

INTRODUCTION À L'IA

PRÉSENTATION DE CERTAINS ÉLÉMENTS THÉORIQUES NÉCESSAIRES À LA RÉALISATION DES ACTIVITÉS

SOMMAIRE

A. Quelques éléments théoriques.....	2
A.1. Objectif.....	2
A.2. RAG ?.....	2
A.3. Démarche générale.....	2
A.4. RAG vs Hallucinations.....	3
A.5. Quelles applications concrètes ?.....	3
B. Apprendre / Stocker - Chercher / Répondre.....	4
B.1. Comment l'IA apprend ?.....	4
B.1.1 Préparer les documents.....	4
B.1.2 Des documents aux embeddings (encodage vectoriel).....	4
B.1.3 Stockage des embeddings.....	5
B.2. Comment la BDD retrouve l'information pertinente ?.....	5
B.3. Comment l'IA répond ?.....	6

A. QUELQUES ÉLÉMENTS THÉORIQUES

A.1. OBJECTIF

Ce labo a pour objectif de découvrir, au travers de scripts Python, les différentes étapes qui permettent de construire un système d'Intelligence Artificielle particulier appelé RAG (*Retrieval Augmented Generation*). On pourra alors disposer d'une IA qui connaît une base de documents et que l'on peut interroger pour retrouver des informations importantes, synthétiser ou demander des explications.

A.2. RAG ?

Le RAG (ou Génération Augmentée par Récupération) est une technique qui combine deux IA :

- Une IA qui va chercher l'information (*Retrieval*)
- Une IA qui génère du texte (*Generation*) pour la réponse

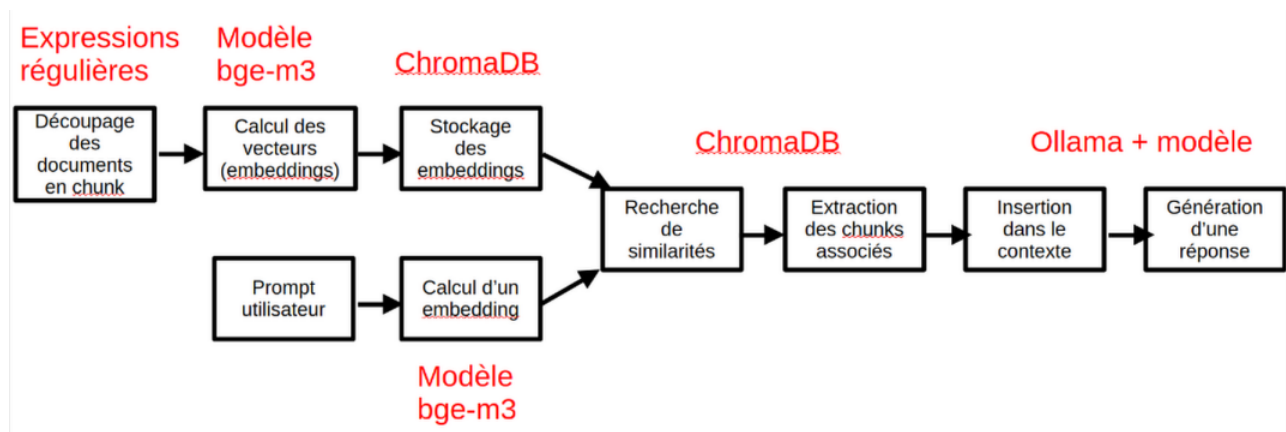
L'idée est simple : au lieu de laisser une IA inventer ou halluciner, on lui fournit les bons documents avant qu'elle réponde.

Dans notre système RAG :

1. Le **modèle d'embedding** (BGE-M3 dans l'exemple), en s'appuyant sur un SGBD (Chroma), permet de trouver les bons documents.
2. Le **modèle de langage** (Ollama3.1) analyse ces documents et génère la réponse.

A.3. DÉMARCHE GÉNÉRALE

Le synoptique de la démarche de notre labo est le suivant (chaque point sera détaillé par la suite) :



1. Découpage des documents de référence en fragments (*chunks*) : phase préparatoire « manuelle » dans laquelle on prépare les informations pour aider le modèle d'IA à mieux apprendre.
2. Ces fragments sont passés au modèle d'embedding qui les analyse et en réalise une représentation vectorielle.
3. Les fragments avec les vecteurs calculés ainsi que des métadonnées sont stockés par le SGBD vectoriel.
4. En parallèle, pour que le SGBD soit en mesure de retrouver les fragments cohérents, le modèle d'embedding vectorise la requête (*prompt*) de l'utilisateur.
5. Le SGBD effectue une recherche de similarité entre le vecteur du prompt et la base de données de vecteurs.
6. Il récupère les fragments de texte liés aux vecteurs trouvés.
7. Le prompt et les fragments sont insérés dans un contexte (qui a pour but de préciser au mieux la réponse attendue).
8. L'ensemble est soumis au modèle de langage qui produit une réponse.

A.4. RAG VS HALLUCINATIONS

L'objectif d'un système RAG est d'éviter le phénomène d'hallucination qu'une IA générative peut produire dans certains contextes. Précisons cette notion d'hallucination :

En effet, un modèle de langage fonctionne comme un « prédicteur de mots » :

1. il calcule la probabilité de chaque mot possible ;
2. il choisit celui qui semble le plus plausible ;
3. puis il recommence pour le mot suivant.

Ce mécanisme est purement statistique. Donc si l'IA manque d'informations ou si les données d'entraînement sont ambiguës/incomplètes, elle peut :

- deviner ;
- combiner des morceaux de connaissances ;
- générer une réponse plausible mais incorrecte.

Pourquoi l'IA ne dit pas « je ne sais pas » ? Parce que son objectif n'est pas de vérifier la vérité, mais de continuer la phrase de la manière la plus probable et donc :

- « Je ne sais pas » n'est pas toujours la suite la plus probable ;
- une réponse inventée mais cohérente peut avoir une probabilité plus élevée.

Donc elle hallucine...

A.5. QUELLES APPLICATIONS CONCRÈTES ?

Cette absence d'hallucination dans un système RAG permet d'envisager un certain nombre d'applications spécifiques à une organisation. Voici quelques exemples :

1. Assistant interne (c'est l'objectif de ce labo) :

- Recherche documentaire intelligente sur de grands volumes : procédures, contrats, normes, guides techniques ;
- Résumés contextualisés basés sur les passages réellement trouvés dans les sources ;
- Très utile dans les secteurs réglementés (banque, assurance, santé, juridique, etc.).

2. Support client automatisé :

- *Chatbots* capables de répondre avec précision à partir de FAQ, bases de connaissances, historiques de tickets ;
- Réponses cohérentes, à jour, et adaptées au contexte du client.

3. Aide à la décision et analyse métier :

- Génération de synthèses à partir de données internes (rapports, KPI, notes de réunion, etc.) ;
- Explications contextualisées pour les équipes opérationnelles ;
- Permet d'ancrer l'IA dans la réalité métier plutôt que dans des connaissances génériques.

4. Formation :

- Assistants pédagogiques personnalisés qui s'appuient sur les documents internes.
- Réponses adaptées au niveau de l'apprenant et aux contenus de l'entreprise.

B. APPRENDRE / STOCKER – CHERCHER / RÉPONDRE

B.1. COMMENT L'IA APPREND ?

B.1.1 Préparer les documents

Pour obtenir les meilleurs résultats, il faut aider l'IA à assimiler les documents de référence en nettoyant les éléments qui ne portent pas de sens et en mettant en avant la structure et les parties importantes. L'objectif est de produire des fragments de texte (*chunks*).

Par exemple :

- pour un texte de loi au format HTML : on retirera les balises, on isolera chaque article dans un fragment auquel on associera des métadonnées (numéro de l'article, titre, etc.) ;
- pour un diaporama PDF : retirer la mise en page, les données répétitives dans les en-têtes et pieds de page (auteur, titre du document, numéros de page, date, etc.), isoler chaque diapo dans un fragment.

B.1.2 Des documents aux embeddings (encodage vectoriel)

Les LLM (*Large Learning Models*) comme GPT, Llama, Claude, Mistral, etc. reposent entièrement sur des **embeddings** (ou intégration) internes pour représenter :

- les mots et les phrases ;
- les images ;
- les sons ;
- etc.

C'est leur manière de transformer le monde réel en **vecteurs mathématiques** qu'ils peuvent manipuler. Sans embeddings, un LLM ne peut pas fonctionner.



Rappel : notion de vecteur en programmation

Un vecteur est simplement une liste de nombres. Par exemple, en Python :

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1.,  4.,  5.,  8.])
>>> type(a)
<type 'numpy.ndarray'>
```

On peut donc définir un **embedding** comme un vecteur particulier, généré par un modèle, et qui encode :

- du sens ;
- des relations ;
- des similarités ;
- des caractéristiques abstraites.

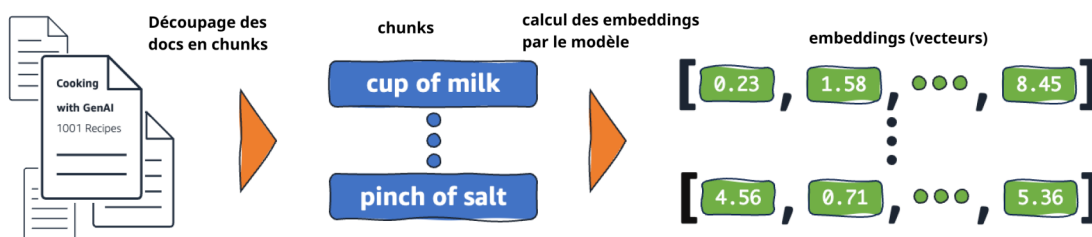
De même, on peut définir un **modèle d'embedding** comme un réseau neuronal qui apprend à représenter des données (analyse sémantique) sous forme de vecteurs qui capturent leur sens et leurs relations.

Pourquoi transformer des informations en vecteurs ?

Parce que les algorithmes d'IA ne comprennent pas directement le langage humain ou les images. Ils ne savent traiter que des nombres. Un embedding sert donc de traduction : il encode une information complexe dans un espace mathématique où :

- des éléments de sens similaires sont « proches » ;
- des éléments de sens différents sont « éloignés » ;
- certaines relations deviennent géométriques (ex. : roi – homme + femme \approx reine).

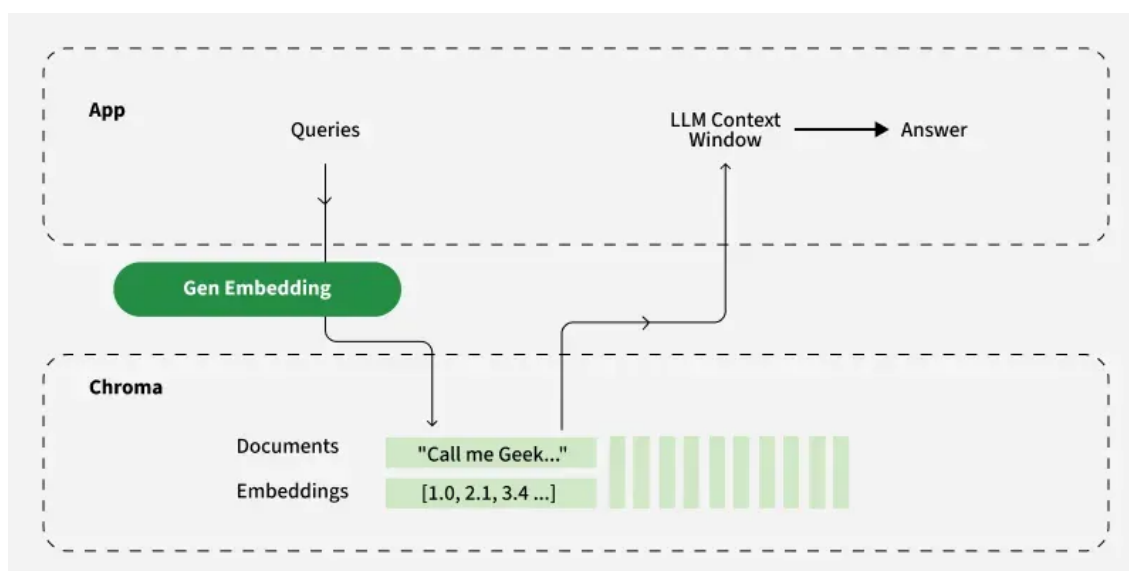
En résumé, le processus est le suivant :



B.1.3 Stockage des embeddings

Lorsque les vecteurs sont créés, il faut les enregistrer dans une base de données. Ces BDD ne sont pas relationnelles (SQL) mais **vectorielles**.

Dans l'exemple, le SGBD libre Chroma est utilisé pour stocker les fragments de documents et les vecteurs associés. L'objectif sera d'utiliser les vecteurs pour effectuer des **recherches par similarité**. Les documents liés seront transmis au modèle de langage pour la réponse :

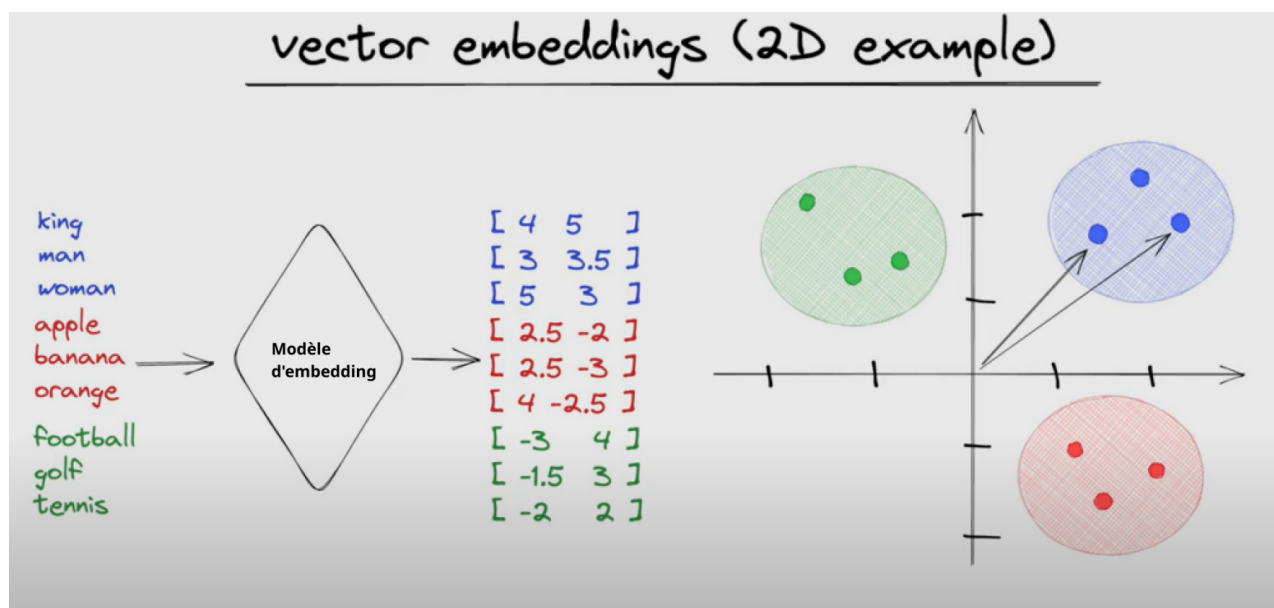


B.2. COMMENT LA BDD RETROUVE L'INFORMATION PERTINENTE ?

En premier lieu, le prompt est lui-même vectorisé (par la même IA que celle qui a calculé les embeddings des documents de référence).

La **recherche de similarités** est l'équivalent d'une « requête » qui permet de trouver des éléments proches les uns des autres - non pas par leurs mots exacts (ou synonymes), mais par leur sens. Concrètement, il s'agit d'un calcul mathématique de distance entre vecteurs (le vecteur du prompt / les vecteurs en BDD).

Exemple en 2 dimensions :



i Simple exemple puisque dans la réalité les vecteurs ont plusieurs centaines voire milliers de dimensions !

B.3. COMMENT L'IA RÉPOND ?

Lorsque la recherche de similarités a été réalisée (= on a retrouvé un certain nombre de fragments en lien avec le prompt utilisateur), un **modèle de langage** est utilisé pour formuler une réponse en langage naturel.

Le modèle de langage fait 3 choses :

- Il lit le contexte : il analyse les passages fournis comme s'il lisait un document.
- Il identifie les éléments pertinents : il repère les phrases, instructions ou définitions qui répondent à la question.
- Il génère une réponse : il reformule, synthétise, combine et complète (sans inventer si on lui impose de ne pas halluciner).

⚠ De l'importance du prompt !

Pour que la réponse soit la plus fiable possible, il faut donner au modèle de langage un maximum de précisions :

1. Un contexte pertinent. Les fragments doivent être :

- suffisamment longs pour être utiles ;
- suffisamment courts pour ne pas noyer le modèle ;
- réellement liés à la question.

2. Une consigne claire. Par exemple :

- « Réponds en tant que analyste juridique » ;
- « Réponds uniquement à partir du contexte fourni. » ;
- « Si l'information n'est pas dans le contexte, dis que tu ne sais pas. » ;

3. Un format structuré. Un prompt bien organisé aide énormément :

- Question
- Contexte
- Instructions
- Format de réponse attendu

4. Un modèle suffisamment grand. Plus le modèle est puissant, plus il :

- comprend le contexte ;
- évite les hallucinations ;
- synthétise correctement ;
- (mais plus la réponse prend du temps et demande de ressources CPU/GPU/RAM...)