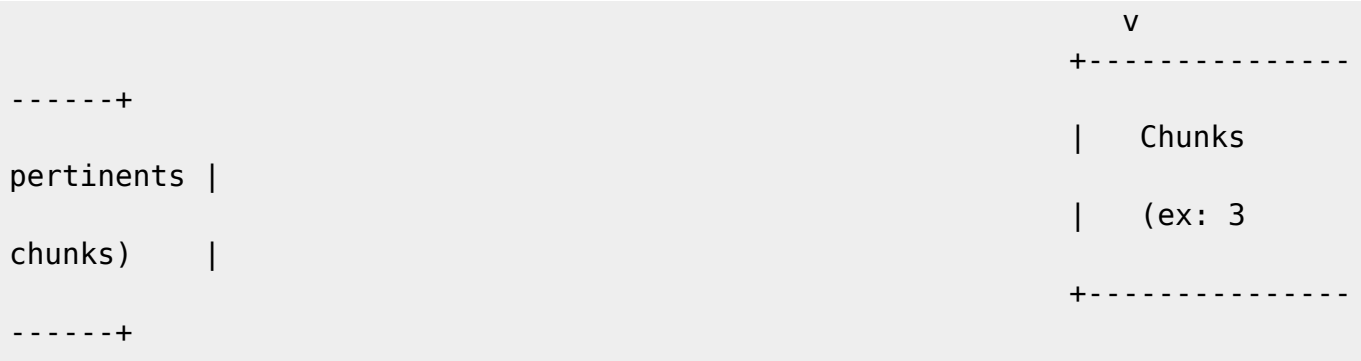
- Requête : “Qu'est-ce que le RAG ?”
- Embedding de la requête :

[0.15, 0.40, -0.25, 0.70, ...]

- Chunks récupérés : Les 3 chunks dont les embeddings sont les plus proches de celui de la requête.

Schéma :





Étape 3 : Génération de la réponse (Generation)

- **Objectif** : Générer une réponse précise en utilisant les chunks récupérés.
- **Processus** :
 1. Les chunks récupérés sont **ajoutés au prompt** du modèle de langage.
 2. Le modèle de langage (ex. : Llama, GPT) **génère une réponse** en se basant sur les chunks et la requête.

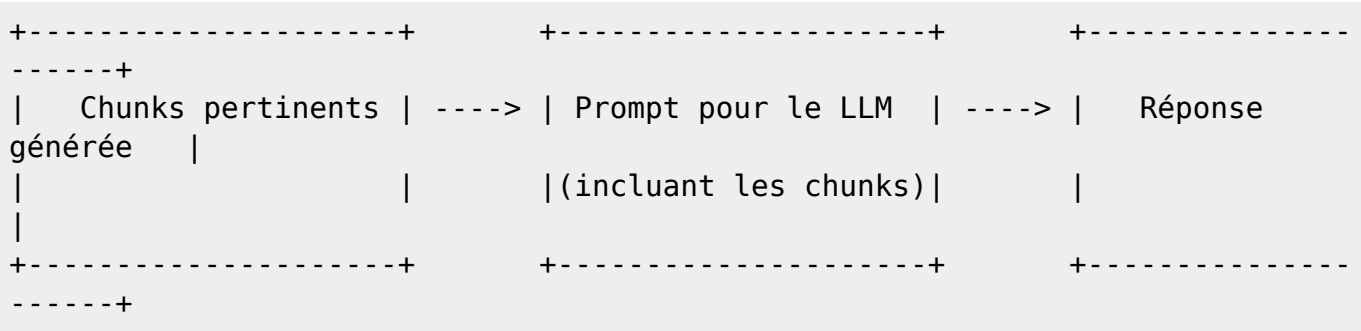
Exemple de prompt :

Réponds à la question en utilisant uniquement les informations suivantes :

Chunk 1 : "Le RAG est une technique qui combine la récupération d'informations..."
Chunk 2 : "Il utilise des embeddings pour représenter les documents..."
Chunk 3 : "Le RAG améliore la précision des modèles de langage..."

Question : Qu'est-ce que le RAG ?

Schéma :



5. Pourquoi les embeddings sont essentiels pour le RAG ?

Les embeddings permettent de :

1. **Transformer du texte en nombres** : Les modèles d'IA ne comprennent pas le texte brut, mais ils peuvent traiter des vecteurs numériques.
2. **Capturer le sens et les relations** : Les embeddings placent les mots et phrases similaires

proches les uns des autres dans l'espace vectoriel.

3. **Effectuer des recherches rapides** : Les bases de données vectorielles (comme FAISS) permettent de trouver rapidement les chunks les plus pertinents.

Exemple concret :

- **Requête** : "Comment fonctionne le RAG ?"
- **Embedding de la requête** :

```
[0.15, 0.40, -0.25, 0.70, ...]
```

- **Chunks similaires** :
 - "Le RAG combine la récupération d'informations et la génération de texte."

```
[0.12, 0.45, -0.23, 0.67, ...]
```

- "Les embeddings sont utilisés pour représenter les documents."

```
[0.14, 0.38, -0.27, 0.72, ...]
```

6. Exemple complet avec du code

Voici un exemple complet en Python pour illustrer le lien entre RAG et embeddings :

Étape 1 : Installation des bibliothèques

```
pip install sentence-transformers faiss-cpu langchain openai
```

Étape 2 : Indexation des documents

[indexation.py](#)

```
from sentence_transformers import SentenceTransformer
from langchain.vectorstores import FAISS
from langchain.text_splitter import CharacterTextSplitter
from langchain.document_loaders import TextLoader

# Charger les documents
loader = TextLoader("document.txt")
documents = loader.load()

# Découper les documents en chunks
text_splitter = CharacterTextSplitter(chunk_size=1000,
chunk_overlap=200)
docs = text_splitter.split_documents(documents)

# Générer les embeddings et créer la base vectorielle
```

```
model = SentenceTransformer('all-MiniLM-L6-v2')
embeddings = model.encode([doc.page_content for doc in docs])
vector_store = FAISS.from_documents(docs, embeddings)
```

Étape 3 : Récupération et génération

[recuperation-regeneration.py](#)

```
from langchain.llms import OpenAI
from langchain.chains import RetrievalQA

# Configurer le LLM
llm = OpenAI(temperature=0)

# Créer la chaîne RAG
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=vector_store.as_retriever()
)

# Poser une question
query = "Qu'est-ce que le RAG ?"
response = qa_chain.run(query)
print("Réponse :", response)
```

7. Exploiter RAG / embeddings dans Mozilla Firefox

Pré-requis : Ollama doit être actif sur votre machine. Si celui-ci n'est pas installé reportez vous aux articles :

- [Installation d'Ollama sous Ubuntu 25.04](#)
- [Installation d'Ollama sous Windows 11](#)

Pour vérifier si Ollama est un service actif :

```
systemctl status Ollama
```

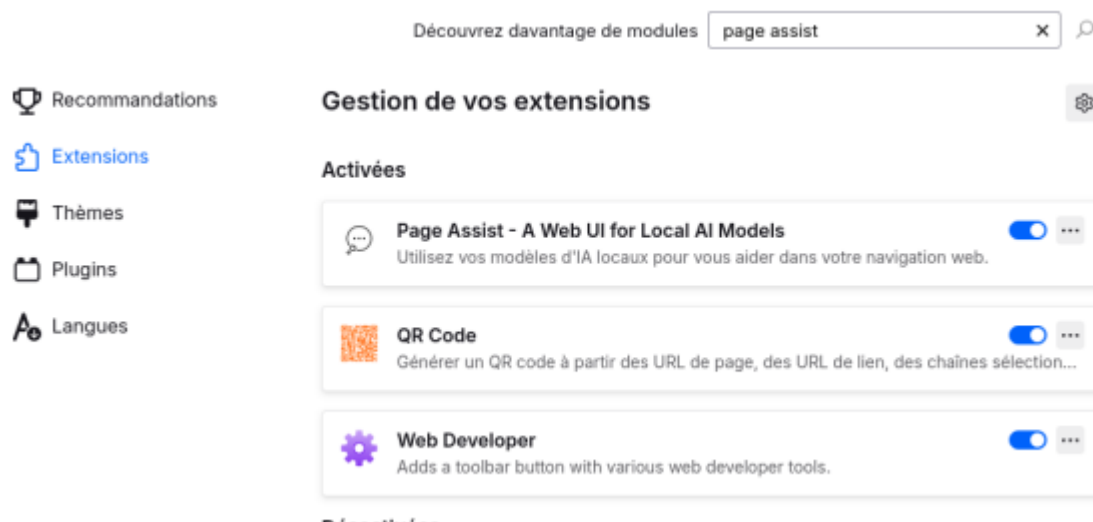
```

Terminal - utilisateur@MED-PC001: ~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
utilisateur@MED-PC001:~$ systemctl status ollama
● ollama.service - Ollama Service
   Loaded: loaded (/etc/systemd/system/ollama.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2025-10-10 11:17:33 CEST; 52min ago
     Main PID: 2701 (ollama)
        Tasks: 20 (limit: 9305)
       Memory: 2.26
          CPU: 38min 1.199s
      CGroup: /system.slice/ollama.service
              └─2701 /usr/local/bin/ollama serve

oct. 10 12:00:16 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:00:16 | 200 |      751.774µs |      127.0.0.1 | GET      "/api/tags"
oct. 10 12:03:09 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:09 | 200 |      778.024µs |      127.0.0.1 | GET      "/api/tags"
oct. 10 12:03:09 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:09 | 200 |      719.288µs |      127.0.0.1 | GET      "/api/tags"
oct. 10 12:03:12 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:12 | 200 |      623.324µs |      127.0.0.1 | GET      "/api/tags"
oct. 10 12:03:14 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:14 | 200 |      965.278µs |      127.0.0.1 | GET      "/api/tags"
oct. 10 12:03:17 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:17 | 200 |        82.184µs |      127.0.0.1 | GET      "/"
oct. 10 12:03:17 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:17 | 200 |    9.555272ms |      127.0.0.1 | GET      "/api/tags"
oct. 10 12:03:17 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:17 | 200 |      768.569µs |      127.0.0.1 | GET      "/api/tags"
oct. 10 12:03:21 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:21 | 200 |    1.149353ms |      127.0.0.1 | GET      "/api/tags"
oct. 10 12:03:21 MED-PC001 ollama[2701]: [GIN] 2025/10/10 - 12:03:21 | 200 |       16.686µs |      127.0.0.1 | GET      "/"
utilisateur@MED-PC001:~$

```

Installez dans Mozilla Firefox l'extension **Page Assist**



Intégrer un modèle multilingue en allant sur le site Ollama : [Modèles LLM - Embedding](#)

Choisir bge-m3

 Models GitHub Discord Docs Cloud

Cloud

Embedding

Vision

Tools

Thinking

Popular

embeddinggemma

EmbeddingGemma is a 300M parameter embedding model from Google.

embedding 300m

67.9K Pulls 5 Tags Updated 1 month ago

nomic-embed-text

A high-performing open embedding model with a large token context window.

embedding

42.5M Pulls 3 Tags Updated 1 year ago

mxbai-embed-large

State-of-the-art large embedding model from mixedbread.ai

embedding 335m

5M Pulls 4 Tags Updated 1 year ago

bge-m3

BGE-M3 is a new model from BAAI distinguished for its versatility in Multi-Functionality, Multi-Linguality, and Multi-Granularity.

embedding 567m

2.4M Pulls 3 Tags Updated 1 year ago

Puisque l'on utilise Page Assis, il nous suffit de cliquer sur le lien de téléchargement comme indiquer ci-dessous :

bge-m3

2.4M Downloads Updated 1 year ago

BGE-M3 is a new model from BAAI distinguished for its versatility in Multi-Functionality, Multi-Linguality, and Multi-Granularity.

embedding 567m

Models

View all →

Name	Size	Context	Input
bge-m3:latest	1.2GB	8K	Text
bge-m3:567m latest	1.2GB	8K	Text

8. Avantages du RAG avec embeddings

Avantage	Description
Précision	Les réponses sont basées sur des informations pertinentes et à jour.
Flexibilité	Les documents peuvent être mis à jour sans réentraîner le modèle de langage.
Transparence	Les sources des réponses peuvent être tracées et vérifiées.
Efficacité	La recherche vectorielle est rapide et scalable.

9. Cas d'Utilisation du RAG

9.1. Chatbots spécialisés

- **Exemple** : Un chatbot pour le support client qui utilise une base de connaissances interne.
- **Avantage** : Réponses précises et basées sur les documents de l'entreprise.

9.2. Recherche d'informations

- **Exemple** : Un moteur de recherche interne pour les employés.
- **Avantage** : Permet de trouver des informations spécifiques dans une grande base de documents.

9.3. Génération de contenu

- **Exemple** : Génération de rapports ou de résumés à partir de documents techniques.
- **Avantage** : Contenu généré basé sur des sources fiables.

10. Outils et bibliothèques pour le RAG avec Embeddings

Outil/Bibliothèque	Description	Lien
LangChain	Framework pour développer des applications RAG.	LangChain
FAISS	Bibliothèque pour la recherche de similarité dans des bases de données vectorielles.	FAISS
Chroma	Base de données vectorielle open-source.	Chroma
Weaviate	Base de données vectorielle avec des fonctionnalités avancées.	Weaviate
Sentence Transformers	Modèles pré-entraînés pour générer des embeddings de phrases.	Sentence Transformers
Hugging Face	Plateforme pour les modèles de langage et les embeddings.	Hugging Face

□ Points clés à retenir

Les **embeddings** sont au cœur du **RAG** :

- Ils **transforment le texte en vecteurs numériques** que les modèles d'IA peuvent comprendre.
- Ils permettent de **trouver les informations pertinentes** dans une base de données.
- Ils **améliorent la précision et la pertinence** des réponses générées par les LLM.

📄 Sources

- [Pinecone – Guide RAG](#)
- [Sentence Transformers – Embeddings](#)
- [GitHub – FAISS](#)
- [GitHub – Chroma](#)
- [Weaviate – Base vectorielle](#)
- [Ollama – Modèles d’embeddings](#)

- 📄 Fondamentaux et intégration de l'IA
- Comment contrôler le comportement de son LLM ?
- Comment marche l'IA générative ?
- Configuration matérielle et LLM
- Heretic – Suppression automatique de la censure des modèles de langage basés sur les transformers
- Histoire de l'IA et des modèles de langage (LLM)
- Installation d'Ollama sous Ubuntu 25.04
- Installation d'Ollama sous Windows 11
- Interroger une base documentaire locale avec Ollama en environnement professionnel
- Les fondamentaux de l'IA et des LLM
- RAG (Retrieval-Augmented Generation)
- RAG et embeddings : comprendre et implémenter
- Séquence 11 : Installation d'Open WebUI en conteneur sur Ubuntu 25.04 avec Ollama
- Statistiques, probabilités et modèles bayésiens
- Tokens, modèles de langage et taille des données
- Utilisation basique d'Ollama

From:

<https://wiki.bozocinformatique.fr/> - **Hack the knowledge, secure the future**

Permanent link:

<https://wiki.bozocinformatique.fr/ia:mozilla-ollama-rag>

Last update: **2025/12/04 10:13**

