HEIG VD
HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Système de recommandation de livres personnalisé basé sur LLMs

Rapport final

Département TIC

Filière Informatique et systèmes de communication

Orientation Informatique logicielle

Rachel Tranchida

16 septembre 2025

Supervisé par :
Prof. F. Nastaran (HEIG-VD)

Hes·so
Haute Ecole Spécialisée de Suisse occidentale

Résumé

Ce projet porte sur la conception et l'implémentation d'un système de recommandation de livres personnalisé. Il utilise les grands modèles de langage (LLM). L'objectif est une analyse sémantique approfondie des oeuvres littéraires. Ce système vise à dépasser les limites des recommandations conventionnelles. Ces dernières se basent souvent sur des métadonnées ou le comportement des utilisateurs. Notre approche offre des suggestions plus pertinentes et nuancées. Elles sont ancrées dans les thèmes, le style et la narration des textes. Pour cela, le système utilise le texte complet des livres.

Le projet se décompose en deux solutions principales :

1. **La pipeline de téléchargement et de traitement des livres** : Une pipeline de traitement a été développée pour constituer et analyser un corpus de plus de 5000 oeuvres issues du Projet Gutenberg. Cette solution est responsable du nettoyage des textes bruts et de l'utilisation de l'API Gemini pour générer des résumés abstractifs et des listes de thèmes pour chaque livre. Ces données sémantiques sont ensuite transformées en représentations vectorielles (embeddings) pour faciliter les comparaisons de similarité.
2. **L'application de recommandation** : Il s'agit d'une application full-stack moderne, conçue sur une architecture de microservices conteneurisée avec Docker. Elle intègre un backend développé en Python avec FastAPI, un frontend en Vue.js, et une base de données PostgreSQL 17 optimisée avec TimescaleDB, pgvector et pgai pour fonctionner comme une base de données vectorielle. La génération des embeddings est automatisée par un worker découplé qui communique avec un service Ollama local. Le système final permet aux utilisateurs de recevoir des recommandations basées sur la similarité avec un livre spécifique, des requêtes en langage naturel, ou l'analyse de leur historique de lecture. De plus, l'application offre la possibilité de demander une explication de la similarité entre deux livres à un LLM local.

Une évaluation qualitative a confirmé la capacité de ce prototype à générer des suggestions sémantiquement cohérentes et pertinentes, démontrant ainsi le fort potentiel des LLM pour enrichir et personnaliser l'expérience de découverte littéraire.

Préambule

Ce travail de Bachelor (ci-après **TB**) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'École.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD
Le Chef du Département

Yverdon-les-Bains, le 16 septembre 2025

Authentification

Je soussignée, Rachel Tranchida, atteste par la présente avoir réalisé seule ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Rachel Tranchida

Yverdon-les-Bains, le 23 juillet 2025

Table des matières

Table des figures

Liste des tableaux

Liste des codes sources

Chapitre 1

Introduction

1.1 Contexte et motivation

Les systèmes de recommandation de livres traditionnels ont longtemps reposé sur des approches basées sur les métadonnées (comme le genre, l'auteur, l'année de publication) ou le filtrage collaboratif, qui s'appuie sur le comportement d'autres utilisateurs (leurs lectures, leurs évaluations). Bien que pertinentes, ces méthodes atteignent souvent leurs limites lorsqu'il s'agit de capturer les nuances sémantiques, stylistiques et thématiques profondes qui résident *à l'intérieur* des oeuvres elles-mêmes. Elles peuvent peiner à identifier des similarités conceptuelles subtiles ou à expliquer pourquoi un livre spécifique serait pertinent pour un utilisateur au-delà de simples corrélations statistiques ou de correspondances de mots-clés.

L'émergence et les progrès rapides des *Large Language Models* (LLMs) offrent désormais des capacités d'analyse textuelle sans précédent. Ces modèles sont capables de comprendre, de synthétiser et d'extraire des informations complexes de manière beaucoup plus sophistiquée que les approches antérieures. Cette avancée ouvre une voie prometteuse pour le traitement de contenus textuels longs et structurés comme les livres.

Ce projet est motivé par la volonté d'explorer et d'exploiter pleinement le potentiel des LLMs pour la recommandation de livres. Le projet a pour motivation d'aller au-delà des métadonnées et utiliser le texte intégral des oeuvres, afin de développer un système capable de fournir des recommandations plus personnalisées, pertinentes et enrichies par une compréhension sémantique profonde. L'objectif est de créer un prototype capable d'identifier des affinités non évidentes au premier abord, basées sur les thèmes, le ton ou le style narratif, et de justifier ces suggestions de manière transparente pour l'utilisateur.

1.2 Objectifs

L'objectif central de ce travail de bachelor est de concevoir, développer et évaluer un prototype de système de recommandation de livres personnalisé qui exploite la puissance des LLMs pour analyser le contenu sémantique intégral des oeuvres littéraires. Pour y parvenir, les objectifs spécifiques sont les suivants :

- Constituer et prétraiter un corpus de livres numériques adapté à l'analyse par LLMs.
- Développer et appliquer des techniques d'analyse sémantique pour extraire les thèmes, la tonalité et les éléments stylistiques clés de chaque texte complet.
- Générer des représentations vectorielles (embeddings) denses et informatives capturant l'essence sémantique de chaque livre.
- Implémenter un moteur de recommandation qui calcule la similarité sémantique entre ces embeddings, personnalise les suggestions en fonction de l'historique et des préférences de l'utilisateur, et génère des explications claires pour chaque recommandation.

Chapitre 2

Cahier des charges

Introduction

Ce projet vise à créer un système de recommandation de livres personnalisé exploitant des modèles de langage (LLMs) pour suggérer des livres selon les préférences de genres et thèmes des utilisateurs, et en prenant en compte les livres lus par l'utilisateur.

Contexte et motivation

Les systèmes de recommandation traditionnels se limitent souvent aux métadonnées des livres ou aux évaluations d'autres utilisateurs. L'émergence des LLMs offre une opportunité unique d'analyser le contenu sémantique des oeuvres littéraires complètes pour proposer des recommandations plus nuancées et personnalisées. Ce projet explore cette approche en utilisant un corpus open-source de livres complets comme base de données principale.

Objectifs principaux

- Extraction et prétraitement du contenu textuel de livres pour l'analyse sémantique
- Développement de méthodes d'analyse sémantique de textes littéraires (identification de thèmes, tonalité, style narratif)
- Création de représentations vectorielles (embeddings) efficaces capturant l'essence des oeuvres
- Conception et optimisation d'un moteur de recommandation basé sur le contenu sémantique
- Développement d'une interface utilisateur minimale pour tester le système

Caractéristiques distinctives du projet

- L'utilisation du texte intégral des oeuvres plutôt que de simples métadonnées
- L'analyse sémantique profonde du contenu littéraire via des modèles de langage avancés
- La capacité à identifier des similarités conceptuelles et thématiques subtiles entre les textes
- La génération d'explications personnalisées justifiant chaque recommandation

Description Générale

Perspective du Produit

Le projet produira un prototype de moteur de recommandation s'appuyant sur les LLMs pour l'analyse littéraire et la recommandation basée sur le contenu sémantique.

Fonctions Principales du Produit

- **Traitement Sémantique des Livres :**
  - Extraction et prétraitement du contenu textuel

  - Analyse sémantique et génération de résumés via LLMs
  - Création d'embeddings capturant les thèmes et caractéristiques stylistiques
- **Moteur de Recommandation :**
  - Calcul de similarité contextuelle entre livres basé sur leurs embeddings
  - Personnalisation des suggestions selon l'historique et les préférences utilisateur
  - Algorithmes d'équilibre entre similarité et diversité des recommandations
  - Génération d'explications interprétables pour chaque recommandation
- **Interface Utilisateur :**
  - Saisie des préférences et de l'historique de lecture
  - Visualisation des recommandations avec leurs explications
  - Recherche basique dans le catalogue de livres

Choix technologiques

- **Langages et frameworks :** Python avec FastAPI ou Flask pour le backend
- **Modèles LLM :** Par exemple, Gemini, Claude ou OpenAI API ou modèles open-source déployés localement (Llama, Mistral)
- **Base de Données :** Par exemple PostgreSQL
- **Frontend :** Par exemple Vue.js

Exigences Spécifiques

Exigences Fonctionnelles

Traitement et analyse des oeuvres littéraires

- Le système doit traiter efficacement des textes intégraux de romans et autres oeuvres littéraires
- Le système doit implémenter des techniques de chunking et de summarization pour gérer les textes très longs
- Le système doit extraire automatiquement les thèmes principaux, le ton narratif et les éléments stylistiques clés
- Le système doit générer des embeddings qui capturent l'essence sémantique des oeuvres complètes

Moteur de Recommandation

- Le système doit générer des recommandations personnalisées basées sur l'historique de lecture
- Le système doit calculer la similarité sémantique entre oeuvres en utilisant des embeddings générés par LLM
- Le système doit générer une explication textuelle pour chaque recommandation en s'appuyant sur l'analyse des similarités réalisée
- Le système doit pouvoir traiter des requêtes textuelles libres décrivant les préférences de lecture (si le temps le permet)

Interface Utilisateur

- Le système doit permettre aux utilisateurs de sélectionner des genres d'intérêt et d'indiquer les livres déjà lus
- Le système doit afficher une liste claire de livres recommandés avec leurs explications
- Le système doit permettre une recherche basique par titre, auteur ou mots-clés
- Le système doit présenter aux utilisateurs des informations synthétiques sur les livres recommandés

Exigences Non Fonctionnelles

Performance

- Les recommandations doivent être générées dans un délai raisonnable (moins de 10 secondes)
- Le système doit pouvoir traiter un corpus d'au moins 1000 livres dans la phase de prototype

Qualité des recommandations

- Les recommandations doivent démontrer une compréhension significative des thèmes et du style des livres
- Les explications générées doivent être précises, pertinentes et compréhensibles

Livrables du Projet

- Pipeline de traitement de texte et de génération d'embeddings pour analyse littéraire
- Prototype fonctionnel du moteur de recommandation avec documentation du code
- Interface utilisateur minimaliste permettant de tester le système
- Rapport détaillé sur les méthodes utilisées, l'architecture, et les résultats obtenus
- Jeu de données structuré avec les embeddings et analyses générées
- Présentation démonstratives des capacités du système

Calendrier estimatif des tâches

Phase 1 : Exploration et préparation (8-10 jours)

- Recherche sur l'état de l'art et sélection des approches (2 jours)
- Identification et acquisition du corpus de livres open-source (1-2 jours)
- Exploration des API LLM et tests préliminaires (2-3 jours)
- Configuration de l'environnement de développement (1 jour)
- Conception détaillée de l'architecture du système (2 jours)

Phase 2 : Traitement des données et génération des embeddings (12-15 jours)

- Extraction et prétraitement du texte des livres (2-3 jours)
- Développement des techniques de chunking et summarization (3-4 jours)
- Expérimentation avec différentes stratégies d'embedding (3-4 jours)
- Génération et stockage des embeddings finaux (2-3 jours)
- Tests et optimisation du pipeline de traitement (2 jours)

Phase 3 : Développement du moteur de recommandation (14-17 jours)

- Implémentation des algorithmes de similarité sémantique (3-4 jours)
- Développement des stratégies de personnalisation (3-4 jours)
- Création du module de génération d'explications (3-4 jours)
- Optimisation des performances du moteur (2-3 jours)
- Tests approfondis des recommandations (3 jours)

Phase 4 : Développement de l'Interface utilisateur (5-7 jours)

- Conception d'une interface minimaliste (1-2 jours)
- Implémentation de l'interface de saisie des préférences (1-2 jours)
- Développement de l'affichage des recommandations (1-2 jours)
- Intégration avec le moteur de recommandation (1-2 jours)

Phase 5 : Évaluation, documentation et finalisation (7-9 jours)

- Évaluation systématique de la qualité des recommandations (2-3 jours)
- Documentation complète du code et des processus (2-3 jours)
- Rédaction du rapport final (2-3 jours)
- Préparation de la présentation et des démonstrations (1-2 jours)

Total estimé : 46 - 58 jours de travail

Chapitre 3

Etat de l'art

3.1 Les systèmes de recommandation

Les systèmes de recommandation désignent une classe de systèmes algorithmiques conçus pour filtrer l'information et fournir aux utilisateurs des suggestions personnalisées d'items. Leur rôle principal est d'estimer les préférences des utilisateurs afin de leur présenter une sélection pertinente, facilitant ainsi la découverte et la prise de décision, face à la grande quantité d'informations disponibles [53]. Ces systèmes sont devenus des éléments essentiels dans de nombreux domaines applicatifs, comme le commerce en ligne (p. ex. Amazon) ou encore la consommation de médias (p. ex. Netflix)

3.1.1 Filtrage collaboratif

Le filtrage collaboratif (FC) déduit les préférences des utilisateurs en identifiant des similitudes de comportement au sein d'un groupe [55]. L'idée principale est que des utilisateurs ayant eu des interactions similaires par le passé auront tendance à avoir des goûts similaires à l'avenir. Le FC comprend principalement des méthodes basées sur la mémoire et celles basées sur un modèle.

Approches Basées sur la Mémoire (Memory-Based)

Les systèmes *basés sur la mémoire*, aussi appelés *basés sur les voisins* (neighbor-based), sont une extension des algorithmes de type *k-plus proches voisins* (k-NN). Ils prédisent l'interaction d'un utilisateur cible avec un item en se basant sur le comportement d'utilisateurs similaires ou sur les interactions avec des items similaires, sans créer de modèle général explicite [38].

Filtrage basé sur l'utilisateur (User-Based)
Cette méthode identifie des utilisateurs ayant des historiques d'interaction similaires à celui de l'utilisateur cible. La similarité entre utilisateurs, calculée à partir de leurs vecteurs d'interactions (les lignes de la matrice utilisateur-item M), permet de pondérer les évaluations des "voisins" pour prédire celles de l'utilisateur cible sur de nouveaux items.

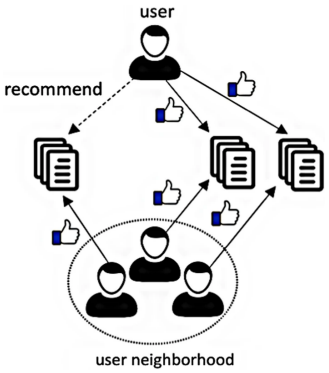


Figure 3.1 – *User-based collaborative filtering*
Source : Roy et al. (2022, p. 4)

Filtrage basé sur l'élément (Item-Based)
Ici, la similarité est calculée entre les items (les colonnes de la matrice M), en fonction de la manière dont l'ensemble des utilisateurs interagit avec eux. Des items sont considérés comme similaires si un grand nombre d'utilisateurs ont interagi avec eux de façon comparable. Les recommandations pour un utilisateur cible se basent alors sur les items similaires à ceux qu'il a déjà appréciés.

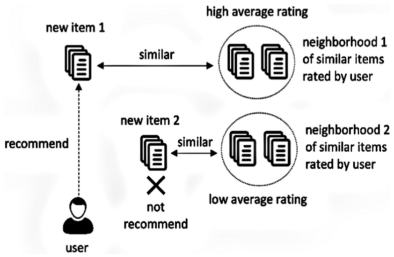


Figure 3.2 – *Item-based collaborative filtering*
Source : Roy et al. (2022, p. 5)

Approches basées sur un modèle (Model-Based)

Contrairement aux méthodes basées sur la mémoire, les approches *basées sur un modèle* construisent un modèle prédictif général à partir de la matrice des interactions utilisateur-item. Ce modèle, développé à l'aide de techniques d'apprentissage automatique (comme les arbres de décision, les réseaux neuronaux ou la factorisation matricielle), a pour but d'estimer les interactions manquantes dans la matrice.

Factorisation Matricielle (Matrix Factorization)
La factorisation matricielle est un modèle de représentation vectorielle continue (parfois appelée *embedding*) simple. Étant donné la matrice d'interactions (ou de retours utilisateurs) $A \in \mathbb{R}^{m \times n}$, où m est le nombre d'utilisateurs (ou de requêtes) et n correspond au nombre d'items, le modèle apprend :

- Une matrice de représentation vectorielle continue des utilisateurs $U \in \mathbb{R}^{m \times d}$, où la i-ème ligne, notée U_i, est la représentation vectorielle continue (ou *embedding*) de l'utilisateur i.
- Une matrice de représentation vectorielle continue des items $V \in \mathbb{R}^{n \times d}$, où la j-ème ligne, notée V_j, est la représentation vectorielle continue (ou *embedding*) de l'item j.

La dimension d correspond au nombre de facteurs latents.

Les représentations vectorielles continues U_i et V_j sont apprises de telle sorte que le produit matriciel UV^T soit une bonne approximation de la matrice d'interactions A. Notez que l'entrée (i, j) de la matrice UV^T est simplement le produit scalaire $\langle U_i, V_j \rangle$ des représentations vectorielles de l'utilisateur i et de l'item j. Cette valeur prédite vise à être proche de l'interaction observée $A_{i,j}$. [17]

Figure 3.3 – *Exemple de factorisation matricielle*
Source : Google for Developers, 2024

3.1.2 Recommandation basée sur le contenu

La recommandation basée sur le contenu (Content-Based, CB) se caractérise principalement par l'utilisation directe des descriptions des items pour formuler des suggestions personnalisées [36]. Au lieu de se fier aux interactions d'une communauté d'utilisateurs, cette approche analyse les caractéristiques propres des items qu'un utilisateur a aimés par le passé, afin de lui proposer des items qui leur ressemblent fortement.

Dans le domaine littéraire, ces caractéristiques peuvent être variées et détaillées : genre, thèmes abordés, structure du récit, style d'écriture, mots-clés issus du texte, ou même l'analyse du texte intégral. Un profil utilisateur, qui représente ses préférences pour ces différentes caractéristiques, est construit au fil du temps. Les recommandations sont ensuite générées en comparant les nouveaux items à ce profil.

Les systèmes basés sur le contenu ont l'avantage de bien fonctionner pour recommander des items nouveaux ou peu populaires, ce qui atténue le problème dit du "démarrage à froid" pour les items cf. 3.1.6, à condition que leurs descriptions soient disponibles. En effet, les systèmes de recommandation basés sur le contenu analysent les caractéristiques des items et les mettent en relation avec les préférences des utilisateurs. Ils permettent ainsi de créer des liaisons entre des oeuvres similaires ou d'en proposer de nouvelles en tenant compte des goûts de chacun. Cette approche favorise une meilleure expérience de lecture et peut aider à découvrir des auteurs ou des styles moins connus.

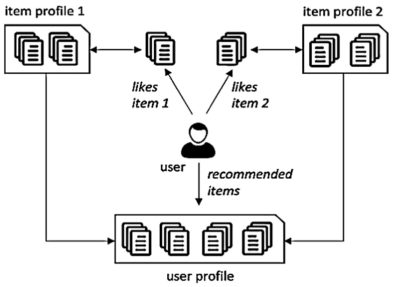


Figure 3.4 – *Content-based recommender system*
Source : Roy et al. (2022, p. 4)

3.1.3 Les systèmes hybrides

Une technique hybride est une combinaison de deux ou plusieurs approches de recommandation. La combinaison de méthode peut se faire de multiple manières. Un algorithme hybride peut, par exemple, fusionner les résultats obtenus par des techniques distinctes ou encore intégrer des aspects du filtrage basé sur le contenu au sein d'une méthode collaborative. Une majorité de systèmes de recommandation industriels ont au moins un élément hybride. La table 3.1 synthétise les différentes approches existantes.

Table 3.1 – *Approches d'hybridation des systèmes de recommandation (Adapté et traduit de Roy et al. (2022, p. 6)*

Méthode d'hybridation	Description
Meta-level	Un modèle pré-appris est utilisé comme donnée d'entrée pour un autre système de recommandation.
Feature combination	Les caractéristiques issues d'un système de recommandation sont injectées (intégrées) dans un autre.
Feature augmentation	Le résultat produit par un modèle est utilisé comme donnée d'entrée pour un autre modèle.
Mixed hybridization	Les sorties de différents systèmes de recommandation sont combinées, et le résultat agrégé est fourni comme recommandation finale.
Cascade hybridization	Un système affine ou perfectionne la sortie produite par un autre système en amont.
Switching hybridization	Sélectionne un modèle de recommandation spécifique en fonction du contexte ou des exigences actuelles de l'utilisateur.
Weighted hybridization	Les scores (ou évaluations/sorties) de différentes techniques sont agrégés, souvent par une moyenne pondérée, pour calculer une recommandation unique.

3.1.4 Autres modèles

Bien que les modèles les plus connus soient les systèmes collaboratifs ou basés sur le contenu, il existe énormément de méthodes, à commencer par le filtrage hybride qui combine plusieurs méthodes.

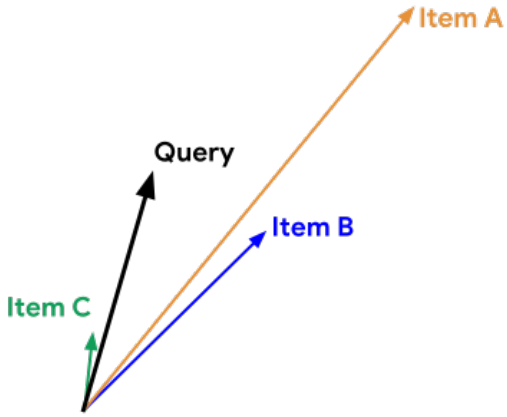
- **Basés sur la connaissance (Knowledge-based)**
  - Recommande des éléments en fonction des exigences explicites de l'utilisateur et des connaissances du domaine. Inclut les systèmes basés sur les contraintes (constraint-based) et les systèmes basés sur les cas (case-based). [6, 1]
- **Sensibles au contexte (Context-aware Recommender Systems) :**
  - Ces systèmes adaptent les recommandations en fonction du contexte actuel de l'utilisateur. Le contexte peut inclure des informations temporelles, spatiales, sociales, ou l'état émotionnel/objectif de l'utilisateur. [53, 1]
- **Démographiques**
  - Utilise les informations démographiques sur l'utilisateur.

3.1.5 Mesures de similarités

Les techniques de filtrage, qu'elles soient basées sur le contenu ou collaboratives, peuvent représenter chaque item (ou élément) ainsi que chaque requête (ou contexte utilisateur associé) sous la forme d'un vecteur au sein d'un espace dit de représentation vectorielle continue (souvent appelé embedding space). Cet espace, noté $E = \mathbb{R}^d$, est généralement de faible dimension, c'est-à-dire que sa dimension d est significativement plus petite que la taille du corpus d'items ou la dimensionnalité initiale des données. L'objectif de cet espace est de capturer la structure latente (ou "cachée") des items et des requêtes. [44] Une mesure de similarité est une fonction $s : E \times E \to \mathbb{R}$ qui, à partir d'une paire de représentations vectorielles continues (embeddings) issues d'un espace $E = \mathbb{R}^d$, renvoie une valeur scalaire mesurant leur ressemblance. Ces représentations vectorielles peuvent être utilisées pour la génération de candidats de la manière suivante : étant donné la représentation vectorielle continue d'une requête $q \in E$, le système recherche les représentations vectorielles continues d'items $x \in E$ qui sont "proches" de q, c'est-à-dire celles présentant une forte similarité $s(q, x)$. Pour déterminer le degré de similarité, la plupart des systèmes de recommandation s'appuient sur une ou plusieurs des mesures suivantes : la similarité cosinus, le produit scalaire, ou la distance euclidienne. Ces mesures peuvent donner des résultat significativement différents comme on peut le voir en figure 3.5.

Figure 3.5 – *Exemple de vecteurs et leur similarité*
Source : Google for Developers, 2024

(a) *Exemples de vecteurs représentant des items et une query*



(b) *Résultats de similarité en fonction des méthodes*

Dot-Product
Query : Item A > Item B > Item C

Cosine
Query : Item C > Item A > Item B

(-) Euclidean Distance
Query : Item B > Item C > Item A

Similarité Cosinus Cette mesure correspond au cosinus de l'angle θ formé par les deux vecteurs q et x. La similarité est donnée par [11, 44] :

$$s(q,x) = \cos(\theta) = \frac{q \cdot x}{\|q\|\|x\|} \tag{3.1}$$

La valeur de $s(q,x)$ varie de -1 (vecteurs opposés) à 1 (vecteurs identiques), 0 indiquant une orthogonalité.

Produit Scalaire Le produit scalaire de deux vecteurs q et x de dimension d est défini par :

$$s(q,x) = \langle q,x \rangle = \sum_{i=1}^{d} q_i x_i \tag{3.2}$$

Il est également relié à l'angle θ entre les vecteurs par l'expression $s(q,x) = \|q\|\|x\| \cos(\theta)$ (c'est-à-dire le cosinus de l'angle multiplié par le produit des normes des vecteurs). Ainsi, si les représentations vectorielles continues sont normalisées (de sorte que $\|q\| = 1$ et $\|x\| = 1$), le produit scalaire et la similarité cosinus coïncident. [13, 44]

Distance Euclidienne Il s'agit de la distance géométrique usuelle dans un espace euclidien de dimension d :

$$d(q,x) = \|q - x\| = \left[\sum_{i=1}^{d} (q_i - x_i)^2\right]^{1/2} \tag{3.3}$$

Une distance plus *faible* indique une similarité plus *élevée*. Par conséquent, la distance elle-même est une mesure de dissimilarité. Pour l'utiliser comme mesure de similarité $s(q,x)$, on a souvent recours à une transformation (par exemple, $s(q,x) = \frac{1}{1+d(q,x)}$ ou une fonction exponentielle négative de la distance). Il est important de noter que lorsque les représentations vectorielles continues sont normalisées à une norme unitaire ($\|q\| = \|x\| = 1$), la distance euclidienne au carré est directement reliée au produit scalaire (et donc à la similarité cosinus), à une constante près, puisque dans ce cas [15, 44] :

$$\frac{1}{2}\|q - x\|^2 = 1 - \langle q,x \rangle \tag{3.4}$$

3.1.6 Les défis des systèmes de recommandation

Les systèmes de recommandation représentent de nombreux défis parmi lesquels :

- **Démarrage à froid (Cold Start)** : Le démarrage à froid fait référence à la difficulté pour un système de recommandation de formuler des suggestions pertinentes pour de nouveaux utilisateurs (n'ayant pas encore d'historique d'interaction) ou pour de nouveaux items (n'ayant pas encore été évalués ou décrits en détail). [71]
- **Data sparsity** : Cela se réfère à la situation où la quantité de données disponibles sur les interactions entre les utilisateurs et les items (produits, films, chansons, etc.) est très faible par rapport au nombre total possible d'interactions. [8]
- **Scalabilité** : Gérer des millions d'utilisateurs et de livres nécessite des algorithmes efficaces et des infrastructures robustes. [8]
- **Serendipité vs. Précision** : Trouver un équilibre entre recommander des livres évidents (très similaires à ceux déjà lus) et surprendre l'utilisateur avec des découvertes inattendues mais pertinentes est un enjeu majeur. [25]
- **Longue traîne (Long Tail)** : Mettre en avant des oeuvres moins populaires mais de qualité, au-delà des best-sellers, par exemple. [48]

3.2 Evaluer un système de recommandation

L'évaluation des systèmes de recommandation est une étape fondamentale, que ce soit pour intégrer la meilleure solution ou pour valider de nouvelles approches algorithmiques. L'exactitude des recommandations a longtemps constitué le principal critère d'évaluation, néanmoins, les utilisateurs n'utilisent pas nécessairement un système d'évaluation comme un prédicteur précis de leurs intérêts. Ils peuvent, par exemple, chercher à découvrir des nouvelles choses, surprenantes et diverses, ce qui ne peut pas être simplement évalué par l'exactitude (accuracy). [53, 1]

3.2.1 Les métriques de l'évaluation

Il existe de multiples métriques communes permettant d'évaluer un système de recommandation qui sont présentées ci -dessous.

- **Précision de la Prédiction (Prediction Accuracy)**
  - Mesure la capacité du système à prédire correctement les évaluations (numériques ou binaires) d'un utilisateur pour un élément, ou l'exactitude du classement des éléments recommandés. [53, 1]
- **Couverture (Coverage)**
  - Évalue la proportion d'éléments du catalogue total que le système *peut* recommander. Un système performant devrait avoir une bonne couverture pour proposer des éléments variés, y compris ceux de niche. [53, 1]
- **Nouveauté (Novelty)**
  - Détermine la probabilité que le système recommande des éléments que l'utilisateur ne connaissait pas auparavant, ajoutant ainsi de la valeur en faisant découvrir de nouveaux éléments pertinents. [53, 1]
- **Sérendipité (Serendipity)**
  - Mesure le degré de surprise dans les recommandations *réussies* (pertinentes) sont surprenantes. Il s'agit de recommander des éléments utiles et intéressants auxquels l'utilisateur ne s'attendait pas, allant au-delà des préférences évidentes. [30, 53, 1]
- **Diversité (Diversity)**
  - S'assure que la liste des recommandations n'est pas répétitive et offre une variété d'éléments. [31, 53, 1]
- **Confiance de l'Utilisateur (User Trust)**
  - Mesure la perception qu'a l'utilisateur de la fiabilité et de la crédibilité des recommandations fournies par le système, souvent influencée par la transparence et la qualité des explications.

[53, 1]

- **Adaptabilité (Adaptivity)**
  - Mesure la capacité du système à ajuster rapidement ses recommandations en fonction de l'évolution des préférences de l'utilisateur ou des dynamiques changeantes du catalogue d'éléments ou du marché. [53, 1]
- **Scalabilité (Scalability)**
  - Évalue la performance du système (temps d'entraînement, temps de prédiction, exigences en mémoire) à mesure que le nombre d'utilisateurs et d'éléments augmente. [53, 1]

3.2.2 Les paradigmes d'évaluation

Une fois que l'on a cerné les différents aspects qui définissent la performance et l'utilité d'un système de recommandation, il est important d'examiner comment mesurer ces critères.

3.2.3 Les études utilisateur

Dans les études utilisateurs, des participants sont activement recrutés pour interagir avec un système de recommandation et accomplir des tâches spécifiques. Leur feedback, avant et après l'interaction, ainsi que les données sur leur interaction avec le système (par exemple, évaluer la qualité des recommandations sur un site de produits ou noter des chansons écoutées), sont collectés pour en déduire leurs préférences et juger de l'efficacité des algorithmes. L'un des principaux avantages de ces études est la possibilité de recueillir des informations détaillées sur l'interaction de l'utilisateur avec le système et de tester l'effet de divers changements, que ce soit au niveau de l'algorithme ou de l'interface utilisateur. Cependant, cette approche présente plusieurs inconvénients. La conscience qu'ont les utilisateurs d'être testés peut souvent biaiser leurs choix et actions. De plus, il est difficile et coûteux de recruter des utilisateurs pour de tels tests. [53, 55]

3.2.4 L'évaluation en ligne

Dans le cas de l'évaluation en ligne, le système est testé sur des utilisateurs réels et en conditions réelles. Cette approche permet de comparer les performances de divers algorithmes, par exemple en utilisant des tests A/B où des échantillons aléatoires d'utilisateurs sont exposés à différentes versions du système pour mesurer des métriques comme le taux de conversion (la fréquence à laquelle un utilisateur sélectionne un item recommandé).

Cependant, les principaux inconvénients de l'évaluation en ligne sont qu'elle nécessite une base d'utilisateurs conséquente, la rendant difficile à mettre en oeuvre en phase de démarrage. [53, 55]

3.2.5 L'évaluation hors-ligne

L'évaluation hors ligne des systèmes de recommandation s'appuie sur des données historiques, telles que des ratings, parfois accompagnés d'un timestamp. L'ensemble Netflix Prize [72] est un exemple de référence pour les films. Ses avantages majeurs résident dans l'absence de nécessité d'une base d'utilisateurs actifs, la capacité à établir des benchmarks standardisés pour comparer les algorithmes. C'est une des méthodes les plus populaires car elle permet une évaluation standardisée. [53, 55]

3.3 Paradigmes de représentation textuelle

La recommandation de livres basée sur le contenu nécessite une compréhension fine et profonde des textes. Le défi majeur réside dans la longueur, la complexité narrative et la richesse sémantique des oeuvres littéraires complètes. La transformation du texte brut en une représentation numérique exploitable par des algorithmes est une étape cruciale. Plusieurs paradigmes ont émergé au fil du temps

3.3.1 Approches statistiques traditionnelles

Les premières tentatives de représentation textuelle reposaient sur des modèles statistiques simples. Le modèle Sac de Mots (Bag-of-Words, BoW) représente un document comme un ensemble non ordonné

de ses mots, faisant abstraction de la grammaire et de l'ordre. Chaque document est alors un vecteur dont chaque dimension correspond à un mot du vocabulaire global, et la valeur est typiquement la fréquence de ce mot dans le document. [5]

La méthode **TF-IDF (Term Frequency-Inverse Document Frequency)** affine cette approche en pondérant l'importance d'un mot non seulement par sa fréquence dans un document (TF) mais aussi par son inverse de fréquence dans l'ensemble du corpus (IDF) [54]. Ainsi, les mots fréquents dans un document mais rares dans le corpus global reçoivent un poids plus élevé, les considérant comme plus discriminants.

Aujourd'hui, les approches BoW et TF-IDF sont principalement utilisées comme des méthodes de référence (baselines) pour évaluer des techniques plus avancées, ou pour des tâches d'extraction d'informations plus ciblées où la sémantique profonde n'est pas l'objectif premier.

3.3.2 Sémantique distributionnelle et plongements lexicaux (Embeddings) Pré-LLM

L'hypothèse distributionnelle, selon laquelle "un mot se caractérise par les compagnies qu'il fréquente"(Firth, 1957) [18], a ouvert la voie à des représentations vectorielles denses, ou plongements lexicaux (word embeddings). Des modèles comme *Word2Vec* (Mikolov et al., 2013) [37], avec ses architectures CBOW (Continuous Bag-of-Words) et Skip-gram, GloVe (Pennington et al., 2014) [20], et *FastText* (Bojanowski et al., 2017) [16] apprennent des vecteurs de mots à partir de grands corpus, de sorte que les mots sémantiquement similaires aient des représentations vectorielles proches dans l'espace.

Pour représenter des documents entiers, des extensions comme *Doc2Vec* (Le & Mikolov, 2014) [32], aussi connu sous le nom de Paragraph Vectors, ont été proposées. D'autres approches consistaient à agréger les embeddings des mots composant le document (par exemple, par moyenne ou somme pondérée).

3.3.3 L'avènement des Transformers

L'introduction de l'architecture Transformer (Vaswani et al., 2017) [68] a constitué une révolution. Son mécanisme clé, l'auto-attention (self-attention), permet au modèle de pondérer l'importance de différents mots dans une séquence lors du traitement de chaque mot, capturant ainsi les dépendances contextuelles indépendamment de leur distance. Le traitement parallèle des tokens a également permis d'entraîner des modèles sur des quantités de données beaucoup plus importantes. Cette architecture a directement ouvert la voie au développement des Grands Modèles de Langage (LLM) pré-entraînés, tels que BERT (Devlin et al., 2019) [12], RoBERTa (Liu et al., 2019) [35], et la série GPT (Radford et al., 2018) [50]. Ces modèles, entraînés sur des corpus textuels colossaux, constituent aujourd'hui le paradigme dominant pour la majorité des tâches de traitement automatique du langage naturel, y compris la compréhension de texte profond, et seront détaillés dans la section suivante.

3.4 Les grands modèles de langage (LLM)

Un Grand Modèle de Langage (LLM) est une catégorie de modèles d'intelligence artificielle spécifiquement conçus pour la compréhension et la génération du langage humain. Fondés sur des architectures de réseaux de neurones, typiquement des Transformers, et entraînés sur de vastes corpus de données textuelles (livres, articles, contenu web), les LLMs acquièrent une connaissance approfondie des structures syntaxiques, des relations sémantiques et des nuances contextuelles du langage. Leur dénomination de *grands* découle du nombre massif de paramètres (de l'ordre de milliards) qu'ils comportent, leur conférant la capacité de traiter des informations d'une grande complexité et de produire des réponses élaborées.

Le fonctionnement des LLM repose sur plusieurs technologies : Le Traitement du Langage Naturel (NLP) leur permet d'analyser et d'interpréter le texte entrant, tandis que la Génération du Langage

Naturel (NLG) leur permet de produire du texte de manière autonome. Ces capacités sont rendues possibles par des réseaux de neurones organisés en couches et par l'apprentissage profond (Deep Learning), qui permet le traitement et l'apprentissage à partir de très grandes quantités de données. L'architecture Transformer, mentionnée précédemment, est cruciale grâce à son mécanisme d'auto-attention.

La phase initiale de pré-entraînement (pre-training) consiste à exposer le modèle à des volumes massifs et diversifiés de données textuelles, lui permettant d'apprendre les fondements du langage (grammaire, vocabulaire, faits du monde, styles). Par la suite, une étape d'affinage (fine-tuning) peut être appliquée pour spécialiser le modèle pré-entraîné à des tâches ou des domaines spécifiques, en l'entraînant sur un corpus de données plus restreint et ciblé.

Parmi les LLM influents, on peut citer :

La série GPT (Generative Pre-trained Transformer) d'OpenAI, pionnière dans les capacités de génération de texte à grande échelle. [76] BERT (Bidirectional Encoder Representations from Transformers) de Google [35] et sa variante optimisée RoBERTa (Facebook AI) [35], qui ont innové par leur compréhension contextuelle bidirectionnelle. Gemini, une famille de modèles multimodaux développée par Google DeepMind. [74] Les modèles de Mistral AI, comme Mistral 7B et Mixtral 8x7B, reconnus pour leur efficacité et leurs performances en open source. [19] Llama et ses successeurs, développés par Meta AI, qui ont également eu un impact significatif dans le domaine des modèles ouverts. [67]

3.4.1 Les limitations des LLMs

- État transitoire (Transient state) : Les LLM sont intrinsèquement dépourvus de mémoire persistante ou d'état, ce qui nécessite des logiciels ou des systèmes supplémentaires pour la rétention et la gestion du contexte.
- Nature probabiliste : La nature stochastique des LLM introduit une variabilité dans les réponses.
- Informations obsolètes : La dépendance aux données de pré-entraînement restreint les LLM à des connaissances historiques [2]
- Fabrication de contenu (hallucination) : Les LLMs peuvent générer des information factuellement incorrectes. [26]

3.4.2 Prompt engineering

Le prompt engineering constitue une discipline axée sur la conception et l'élaboration de requêtes textuelles (prompts) destinées à optimiser la performance des LLMs. L'objectif principal est de susciter des réponses pertinentes et exactes en exploitant au mieux les capacités de ces modèles. En effet, les prompts peuvent avoir une importance non négligeable sur la qualité des réponses. (Barkley et al. 2024) [4] ont trouvé que les techniques de prompts pouvaient influencer le taux d'hallucination et que différentes stratégies de prompting étaient préférables selon la tâche demandée. Il existe énormément de stratégies de prompting, parmi lesquelles :

Zero-Shot Prompting Cette technique consiste à solliciter un LLM pour une tâche sans lui fournir d'exemples de résolution au préalable. Le modèle est censé interpréter et exécuter la tâche en s'appuyant uniquement sur sa formation initiale et la description fournie dans le prompt. [47, 49]

- *Exemple d'application :* Demander au LLM de résumer un document ou de déterminer le sentiment d'un texte sans lui montrer d'exemples.

Few-Shot Prompting À l'opposé du "zero-shot", cette méthode incorpore dans l'invite quelques exemples (généralement de 1 à 5) qui illustrent la tâche à réaliser et le format de réponse souhaité. Ces démonstrations orientent le LLM et améliorent de manière notable la qualité et la pertinence de ses réponses pour des tâches plus subtiles. [47]

— *Exemple d'application :* Présenter 2-3 paires de phrases (entrée) et leur traduction (sortie attendue) avant de demander la traduction d'une nouvelle phrase. Pour l'extraction de métadonnées, on pourrait exposer des exemples de textes et les métadonnées structurées correspondantes.

Chain-of-Thought Prompting (CoT) Pour les tâches requérant un raisonnement complexe (problèmes mathématiques, questions de logique, inférences multi-étapes), la technique CoT incite le LLM à décomposer le problème et à verbaliser son processus de raisonnement étape par étape avant de livrer la réponse finale. Ceci est fréquemment activé en incluant dans les exemples du "few-shot prompting" non seulement la question et la réponse, mais aussi les étapes intermédiaires du raisonnement. Une simple directive comme "Réfléchis étape par étape" peut aussi déclencher un comportement CoT en mode "zero-shot". [47]

— *Exemple d'application :* Pour identifier les thèmes complexes d'un ouvrage, on pourrait demander au LLM d'abord de repérer les événements majeurs, puis les motivations des personnages, ensuite les conflits, et enfin de synthétiser les thèmes.

Role Prompting Cette approche consiste à attribuer un rôle ou une "persona" spécifique au LLM dans l'invite (par exemple, "Tu es un critique littéraire aguerri...", "Comporte-toi comme un bibliothécaire expert en littérature du XIXe siècle..."). Cela aide à contextualiser la tâche et à orienter le style, le ton, et le type d'informations que le modèle produira. [47]

— *Exemple d'application :* Pour l'extraction de thèmes, assigner le rôle d'un "critique littéraire" pourrait engendrer des résultats plus approfondis.

Tree of Thoughts (ToT) Cette technique permet aux modèles de langage d'aborder des problèmes complexes en explorant de manière structurée différentes voies de raisonnement. Au lieu de suivre une unique chaîne de pensée, ToT érige un arbre où chaque "pensée" constitue une étape intermédiaire vers la solution. Le modèle de langage peut ainsi auto-évaluer la progression de chaque branche et recourir à des algorithmes de recherche (tels que la recherche en largeur ou en profondeur) pour explorer méthodiquement ces pensées, y compris par anticipation et retour en arrière si une piste s'avère peu prometteuse. Cette approche se caractérise par sa faculté à décomposer le problème, à générer plusieurs options pour chaque étape, et à les évaluer pour ne retenir que les plus pertinentes. [47]

— *Exemple d'application :* Pour un problème de planification complexe, le LLM génère plusieurs scénarios initiaux (branches), évalue la faisabilité de chaque étape, explore les scénarios prometteurs et abandonne les impasses.

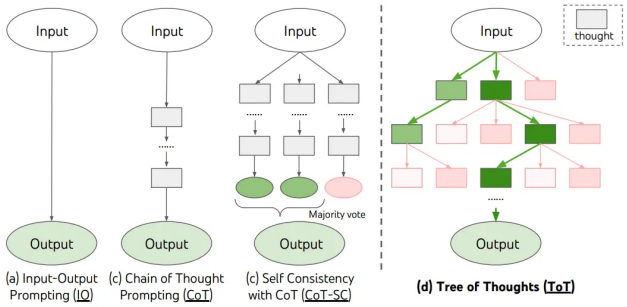


Figure 3.6 – *Schéma illustrant diverses approches de résolution de problèmes avec les LLM. Chaque boîte rectangulaire représente une pensée, qui est une séquence de langage cohérente servant d'étape intermédiaire pour la résolution de problèmes*
Source : Yao et al. (2023, p. 2)

ReAct Ce cadre méthodologique permet aux LLM de produire des "traces de raisonnement" (étapes logiques explicites) et des "actions" spécifiques à la tâche. La génération de traces de raisonnement

dote le modèle de la capacité d'élaborer, de suivre et d'ajuster dynamiquement ses plans d'action, tout en gérant les exceptions. Parallèlement, l'exécution d'actions concrètes lui donne la possibilité d'interagir avec des sources d'information externes (bases de connaissances, autres environnements) pour recueillir des données supplémentaires. L'approche la plus performante combine souvent ReAct et CoT. [47]

— *Exemple d'application :* Pour répondre à une question d'actualité, le LLM identifie les entités clés, formule une requête pour un moteur de recherche (action), analyse les résultats et synthétise une réponse.

Self-Consistency Cette technique vise à améliorer la fiabilité des réponses, notamment pour les tâches de raisonnement complexes. Elle consiste à interroger le LLM plusieurs fois sur la même question, en l'incitant à générer à chaque fois une chaîne de pensée différente (par exemple, en variant légèrement le prompt ou en utilisant une température de génération plus élevée). La réponse finale est ensuite déterminée en sélectionnant la conclusion la plus fréquente parmi les diverses chaînes de pensée produites. [47]

— *Exemple d'application :* Pour un problème mathématique, le LLM est invité à exposer son raisonnement plusieurs fois. Si la majorité des raisonnements distincts aboutissent à la même solution, celle-ci est considérée comme étant plus fiable.

Retrieval Augmented Generation (RAG) La Génération Augmentée par Récupération (RAG) est une technique qui enrichit la capacité de génération des LLM en les connectant à des sources de connaissances externes et à jour. Avant de générer une réponse, le système RAG récupère des informations pertinentes (des "chunks" de texte) depuis une base de données documentaire (souvent une base vectorielle). Ces informations sont ensuite injectées dans le prompt soumis au LLM, lui permettant de fonder sa réponse sur des données factuelles, spécifiques et récentes, réduisant ainsi les hallucinations. [47]

— *Exemple d'application :* Pour répondre à une question sur une nouvelle réglementation, le système RAG recherche d'abord les textes de loi et articles pertinents dans sa base documentaire, puis le LLM utilise ces extraits pour formuler une réponse précise et actualisée.

3.5 Stockage et gestion des embeddings dans les bases de données relationnelles

L'avènement des modèles de langage de grande taille (LLMs) et des applications basées sur l'intelligence artificielle a considérablement accru l'importance des *embeddings* vectoriels. Un **embedding** est une représentation numérique de données (texte, images, audio, etc.) sous forme de vecteur, où la proximité dans l'espace vectoriel reflète la similarité sémantique ou conceptuelle. Le stockage et la gestion efficaces de ces vecteurs sont devenus un défi central pour le développement d'applications telles que la recherche sémantique, les systèmes de recommandation ou la génération augmentée par récupération (RAG).

3.5.1 Stockage des embeddings avec `pgvector`

Traditionnellement, les bases de données relationnelles comme PostgreSQL n'étaient pas nativement conçues pour manipuler des vecteurs de haute dimension ou effectuer des recherches de similarité complexes. Cependant, l'émergence d'extensions dédiées a transformé PostgreSQL en une solution viable et puissante pour les *vector databases*. L'extension `pgvector` [43] est devenue la solution de facto pour stocker et interroger des embeddings directement au sein d'une base de données PostgreSQL.

`pgvector` introduit un nouveau type de données `VECTOR` qui permet aux utilisateurs de définir des colonnes capables de contenir des tableaux de nombres à virgule flottante de n'importe quelle dimensionnalité. En plus de ce type de données, `pgvector` fournit des opérateurs optimisés pour calculer différentes métriques de similarité ou de distance entre les vecteurs, telles que la similarité cosinus, la

distance Euclidienne (L2) ou le produit interne. Cela permet d'exécuter des requêtes SQL pour trouver les "plus proches voisins" d'un vecteur donné, ce qui est le fondement de la recherche sémantique. Pour garantir des performances acceptables sur des ensembles de données volumineux, `pgvector` supporte également des méthodes d'indexation spécifiques, comme `IVFFlat` et `HNSW` (Hierarchical Navigable Small World) [42]. Ces index sont des algorithmes de recherche de plus proches voisins approximatifs (ANN - Approximate Nearest Neighbor) qui équilibrent la vitesse de la requête avec la précision des résultats, devenant indispensables pour des millions, voire des milliards de vecteurs. L'intégration des embeddings directement dans PostgreSQL simplifie l'architecture en évitant le besoin d'une base de données vectorielle séparée, permettant de co-localiser les données structurées et les embeddings.

De plus, des systèmes comme *TimescaleDB*, bien que reconnus principalement pour leurs optimisations des **séries temporelles**, sont de plus en plus utilisés comme bases de données polyvalentes pour les applications d'IA et de RAG [64, 65]. L'utilisation d'une image Docker `timescale/timescaledb-ha:pg17` dans un contexte RAG, même sans un usage intensif des fonctionnalités de séries temporelles, est pertinente. TimescaleDB améliore la performance et la scalabilité de PostgreSQL de manière générale, ce qui est bénéfique pour des workloads de données importants, qu'ils soient transactionnels, analytiques ou vectoriels. En tirant parti de PostgreSQL comme base de données vectorielle unique, on bénéficie de la robustesse, de la cohérence transactionnelle et de l'écosystème riche de PostgreSQL, tout en ayant les capacités nécessaires pour les applications modernes d'IA.

3.5.2 Automatisation et gestion des embeddings

Le simple stockage des embeddings est une première étape, mais le véritable défi réside dans leur gestion continue : comment s'assurer que les embeddings restent synchronisés avec les données textuelles sous-jacentes à mesure qu'elles sont insérées, mises à jour ou supprimées ? Comment gérer les appels aux API externes de génération d'embeddings qui peuvent être lents, coûteux ou sujets à des erreurs ?

C'est là que des solutions comme l'extension **`pgai`** [63, 66] (et des concepts similaires dans d'autres écosystèmes [57]) prennent tout leur sens. `pgai` vise à automatiser la création et la gestion des embeddings, les traitant comme des "index" intelligents sur les données. Au lieu que l'application cliente gère manuellement la logique de génération et de mise à jour des vecteurs, `pgai` permet de déclarer des "vectorizers" directement dans la base de données via SQL. Ces "vectorizers" définissent des pipelines qui spécifient :

- La **source des données** (par exemple, une colonne de texte dans une table PostgreSQL).
- Les **étapes de prétraitement** (comme le découpage du texte en "chunks" gérables ou le formatage).
- Le **modèle d'embedding** à utiliser (via une API externe comme OpenAI, Ollama, etc.).
- La **destination** où les embeddings générés doivent être stockés (souvent une table séparée liée à la source).

Le code en 3.1 montre la création d'un tel vectorizer dans la base de donnée.

```
SELECT ai.create_vectorizer(
           'themes'::regclass,
           loading => ai.loading_column(column_name => 'theme_name'),
           destination => ai.destination_column('theme_embedding'),
           chunking => ai.chunking_none(),
           embedding => ai.embedding_ollama(
                   'nomic-embed-text',
                   768,
                   base_url => 'http://ollama:11434/'));
```

Listing 3.1 – *Example de création d'un vectorizer dans postgres*

Ce processus est généralement exécuté de manière asynchrone par des "workers" dédiés (tel que le `pgai_worker`) qui surveillent les changements dans les données sources. Lorsque des données sont insérées ou modifiées, des mécanismes internes (souvent des triggers ou des systèmes de file d'attente)

signalent au worker qu'un nouvel embedding doit être généré ou mis à jour. Le worker effectue ensuite l'appel au modèle d'IA externe et stocke le résultat dans la base de données. L'avantage majeur de cette approche réside dans sa résilience et sa simplicité opérationnelle [61, 56]. Les opérations de modification des données primaires de l'application ne sont pas bloquées par la latence ou les défaillances des services d'embedding externes. Le système gère automatiquement les files d'attente, les tentatives en cas d'échec et la synchronisation des embeddings en arrière-plan, réduisant considérablement la complexité de l'infrastructure d'IA pour les développeurs. Cela permet aux applications de bénéficier pleinement des capacités de recherche sémantique et de RAG sans les fardeaux de l'orchestration manuelle des embeddings. Pour une compréhension visuelle de ce pipeline, le schéma d'architecture de `pgai` disponible dans la documentation du dépôt GitHub est visible en figure 3.7

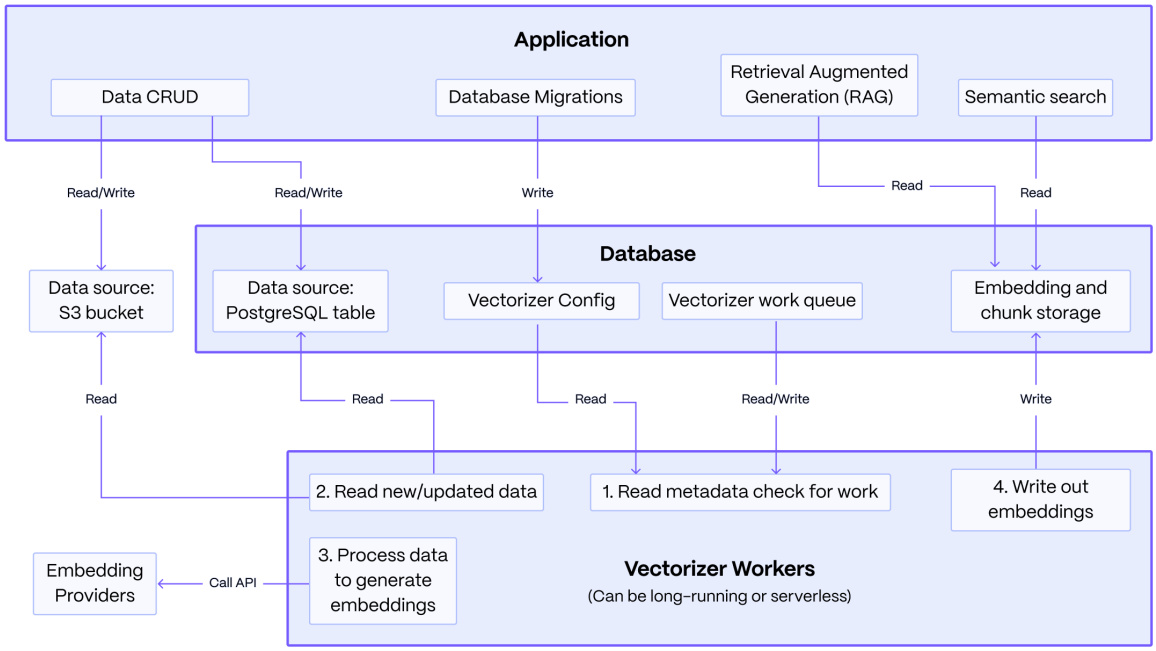


Figure 3.7 – *Architecture de pgai (source : Github pgai [62])*

3.6 Etudes pertinentes et industrie

3.6.1 Les systèmes de recommandation industriels

Amazon

Amazon est une référence en matière de systèmes de recommandation.

- Utilisation intensive du filtrage collaboratif item-item ("Les clients ayant acheté cet article ont également acheté...").
- Analyse de l'historique de navigation, des achats, des listes de souhaits, et même des passages surlignés dans les Kindle.
- Personnalisation en temps réel : la page d'accueil et les recommandations de produits changent dynamiquement en fonction des interactions récentes.
- Des services comme Amazon Personalize [58] (basé sur AWS) permettent à d'autres entreprises de bâtir des systèmes similaires, en s'appuyant sur des modèles pré-entraînés ou en entraînant des modèles sur leurs propres données.
- Des recherches récentes sur des jeux de données comme "Amazon-Book" explorent des modèles

avancés, incluant des réseaux de neurones graphiques (Graph Neural Networks) [78] et des approches de factorisation de matrices pour améliorer la précision [33].

Goodreads

Goodreads, une plateforme sociale axée sur les livres et détenue par Amazon, adopte une approche communautaire et algorithmique :

- Les utilisateurs cataloguent leurs lectures via des "étagères" (ex : "lu", "en cours de lecture", "à lire") et attribuent des notes.
- Le système utilise ces données pour du filtrage collaboratif user-based et item-based. Un document de l'Université de Stanford datant de 2008 décrivait déjà l'utilisation de K-Nearest Neighbors pour regrouper les utilisateurs [27].
- Goodreads a acquis Discovereads en 2011, un moteur de recommandation basé sur l'apprentissage machine, pour renforcer ses capacités [29].
- Les recommandations sont basées sur les livres notés, les genres préférés et les amis suivis sur la plateforme.

Kobo (Rakuten Kobo)

Kobo, un acteur majeur dans le domaine des liseuses et des ebooks :

- Exploite le big data issu des interactions de millions de lecteurs.
- Analyse les titres de la bibliothèque d'un utilisateur via des techniques d'apprentissage machine non supervisées pour extraire les principaux sujets d'intérêt et pondérer ces intérêts.
- Prend en compte le contenu réel des ebooks, les métadonnées (auteur, série, genre), la popularité, les notes et la date de publication.
- Les recommandations sont diffusées sur la page d'accueil, les pages produits et par email (ex : "Votre prochaine lecture") [51].

3.6.2 Dans la recherche

Domaines associés

- L'étude "BooookScore" de Chang et al. (2024) [9] s'attaque au résumé de livres par LLMs, limité par la taille de leur contexte. Elle compare deux stratégies pour traiter les textes longs segmentés : la fusion hiérarchique de résumés et la mise à jour incrémentale d'un résumé global . Face aux difficultés d'évaluation (contamination des datasets, métriques inadaptées), les auteurs proposent BooookScore, une métrique automatique utilisant un LLM pour évaluer la cohérence des résumés en détectant 8 types d'erreurs prédéfinis . L'étude met en évidence un compromis : la fusion hiérarchique améliore la cohérence au détriment du détail, tandis que la mise à jour incrémentale favorise le détail mais réduit la cohérence
- La revue de Cao [7] analyse les progrès récents concernant les *embeddings de texte universels*, en se basant sur les modèles les plus performants du benchmark MTEB [14]. L'objectif est de créer des représentations généralisables à travers diverses tâches, domaines et langues. Les avancées majeures proviennent de l'amélioration des données d'entraînement (quantité, qualité, diversité, données synthétiques générées par LLM), de fonctions de perte innovantes (ex : AnglE, MRL/2dMSE), et de l'intégration des LLMs(utilisés comme backbones finetunés ou comme générateurs de données, souvent avec instruction tuning). Bien que les performances moyennes sur MTEB aient considérablement augmenté, l'étude montre que les gains varient fortement selon les tâches : très importants pour la recherche d'information (Retrieval), mais faibles pour l'évaluation de résumés (Summarization). Cette revue souligne donc l'importance des techniques récentes et la nécessité d'une évaluation spécifique à la tâche pour choisir un modèle d'embedding.

Systèmes de recommandation

— L'étude de Liu et al. [34] propose le framework ONCE pour améliorer la recommandation basée sur le contenu en exploitant conjointement les LLMs *open-source* et *closed-source*. D'une part, l'approche DIRE utilise des LLMs open-source (comme LLaMA) finetunés comme encodeurs pour générer de meilleurs *embeddings* de contenu D'autre part, l'approche GENRE emploie des LLMs closed-source (comme GPT-3.5) via prompting pour enrichir les données d'entraînement (par exemple, en générant de meilleures descriptions d'items ou des profils utilisateurs). Les auteurs démontrent l'efficacité de chaque approche et une synergie bénéfique lorsqu'elles sont combinées, résultant en des gains de performance significatifs sur des tâches de recommandation de livres et d'actualités.

— L'étude de Zhiyuli et al. [77] propose le framework BookGPT, qui utilise un LLM (ChatGPT) via prompting pour diverses tâches de recommandation de livres. Particulièrement pertinent pour l'explicabilité des recommandations, l'article montre que les LLMs peuvent générer efficacement des résumés de livres. Ces résumés, souvent jugés de bonne qualité (parfois préférés aux résumés humains, peuvent servir directement d'explication ou de justification pour une recommandation. L'étude met aussi en avant la capacité potentielle des LLMs à produire des explications personnalisées en intégrant des informations spécifiques à l'utilisateur dans le prompt. Le succès de l'approche dépend fortement d'une ingénierie de prompts soignée, tout en restant vigilant face au risque d'erreurs factuelles dans le texte généré.

— "Enhancing Recommender Systems with Large Language Model Reasoning Graphs" par Wang et al. (2024) [69], présente une approche novatrice appelée LLM Reasoning Graphs (LLMRG), visant à améliorer significativement les systèmes de recommandation. L'objectif principal de cette recherche est de surmonter les limitations des systèmes de recommandation traditionnels, qui souffrent souvent d'un manque d'interprétabilité et d'une difficulté à capturer les relations sémantiques de haut niveau entre les comportements passés et les profils des utilisateurs. Les LLMRG proposent de construire des graphes de raisonnement personnalisés en exploitant les capacités des grands modèles de langage (LLM). Ces graphes ne se contentent pas de lier les profils des utilisateurs et leurs séquences comportementales de manière superficielle ; ils le font à travers des inférences causales et logiques. Cela permet de représenter les intérêts des utilisateurs d'une manière beaucoup plus interprétable et nuancée, allant au-delà des simples corrélations observées.

Chapitre 4

Choix et approches

4.1 Sélection et caractérisation du corpus

L'efficacité d'un système de recommandation de livres basé sur des modèles de langage (LLM) dépend intrinsèquement de la pertinence et de la richesse des données utilisées pour l'entraînement ou l'analyse.

4.1.1 Analyse de sources potentielles

Plusieurs sources de données ont été évaluées :

Projet Gutenberg

Le Project Gutenberg est une bibliothèque numérique en ligne offrant gratuitement une vaste collection de livres électroniques. Les textes fournis sont essentiellement du domaine public (droits d'auteur expirés ou jamais sujets), bien qu'il contienne quelques textes sous droit d'auteur avec permission. Il a été fondé en 1971 par Michael S. Hart et aujourd'hui compte plus de 70'000 livres.

Avantages

- Accès libre et gratuit au texte intégral des livres.
- Les données sont généralement propres et structurées pour le format numérique.

Inconvénients

- Le catalogue est majoritairement composé d'oeuvres anciennes tombées dans le domaine public, ce qui limite la diversité des genres et des styles contemporains.
- Ne fournit pas nativement de descriptions ou de résumés détaillés, mais plutôt les textes bruts.

CMU Books Summary

Ce jeu de données propose des résumés d'intrigue (plot summaries) pour 16 559 ouvrages, extraits de Wikipedia, ainsi que des métadonnées alignées provenant de Freebase, incluant l'auteur du livre, le titre et le genre. [10]

Avantages

- Possède des résumés potentiellement plus complets qu'une simple description de quatrième de couverture.

Inconvénients

- Ne fournit pas le texte intégral des livres, seulement des résumés.
- La qualité et la longueur des résumés, provenant de Wikipedia, ne sont pas uniformes et peuvent varier considérablement, voire être manquants. Parfois le résumé est complet, mais parfois s'apparente à un synopsis.

Google Books

Google Books est un service de numérisation et de consultation de livres. Il offre un vaste catalogue et une API (Google Books API) pour accéder aux métadonnées [21].

Avantages
- Très grand nombre de livres référencés, dans de nombreuses langues.
- Google Books API est gratuite pour l'accès aux métadonnées et, parfois, à des aperçus.
- Descriptions (souvent la quatrième de couverture) et autres métadonnées riches sont disponibles.

Inconvénients
- L'accès au texte intégral est très limité ou inexistant pour la majorité des ouvrages sous droit d'auteur.
- La gestion des différentes éditions d'un même livre peut être complexe.

OpenLibrary

OpenLibrary est un projet de Internet Archive, une bibliothèque en ligne à but non lucratif visant à rendre les textes accessibles. [70] OpenLibrary propose des dumps mensuels contenant toutes leurs données sous forme de fichiers tabulés.

Avantages
- Vaste catalogue de livres et de métadonnées.
- Propose des dumps complets facilitant le traitement hors ligne.
- Contient des liens vers des textes intégraux lorsque ceux-ci sont disponibles (souvent via Internet Archive, pour les oeuvres du domaine public ou prêt numérique).

Inconvénients
- L'accès systématique au texte intégral pour une large échelle reste un défi, dépendant de la disponibilité via Internet Archive (qui a ses propres règles).
- Les livres ne sont pas toujours disponibles en format électroniques.
- L'API de OpenLibrary ne doit pas être utilisée pour des bulk downloads [3].

4.1.2 Sélection de la source de données pour le projet

Suite à l'analyse comparative des différentes sources, le *Projet Gutenberg* a été retenu comme source principale de données pour la phase initiale de ce projet.

Cette décision est motivée par la volonté d'avoir accès au texte intégral des livres. Bien que le catalogue du Projet Gutenberg soit moins diversifié en termes de modernité et de genres par rapport à des sources comme Google Books ou OpenLibrary, qui offrent une richesse de métadonnées et un plus grand nombre de titres, ces dernières ne garantissent pas un accès systématique au texte intégral.

En conséquence, le Projet Gutenberg offre le meilleur compromis pour l'objectif principal de ce projet : explorer et mettre en oeuvre des méthodes de traitement basées sur le contenu intégral des livres avec des LLM, en dépit de la limitation de son catalogue aux oeuvres majoritairement classiques et du domaine public. Les techniques développées pourront potentiellement être adaptées et appliquées à d'autres sources si l'accès au texte intégral devient viable à plus grande échelle. La suite va présenter plus en détails les différentes informations disponibles sur le projet Gutenberg.

4.2 Acquisition et nettoyage des données

Le Projet Gutenberg propose des livres numériques dans plusieurs formats de fichiers. Pour la pipeline, le format de texte brut `text/plain; charset=utf-8` a été privilégié pour sa simplicité et sa compatibilité directe avec les outils de traitement textuel.

De nombreuses métadonnées sont disponibles et sont détaillées en annexe B.

Le nettoyage des données est une étape essentielle pour transformer les textes bruts issus d'une source comme le Projet Gutenberg en un format utilisable par les modèles de langage. Ce processus vise à éliminer le "boilerplate" spécifique au Projet Gutenberg et à normaliser le texte pour une analyse cohérente.

Le processus de nettoyage se déroule comme suit :

- **Suppression des en-têtes et pieds de page du Projet Gutenberg** : Les fichiers du Projet Gutenberg contiennent des sections standardisées (informations de licence, notes sur le projet, etc.) au début et à la fin du document. Ces sections sont identifiées par des motifs textuels spécifiques (par exemple, *** START OF THE PROJECT GUTENBERG EBOOK *** et *** END OF THE PROJECT GUTENBERG EBOOK ***) et sont retirées pour ne conserver que le contenu de l'oeuvre.
- **Élimination des informations de production et de transcription** : D'autres lignes de boilerplate, telles que les mentions de production, de transcription ou de préparation de l'Etext, sont également supprimées du corps du texte.
- **Normalisation des sauts de ligne et des espaces** : Les retours chariot (\r \n) sont convertis en simples sauts de ligne (\n). Les sauts de ligne uniques (qui représentent souvent des retours à la ligne dans un paragraphe) sont remplacés par des espaces pour former des paragraphes continus. Les multiples sauts de ligne consécutifs (qui indiquent des séparations de paragraphes) sont normalisés en un seul séparateur de paragraphe logique (ici, un simple espace après la conversion finale pour uniformiser le texte). Enfin, les espaces multiples sont réduits à un seul espace et le texte est débarrassé des espaces inutiles au début et à la fin.

À l'issue de cette étape, un texte propre, continu et uniforme est obtenu, prêt pour les étapes ultérieures de découpage et d'analyse par les LLM.

4.3 Traitement des données

Dans la conception du système de recommandation de livres, il a été choisi d'exploiter les grands modèles de langage (LLM) pour générer des résumés et identifier les thèmes principaux des ouvrages. Ces éléments servent ensuite de base à la création d'embeddings utilisés pour formuler les recommandations. Cette orientation repose sur des avantages significatifs par rapport aux approches classiques.

4.3.1 Intérêt de l'usage des LLM pour la synthèse et l'analyse thématique

Les méthodes traditionnelles de traitement du texte, telles que l'extraction de mots-clés, les scores TF-IDF ou les résumés extractifs, montrent rapidement leurs limites. Elles peinent à restituer la complexité des oeuvres littéraires. Par exemple, deux textes partageant un vocabulaire commun peuvent porter des messages très différents. Les résumés produits automatiquement par extraction de phrases tendent également à manquer de cohérence et n'offrent qu'une vision partielle du contenu.

À l'inverse, les grands modèles de langage permettent une compréhension plus fine du texte. En s'appuyant sur un entraînement massif, ces modèles parviennent à saisir les nuances de sens, les relations entre les idées et la structure globale du récit. Il devient alors possible de produire des résumés synthétiques, cohérents et adaptés, proches de ceux que rédigerait un humain. De même, les thèmes dégagés ne se limitent pas à des mots récurrents, mais reflètent des concepts sous-jacents comme le deuil, la quête d'identité ou la résilience, même lorsqu'ils ne sont pas exprimés explicitement dans le texte.

La cohérence narrative est également mieux assurée. Grâce à des mécanismes de maintien du contexte à long terme (notamment via le découpage hiérarchique), les résumés peuvent suivre le fil de l'histoire, même en présence d'une structure non linéaire ou complexe.

Pertinence du choix des résumés et des thèmes comme représentations textuelles

Le recours à ces deux types de sortie, (résumé et thèmes) est une décision pensée pour optimiser la qualité des recommandations.

Les livres constituant des documents particulièrement longs (voir table 5.2), une vectorisation directe de l'intégralité du texte s'avère souvent impraticable. La fenêtre contextuelle des modèles est fréquemment dépassée, et même lorsqu'elle ne l'est pas, le sens sémantique risque d'être dilué. En condensant le contenu à travers un résumé et en le catégorisant par des thèmes, il devient possible de produire des représentations textuelles à la fois compactes et informatives, idéales pour la génération d'embeddings.

Ces deux formats se révèlent par ailleurs complémentaires. Le résumé restitue les éléments narratifs essentiels : intrigue, personnages, structure. Les thèmes fournissent une grille de lecture plus conceptuelle, utile pour faire émerger des similarités d'ordre plus abstrait. Leur association permet une description riche et équilibrée de chaque oeuvre.

Sur le plan de la comparaison sémantique, ces représentations facilitent les rapprochements pertinents entre livres. Elles rendent possible la mise en relation d'ouvrages très différents en apparence, mais traitant de problématiques similaires. Il devient ainsi plus facile d'élargir le champ des recommandations tout en conservant une cohérence thématique forte.

Ces éléments textuels peuvent également être présentés tels quels à l'utilisateur. Un résumé permet de saisir rapidement l'univers du livre recommandé, tandis que les thèmes aident à en cerner les grands axes ou à explorer de nouveaux genres. Cette transparence améliore l'expérience utilisateur et renforce la confiance envers le système.

Enfin, le choix de ne pas imposer de liste de thèmes prédéfinie permet de s'affranchir des limites d'une taxonomie figée. Les modèles peuvent ainsi identifier des nuances spécifiques à chaque oeuvre ou faire émerger des thématiques nouvelles. Cela ouvre la voie à une recommandation plus fine, capable de s'adapter à des corpus évolutifs.

4.3.2 De la représentation textuelle aux recommandations vectorielles

Une fois les résumés et les thèmes générés, ceux-ci sont transformés en vecteurs via un encodeur sémantique. Cette vectorisation permet de positionner chaque livre dans un espace de représentation où les distances traduisent des proximités de sens. Il devient alors possible d'identifier des relations fines entre des oeuvres appartenant à des genres ou registres différents, mais portées par des thématiques ou des structures narratives communes.

Grâce à cette approche, la recommandation s'appuie non plus sur des similarités superficielles (vocabulaire, popularité, co-occurrence), mais sur une compréhension plus profonde du contenu littéraire. Cela permet d'accompagner l'utilisateur vers des découvertes plus pertinentes, parfois inattendues, mais toujours justifiées par une logique sémantique forte.

4.3.3 Stratégie de découpage du texte (chunking)

Plutôt que de segmenter les livres par chapitres, il a été opté pour une stratégie de découpage en *chunks* de taille fixe. Ce choix est motivé par plusieurs considérations techniques et pratiques :

- **Incohérence structurelle des chapitres :** Tous les livres, en particulier les oeuvres anciennes du Projet Gutenberg, ne possèdent pas une structure de chapitres uniforme ou clairement délimitée. Certains peuvent n'avoir aucune division explicite, d'autres des divisions irrégulières, rendant une extraction fiable par chapitre complexe et sujette aux erreurs.
- **Alignement avec la fenêtre contextuelle du LLM :** La taille des chapitres est variable et ne correspond pas nécessairement à la fenêtre contextuelle optimale des LLM. Un chapitre peut être trop court, diluant le contexte, ou trop long, dépassant la capacité de traitement du modèle en un seul appel. Le découpage en chunks de taille fixe permet de mieux contrôler la quantité d'informations soumises au LLM à chaque itération, assurant une utilisation efficace de sa fenêtre contextuelle.

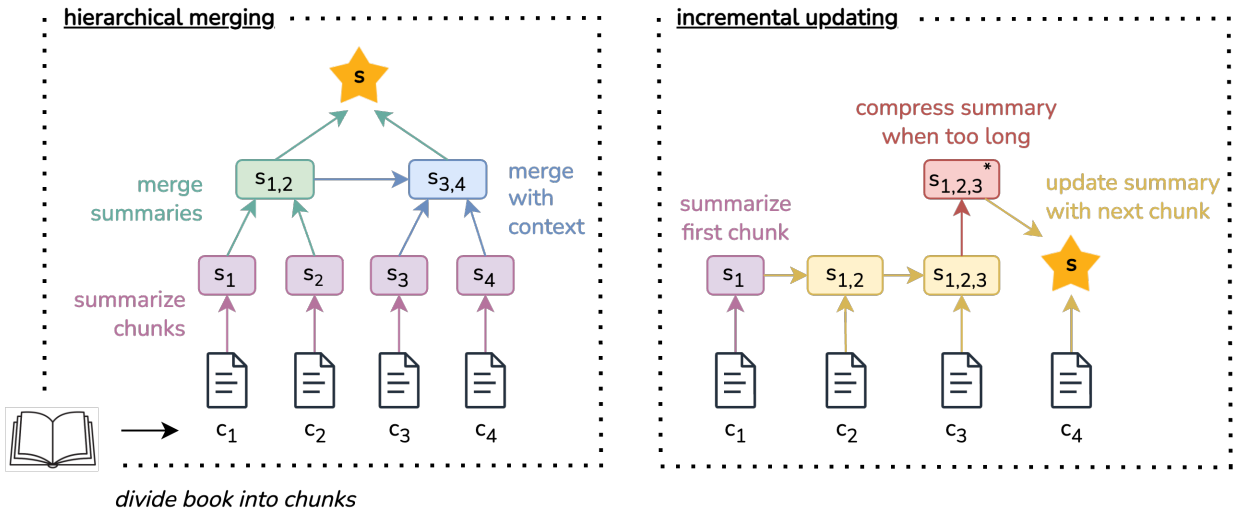


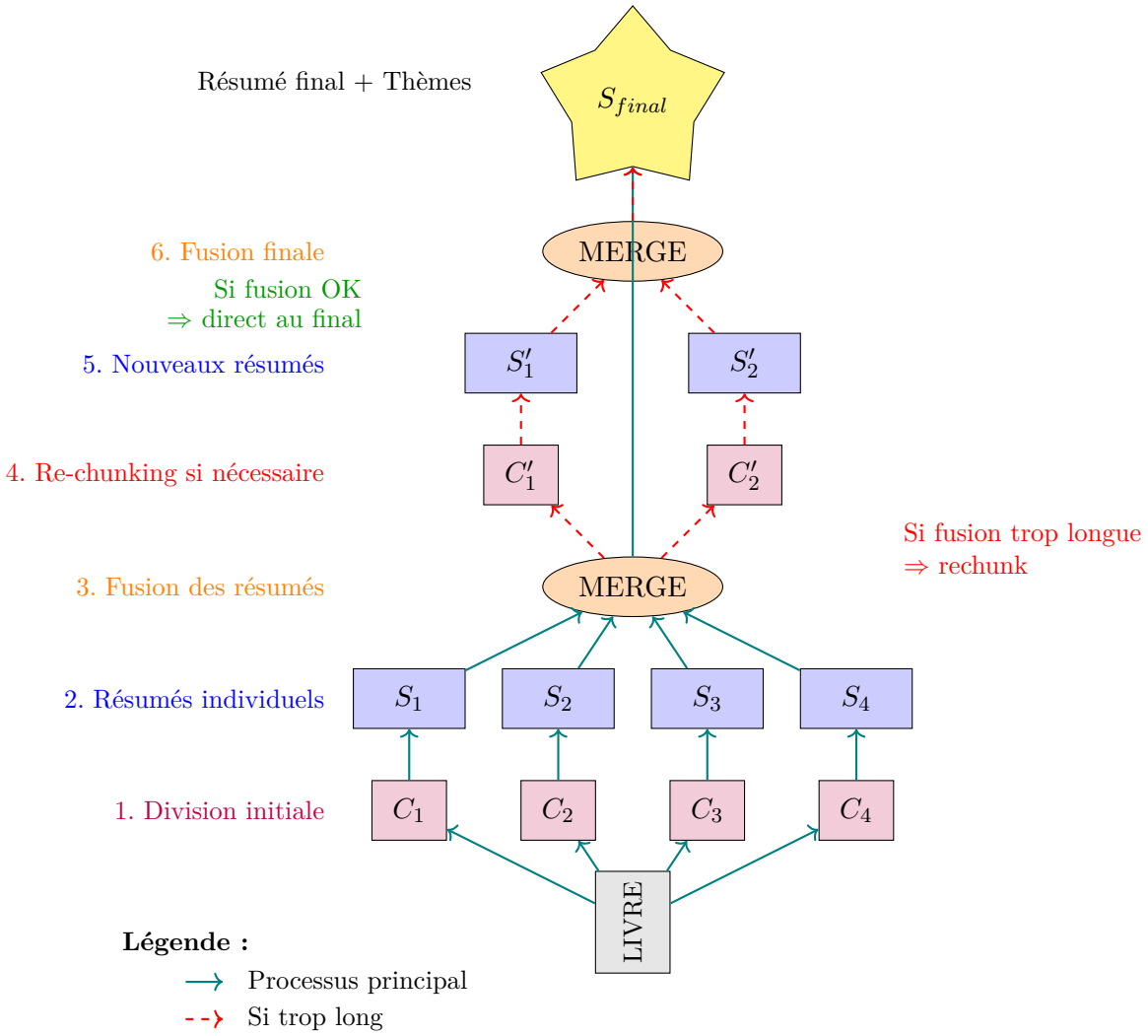
Figure 4.1 – *Deux méthodes de résumé : hiérarchique et incrémental (source : Chang et al. (2024) [9])*

— **Uniformité de l'entrée pour le modèle hiérarchique :** L'approche de résumé est hiérarchique : des résumés intermédiaires sont générés pour des segments, puis ces résumés sont combinés pour un résumé final. Pour ce processus, il est plus avantageux d'avoir des unités d'entrée de taille relativement constante pour le premier niveau de résumé. Cela garantit une charge de travail prévisible pour le LLM et une meilleure comparabilité des résumés intermédiaires.
— **Simplicité et robustesse :** Le découpage par taille fixe est une méthode simple et robuste qui fonctionne universellement sur tout type de texte, sans nécessiter d'analyse structurelle complexe. C'est une approche moins sujette aux échecs que les méthodes basées sur la détection de chapitres. La cohérence narrative est ensuite reconstruite par la phase de résumé hiérarchique, qui est spécifiquement conçue pour gérer la fusion de ces segments.

À l'issue de cette étape, chaque livre est représenté par une séquence de chunks de texte, prêts à être traités individuellement par le modèle de langage pour la génération de résumés intermédiaires.

4.3.4 Stratégie de résumé hiérarchique

La nature des LLM, avec leur fenêtre contextuelle limitée, constitue un défi majeur lors du traitement d'oeuvres littéraires complètes qui peuvent s'étendre sur des centaines de milliers de mots. Pour surmonter cette contrainte et garantir une analyse exhaustive sans perte de sens, nous avons implémenté une stratégie de résumé hiérarchique et récursif. Le choix d'utiliser un résumé hérarchique est motivé par les travaux effectués pour "BooookScore" de Chang et al. (2024) [9] qui ont trouvé que les résumé hiérarchiques donnaient une meilleure cohérence que ceux itératifs. La Figure 4.1 présente les deux types de génération de résumés évalués par Chang et al. (2024). L'approche utilisée dans ce projet est similaire à celle visible sous "hierarchical merging". Cette approche se décompose en plusieurs étapes, le schéma de l'approche utilisée dans ce projet est visible en Figure 4.3.4 :



— **Résumé des "chunks" initiaux** : Chaque livre est d'abord divisé en "chunks" de taille fixe. Le LLM est ensuite sollicité pour générer un résumé intermédiaire pour chaque "chunk" individuel. Cette première passe permet de condenser localement l'information tout en respectant la limite de contexte du modèle.

— **Combinaison et évaluation de la taille** : Les résumés intermédiaires de tous les "chunks" sont ensuite combinés en un seul texte. Avant de procéder à la génération du résumé final, la taille de ce texte combiné est évaluée en termes de tokens.

— **Résumé récursif en cas de dépassement** : Si le texte combiné des résumés intermédiaires dépasse la fenêtre contextuelle maximale du LLM, on réitère le processus en générant des chunks, cette fois à partir des livres intermédiaires. Cela crée un nouveau niveau de résumés, plus concis, qui condensent l'information des résumés précédents.

— **Génération du résumé et des thèmes finaux** : Une fois que l'ensemble des informations intermédiaires tient dans la fenêtre contextuelle, le LLM est utilisé une dernière fois pour générer le résumé final du livre et extraire ses thèmes principaux.

4.3.5 Conclusion sur les décisions de traitement des données

Les choix méthodologiques présentés visent à établir une base solide pour le développement du système de recommandation de livres par LLM.

La sélection du Projet Gutenberg comme source principale est motivée par l'accès au texte intégral des oeuvres, jugé essentiel pour une analyse approfondie par LLM, malgré une diversité de genres potentiellement plus limitée.

L'acquisition et le nettoyage des données devraient garantir un corpus textuel pur et uniforme, grâce à l'utilisation du format texte brut et à un processus rigoureux de suppression du "boilerplate" et de normalisation.

La stratégie de découpage en chunks de taille fixe est préférée à la segmentation par chapitres. Ce choix devrait assurer une robustesse face à l'hétérogénéité des structures de livres et un alignement optimal avec la fenêtre contextuelle des LLM, favorisant l'efficacité du résumé hiérarchique.

Enfin, l'exploitation des LLM pour générer des résumés abstraits et des thèmes dynamiques est au coeur de l'approche sémantique. Cette méthode est conçue pour surmonter les limites des techniques traditionnelles en capturant la richesse narrative et conceptuelle des livres, et devrait fournir une base solide pour la génération d'embeddings.

Ces décisions convergent vers l'objectif de transformer des textes bruts en représentations vectorielles riches, permettant une analyse sémantique profonde.

Chapitre 5

Implémentation de la collecte et du traitement des données

5.1 Téléchargement des livres du projet Gutenberg

Cette partie explique la partie du projet développée permettant le téléchargement et le nettoyage des livres provenant du Projet Gutenberg. Le code est disponible sur un repértoire Github public à l'adresse suivante : https://github.com/Sainane/books_processing

5.1.1 Objectif et périmètre de l'outil

Dans le cadre de ce projet, un module dédié a été implémenté pour télécharger automatiquement des ouvrages depuis le projet Gutenberg, en utilisant l'API publique fournie par Gutendex. [24] Ce module a pour but de récupérer les métadonnées des livres, d'en télécharger le contenu textuel brut (soit depuis les serveurs de Gutenberg, soit à partir de fichiers locaux en `.txt.gz` qui sont fournis par Gutenberg), de nettoyer ce contenu afin de le rendre exploitable pour traitement ultérieur et d'enregistrer les résultats structurés au format `JSON`.

5.1.2 Fonctionnalités principales

Le module est structuré autour de la classe abstraite `GutenbergDownloader` qui formalise les étapes essentielles du processus :

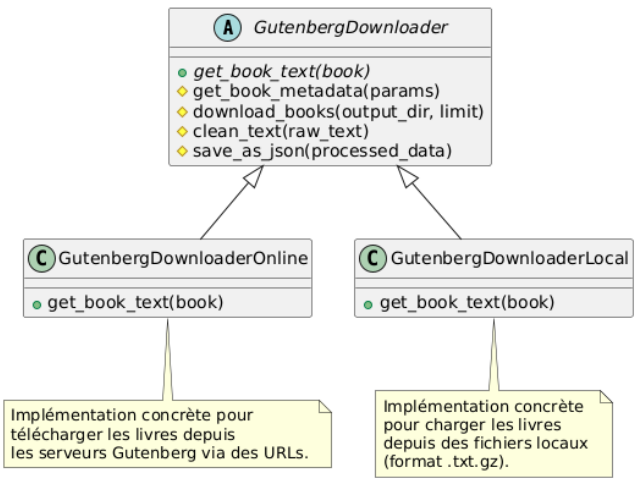


Figure 5.1 – *Diagramme de classe du GutenbergDownloader*

— Récupération des métadonnées d'un ou plusieurs ouvrages via l'API Gutendex en fonction de paramètres configurables (langue, mot-clé, IDs, etc.).
— Téléchargement du texte brut d'un livre selon son emplacement (en ligne ou local).
— Nettoyage du texte pour en extraire le contenu littéraire pertinent, en supprimant les en-têtes, pieds de page et autres artefacts spécifiques au formatage Gutenberg.
— Sérialisation du texte nettoyé ainsi que des métadonnées (titre, auteurs, identifiant, etc.) dans un fichier au format `JSON`.

5.1.3 Architecture logicielle

Classes et héritage Le coeur logiciel repose sur le schéma suivant, visible en Figure 5.1 :

— `GutenbergDownloader` : classe de base abstraite définissant l'interface et le comportement général.
— `GutenbergDownloaderOnline` : classe concrète permettant de télécharger les livres depuis les URLs fournies par l'API (format texte brut UTF-8 ou US-ASCII).
— `GutenbergDownloaderLocal` : variante locale qui récupère les livres au format compressé `.txt.gz` depuis une arborescence définie.

Structures de données Le module s'appuie sur des modèles `pydantic` pour définir les schémas attendus des objets :

— `Book` : représente un livre unique avec toutes ses métadonnées.
— `Person` : encode les informations sur les auteurs ou traducteurs.
— `GutendexBookListResponse` : modèle correspondant à une réponse paginée de l'API Gutendex.
— `ProcessorInput` : objet englobant le texte nettoyé d'un livre ainsi que ses métadonnées, prêt à être sérialisé ou traité.

5.1.4 Script principal : `download_books.py`

Un script principal est mis à disposition qui permet de choisir entre les différents modèles de Downloader et utilise le fichier de configuration permettant de choisir les paramètres de recherches des livres. Un schéma de la pipeline du script principal pour le mode online est visible en Figure 5.2.

5.1.5 Détails du processus technique

Chargement et filtrage des métadonnées

Les métadonnées des livres sont récupérées grâce à des appels successifs à l'API REST de Gutendex. Le processus est paginé et prend en compte un ensemble de paramètres personnalisables définis dans un fichier `YAML`, tels que :

— `ids` : liste des identifiants (Project Gutenberg ID) des ouvrages à télécharger.
— `languages`, `search`, `topic`, etc. : autres critères liés au contenu recherché.

L'URL de chaque page est construite dynamiquement et les livres récupérés sont convertis en instances de modèles `Book`. Il est possible d'hoster localement l'API Gutendex, les informations pour le faire sont disponible sur le repository Github de l'auteur. [28]

Récupération et décodage du texte brut

Le téléchargement du texte diffère selon le contexte :

— Dans le cas **en ligne**, le module tente d'accéder à l'URL du format `text/plain` le plus pertinent (UTF-8 ou US-ASCII). Un décodage sécuritaire est réalisé avec prise en charge d'erreurs `Unicode`.
— Dans le cas **hors-ligne**, le texte est extrait à partir d'un fichier `.gz` local. La décompression s'effectue en mode texte et en UTF-8. La structure du dossier local doit respecter la structure du dossier contenant les livres disponible sur le site de Gutenberg [46]

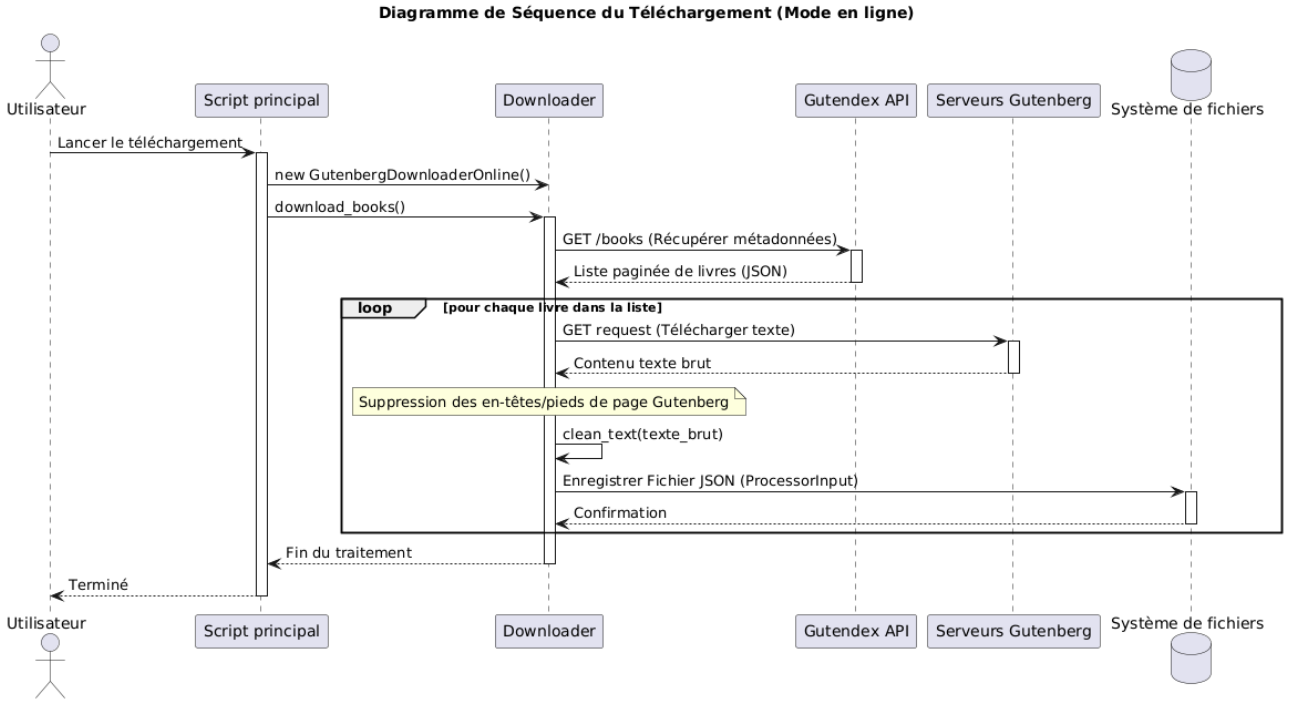


Figure 5.2 – *Pipeline du script principal de téléchargement de livres de Gutenberg en mode online*

Nettoyage du contenu littéraire

Le texte brut d'un livre contient généralement des sections parasites (métadonnées inutiles, disclaimer Gutenberg, mentions légales, etc.). Un algorithme de *prétraitement* a été mis en place pour :

- Supprimer les en-têtes et pieds-de-page propres à Gutenberg.
- Éliminer les lignes de type bibliographique ou commentaires sur l'origine du fichier.
- Remplacer les sauts de ligne non significatifs (line wrapping) par des espaces.
- Normaliser les paragraphes en supprimant les multiples sauts de ligne.

Ce prétraitement vise à produire un texte fluide, structuré par paragraphes continus, et prêt pour un éventuel pipeline de traitement via LLM.

Sérialisation et archivage

Une fois nettoyé, le texte est encapsulé dans une structure `ProcessorInput` décrivant le livre (titre, auteurs, identifiant) et son contenu. Il est ensuite stocké sous forme de fichier `.json`, un par livre, dans un répertoire de sortie configurable.

Ces fichiers forment une base exploitable pour les étapes suivantes du pipeline de traitement. Ce format de sortie est décrit en section 5.2.7

5.2 Module de traitement des données

La section suivante parle du module de traitement des données des livres développé. Le code est disponible publiquement sur Github sur le même répertoire que la partie permettant le téléchargement des données de Gutenberg à l'adresse suivante : https://github.com/Sainane/books_processing

5.2.1 Objectif et périmètre de l'outil

L'objectif principal de ce système est de traiter efficacement le contenu de livres, pour en extraire des résumés concis et des informations thématiques pertinentes. La conception met l'accent sur une

séparation claire des préoccupations, permettant une intégration facile de nouvelles techniques de résumé ou de différents modèles linguistiques sans modifier la logique de traitement de base.

5.2.2 Architecture générale

L'architecture du système est structurée autour de plusieurs classes Python interconnectées, comme le montre le diagramme de classe en 5.3 L'architecture a été pensée selon un principe de *modularité* et de *séparation des préoccupations*. Cette approche garantit une grande maintenabilité et permet des extensions futures sans avoir à remanier l'ensemble du code.

Les principaux composants, sont :

- **Schémas de données** : Des modèles Pydantic qui définissent la structure des objets de données.
- **Génération de chunk (Chunker)** : Module responsable de la division des textes longs.
- **Abstraction des modèles de langage (LanguageModel)** : Une interface unifiée pour communiquer avec diverses familles de LLM.
- **Génération de résumés (Summarizer)** : La module principal qui organise le processus de génération résumés et thèmes en plusieurs étapes.
- **Processeur de livre (BookProcessor)** : Le composant qui assemble les différentes briques pour traiter un livre de A à Z.
- **Script principal (`process_books.py`)** : Le point d'entrée pour l'utilisateur, permettant de lancer et de configurer la pipeline.

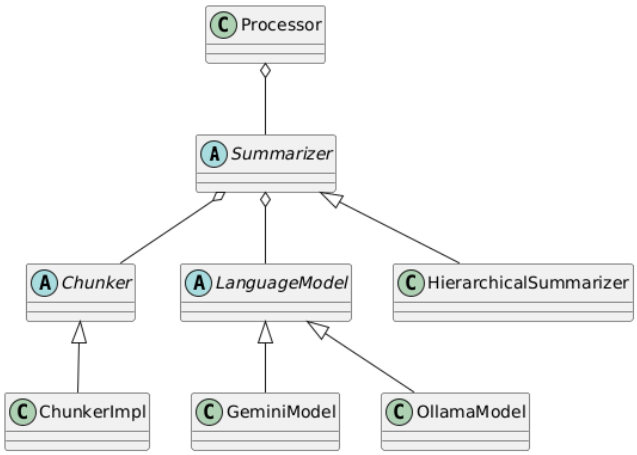


Figure 5.3 – *Diagramme de classe*

La logique de résumé est encapsulée derrière une interface abstraite `Summarizer`, ce qui permet d'échanger différentes stratégies de résumé. L'implémentation concrète actuelle, `HierarchicalSummarizer`, effectue un résumé en plusieurs étapes en utilisant le `LanguageModel` et le `Chunker`. Le `LanguageModel` lui-même est une interface abstraite, permettant au système d'interagir avec divers fournisseurs de LLM (par exemple, Gemini et Ollama). L'intégrité et la structure des données sont maintenues grâce à des modèles Pydantic tels que `ProcessorInput`, `ProcessorOutput` et `SummarizerOutput`.

Le résultat d'un livre traité par cet outil est disponible en annexe B

5.2.3 Summarizer

Le `Summarizer` est défini comme une Classe de Base Abstraite (ABC). Cette architecture permet de définit un contrat clair pour toute classe souhaitant agir comme un Summarizer. Tout Summarizer concret doit implémenter la méthode abstraite `summarize`. Le `BookProcessor` interagit avec l'interface `Summarizer`, et non avec une implémentation spécifique. Cela signifie que différents algorithmes de résumé (par exemple, extractif, abstrait, hiérarchique, basé sur des mots-clés) peuvent être développés et utilisés de manière interchangeable sans modifier le `BookProcessor`.

HierarchicalSummarizer

`HierarchicalSummarizer` est une implémentation concrète de `Summarizer`. Comme son nom l'indique, elle utilise une approche hiérarchique pour le résumé, particulièrement utile pour les textes longs :

1. Elle utilise d'abord un `Chunker` pour diviser le texte d'entrée en morceaux plus petits et gérables.
2. Pour chaque morceau, elle génère un *résumé intermédiaire* en sollicitant un `LanguageModel`.
3. Ces résumés intermédiaires sont ensuite combinés
4. Sis résumés combinés sont encore trop longs pour la fenêtre de contexte du modèle, le processus de résumé par chunk est réitéré sur la combinaison des résumés générés
5. Enfin, le `LanguageModel` est sollicité à nouveau pour produire un *résumé final* et extraire les thèmes.
6. Elle s'appuie sur un fichier yaml pour charger les prompts, rendant les prompts configurables

5.2.4 LanguageModel

Comme `Summarizer`, `LanguageModel` est une ABC qui définit l'interface pour interagir avec divers grands modèles de langage. Le reste du système (par exemple, `HierarchicalSummarizer`) n'a pas besoin de connaître les spécificités de l'interaction avec Gemini ou Ollama. Il sait seulement comment appeler la méthode `generate_content`. Cela permet de plus de facilement rajouter un autre modèle de langage.

Implémentations concrètes de `LanguageModel`

Le système comprend actuellement deux implémentations concrètes de `LanguageModel` :

- **GeminiModel :** Interagit avec l'API Gemini de Google. Il prend en charge la sortie structurée via `response_schema` et `response_mime_type` pour les modèles qui le prennent en charge nativement [22].
- **OllamaModel :** Conçu pour les instances Ollama locales ou auto-hébergées. Il tente également d'exploiter les capacités de sortie structurée si le modèle le prend en charge. [40]

Chaque implémentation gère les spécificités de son API respective, en abstrayant ces détails de la logique du Summarizer.

5.2.5 Chunker

Le `Chunker` est une classe abstraite chargée de définir comment le texte doit être divisé en segments plus petits.

ChunkerImpl

ChunkerImpl est l'implémentation concrète qui divise le texte en phrases, puis regroupe ces phrases en morceaux basés sur un nombre maximal de tokens. Cela évite d'envoyer des entrées excessivement longues au modèle linguistique, ce qui pourrait dépasser les limites de tokens ou augmenter le temps/-coût de traitement. Le segmentation en phrase est implémenté grâce à la libraire `nltk` de python. [39]

- **ChunkerImpl :**

Cette conception abstraite permet d'introduire facilement différentes stratégies de découpage (par exemple, des morceaux de taille fixe, basés sur des paragraphes, découpage sémantique).

5.2.6 BookProcessor

La classe `BookProcessor` est la classe encapsulant la gestion de l'input et de l'output. Sa méthode `process` prend un objet `ProcessorInput` (contenant le texte du livre, le titre, les auteurs, etc.), délègue la tâche de résumé à son instance `Summarizer` injectée, puis construit un objet `ProcessorOutput` en utilisant les résultats. Cette classe est intentionnellement légère, se concentrant uniquement sur le flux de travail de haut niveau et la gestion des formats. Le fonctionnement de la éthode process de la classe `BookProcessor` peut être visualisé sur la Figure 5.4

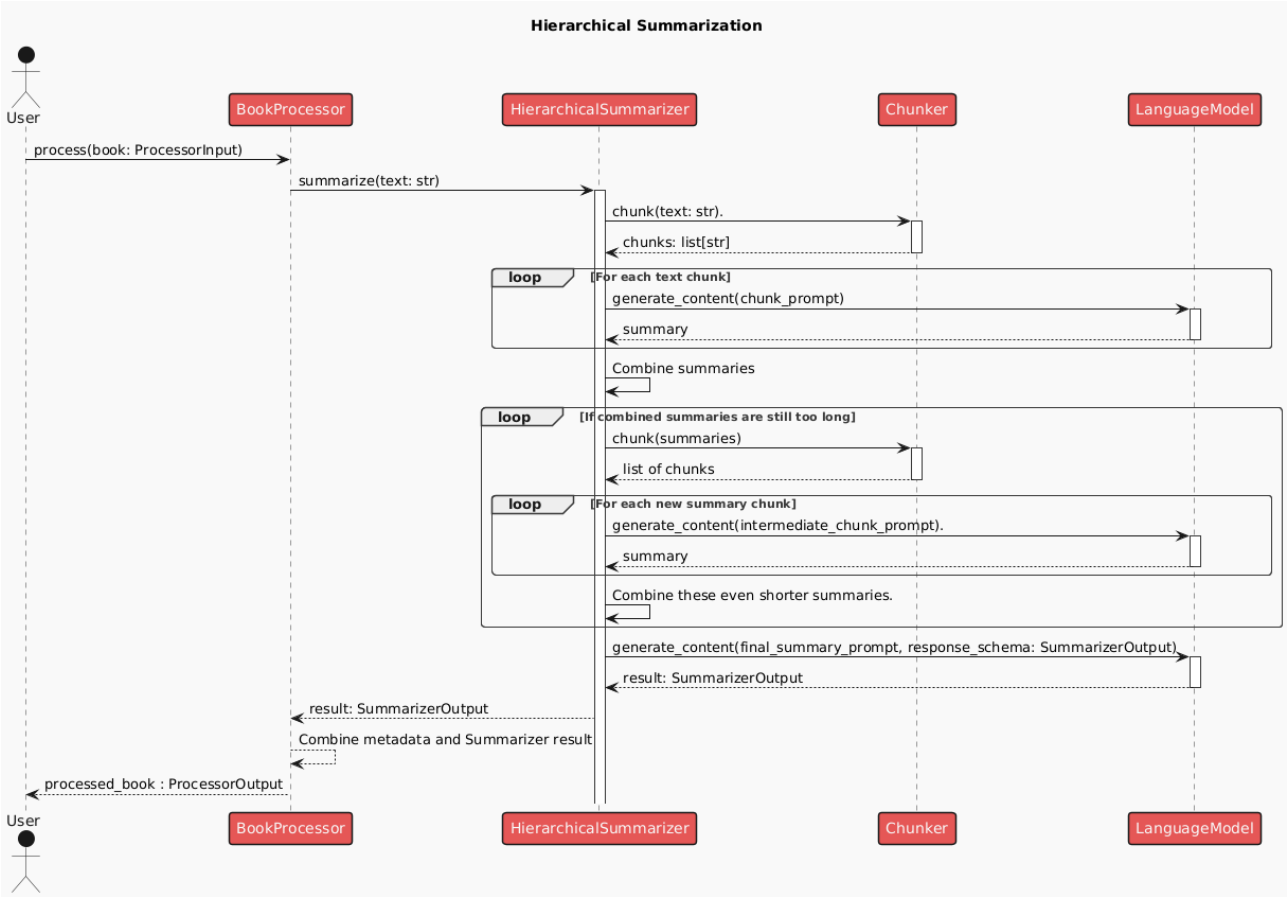


Figure 5.4 – *Génération de résumé hiérarchique*

5.2.7 Modélisation des données avec `Pydantic`

Pour assurer une structure de données robuste et validée tout au long de la pipeline de traitement des livres, nous utilisons la bibliothèque `Pydantic`. Cette section détaille les schémas de données clés qui définissent les formats d'entrée et de sortie de nos opérations.

Schéma d'entrée de `BookProcessor` : `ProcessorInput`

Le modèle `ProcessorInput` définit la structure des données brutes d'un livre, telles qu'elles sont lues depuis un fichier JSON après la phase de téléchargement et de nettoyage. Il encapsule le texte du livre ainsi que ses métadonnées essentielles.

Un exemple de fichier JSON conforme au schéma `ProcessorInput` est présenté dans le listing 5.1 Ce fichier est le point de départ du traitement pour chaque livre.

Schéma de sortie du `Summarizer` : `SummarizerOutput`

Le modèle `SummarizerOutput` représente la sortie structurée attendue du modèle de Langage (LLM) après l'opération de résumé hiérarchique et d'extraction de thèmes. Le LLM, si compatible est configuré pour générer directement un JSON conforme à ce schéma. Si ce n'est pas le cas, un exemple est rajouté au prompt. Un exemple de la sortie JSON générée par le LLM, respectant le schéma `SummarizerOutput`, est illustré dans le listing 5.2

Schéma de sortie du `BookProcessor` : `ProcessorOutput`

Enfin, le modèle `ProcessorOutput` agrège les informations du livre original avec les résultats du résumé et de l'extraction de thèmes. C'est le format final des données traitées, destiné à être sauvegardé dans un nouveau fichier JSON.

```
{
  "text": "Le texte nettoyé du livre commence ici. Il peut être très long...",
  "title": "Un grand classique",
  "authors": [
    {
      "name": "Jane Doe",
      "birth_year": 1800,
      "death_year": 1890
    }
  ],
  "gutenberg_id": 12345
}
```

Listing 5.1 – *Exemple de fichier JSON d'entrée pour* `ProcessorInput`

```
{
  "summary": "Ce résumé concis décrit les points clés du livre, abordant les personnages principaux,
  ↪  l'intrigue et les résolutions. Il met en lumière les aspects les plus pertinents du récit
  ↪  original.",
  "themes": [
    "aventure",
    "découverte",
    "amitié",
    "conflit intérieur"
  ]
}
```

Listing 5.2 – *Définition du modèle Pydantic* `SummarizerOutput`

Un exemple de fichier JSON de sortie, conforme au schéma `ProcessorOutput`, est présenté dans le listing 5.3

5.2.8 Avantages de l'utilisation des classes abstraites

L'utilisation omniprésente des classes abstraites (`Summarizer`, `LanguageModel`, `Chunker`) tout au long de l'architecture offre plusieurs avantages significatifs :

- **Couplage faible :** Les composants sont faiblement couplés, ce qui signifie que les modifications apportées à une implémentation concrète (par exemple, le passage de Gemini à Ollama) ont un impact minimal sur les autres parties du système.
- **Haute cohésion :** Chaque classe ou module a une responsabilité unique et bien définie.
- **Extensibilité :** Le système est hautement extensible. De nouveaux algorithmes de résumé, fournisseurs de LLM ou stratégies de découpage peuvent être ajoutés en implémentant simplement l'interface abstraite respective sans modifier le code existant et testé. Cela respecte le Principe Ouvert/Fermé (OCP).
- **Maintenabilité :** La conception modulaire rend la base de code plus facile à comprendre, à déboguer et à maintenir. Les problèmes peuvent souvent être isolés à des implémentations spécifiques.

5.2.9 Script principal

Le script `process_books.py` fourni un script de traitement en bulk de fichier locaux et utilise le module `argparse` de Python pour créer une interface en ligne de commande complète et documentée. Le script instancie dynamiquement les objets nécessaires en fonction de ces arguments et effectue le process sur chaque fichier trouvé. Le flux principal du programme peut-être visualisé en figure 5.5

```
{
  "summary": "Ce résumé concis décrit les points clés du livre, abordant les personnages principaux,
  ↪  l'intrigue et les résolutions. Il met en lumière les aspects les plus pertinents du récit
  ↪  original.",
  "title": "Un Grand Classique",
  "authors": [
    {
      "name": "Jane Doe",
      "birth_year": 1800,
      "death_year": 1890
    }
  ],
  "themes": [
    "aventure",
    "découverte",
    "amitié",
    "conflit intérieur"
  ],
  "gutenberg_id": 12345
}
```


Listing 5.3 – *Exemple de fichier JSON de sortie pour* `ProcessorOutput`

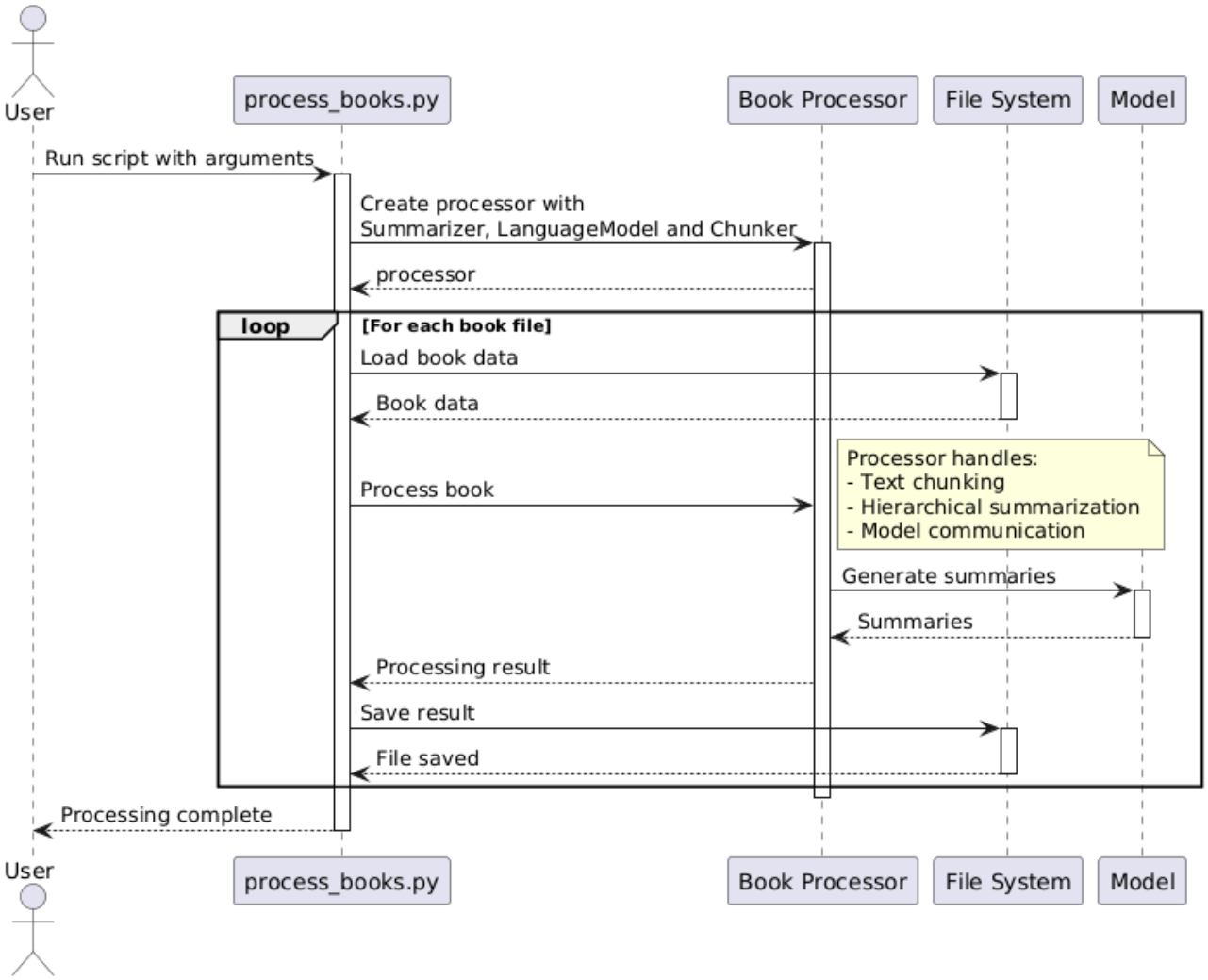


Figure 5.5 – *Pipeline de processing du script process_books.py*

5.3 Traitement des livres effectué

Cette section détaille l'application pratique du module de traitement des livres, tel que décrit précédemment, pour générer des résumés et extraire des thèmes à partir d'un corpus de livres. Nous décrivons les paramètres utilisés, le modèle de langage sélectionné et le processus d'exécution sur le dataset choisi.

5.3.1 Corpus de livres utilisé

Pour cette phase de traitement, nous avons utilisé un sous-ensemble du catalogue Project Gutenberg, spécifiquement les auteurs du 19^{e} siècle, disponible sur Kaggle [73]. Ce dataset a été choisi pour sa richesse en contenu textuel et le pré-filtrage d'oeuvre de fiction.. Au total, **5060 livres** ont été traités via la pipeline d'ingestion et de nettoyage, puis soumis au processus de résumé.

5.3.2 Configuration du traitement

Le traitement des livres a été effectué en utilisant le script `process_books.py`, qui qui définit les différentes classes nécessaires au traitement et s'occupe de récupérer tous les fichiers à traiter d'un dossier et d'enregistrer la sortie dans un autre fichier déterminé. La configuration qui a été choisie pour le traitement est récapitulée dans le tableau 5.1.

Table 5.1 – *Paramètres de configuration utilisés pour le traitement des livres.*

Paramètre	**Valeur / Description**
Modèle de langage (LLM)	`gemini-2.5-flash-lite-preview-06-17`
Type de modèle	`gemini`
Taille des chunks (`-chunk-size`)	28000 caractères
Longueur cible du résumé (`-summary-length`)	800 mots

5.3.3 Détails des prompts utilisés

La qualité des résumés et des thèmes générés par le LLM dépend fortement de la formulation des prompts. Pour le résumé hiérarchique, deux prompts principaux ont été utilisés, configurés dans le fichier `prompts/prompts.yaml`. Ces prompts ont été conçus pour guider le modèle à travers les étapes de résumé intermédiaire et de synthèse finale, tout en assurant l'extraction d'informations clés et un format de sortie cohérent.

Prompt pour le résumé de chunk (`chunk_summary`)

Ce prompt 5.4 est appliqué individuellement à chaque segment (*chunk*) du livre. Il vise à obtenir un résumé concis de la partie du texte fournie, en incluant les informations vitales et en gérant les introductions de nouveaux éléments. L'objectif est de préparer des résumés intermédiaires qui pourront être fusionnés de manière cohérente par la suite.

Prompt pour les résumés de résumés

Lorsque la combinaison de tous les résumés est trop importante pour le contexte du modèle de langage, les résumés sont de nouveaux découpés en chunk et résumés par chunk. Pour refléter le fait que dans ce cas on demande au LLM de combiner des résumés, le prompt utilisé est légèrement différent. Le prompt est visible en 5.5

Prompt pour le résumé final et l'extraction de thèmes (`final_summary`)

Après avoir obtenu des résumés intermédiaires pour tous les chunks, ce prompt est utilisé pour synthétiser ces résumés intermédiaires en un résumé final cohérent et pour extraire les thèmes globaux

```
chunk_summary: |
  Below is a part of a story:

  ---

  {chunk}

  ---

  We are creating one comprehensive summary for the story by recursively merging summaries of its
  ↪  chunks. Now, write a summary for the excerpt provided above, make sure to include vital
  ↪  information related to key events, backgrounds, settings, characters, their objectives, and
  ↪  motivations. You must briefly introduce characters, places, and other major elements if they are
  ↪  being mentioned for the first time in the summary. The story may feature non-linear narratives,
  ↪  flashbacks, switches between alternate worlds or viewpoints, etc. Therefore, you should organize
  ↪  the summary so it presents a consistent and chronological narrative. Despite this recursive
  ↪  merging process, you need to create a summary that seems as though it is written in one go.

  Summary:
```

Listing 5.4 – *Prompt pour le résumé de chunk (`intermediate_summary`)*

```
intermediate_summary: |
  Below are several intermediate summaries of parts of a story:

  ---

  {chunk}

  ---

  We are creating one comprehensive summary by recursively merging summaries of its parts. Combine the
  ↪  summaries provided above into a single, more concise summary.

  Ensure the new summary preserves all vital information from the parts, including key events,
  ↪  characters, and settings. The summary should be organized to present a consistent and
  ↪  chronological narrative. Despite this recursive merging process, you need to create a summary
  ↪  that seems as though it is written in one go.

  Combined Summary:
```

Listing 5.5 – *Prompt pour les résumés de résumés*

du livre. Il est crucial que ce prompt guide le LLM vers une sortie structurée conforme au schéma `SummarizerOutput`.

Note : Le formatage du JSON dans le prompt est essentiel pour guider le LLM vers la sortie structurée souhaitée, en conjonction et d'autant plus efficaces avec un modèle étant compatible avec les sorties structurées, dans le cas contraire, le format attendu est rajouté au prompt.

5.3.4 Processus d'exécution

Le script `process_books.py` parcourt le répertoire d'entrée, charge chaque fichier JSON de livre (conforme à `ProcessorInput`), le valide avec Pydantic, le soumet au `BookProcessor`, et sauvegarde le résultat (conforme à `ProcessorOutput`) dans le répertoire de sortie spécifié. Le processus a été conçu pour ignorer les fichiers déjà traités dans le répertoire de sortie afin d'éviter des traitements redondants. Ce processus a permis de générer des résumés et des thèmes pour les 5060 livres du corpus, créant ainsi une base de données riche pour le système de recommandation.

```
final_summary: |
  Below are several intermediate summaries of parts of a story:

  ---

  {summaries}

  ---

  Combine these summaries into one comprehensive summary of the entire story. Ensure vital information
  ↪ related to key events, backgrounds, settings, characters, their objectives, and motivations are
  ↪ included. Briefly introduce characters, places, and other major elements if they are being
  ↪ mentioned for the first time in the overall summary. Organize the summary to present a
  ↪ consistent and chronological narrative, even if the original story was non-linear. The final
  ↪ summary must appear as though it was written in one go and must be within 800 words.

  After you write the summary, provide a list of themes or genres like : ["coming of age", "betrayal",
  ↪ "friendship", "supernatural"]).

  Output:
```

Listing 5.6 – *Prompt pour le résumé final et la génération de thèmes*

5.3.5 Justification des choix de configuration

Les choix architecturaux et de configuration pour le traitement des livres ont été guidés par des considérations à la fois techniques et pratiques, visant à optimiser la performance, la faisabilité et le coût du projet dans le contexte des ressources disponibles.

Sélection du modèle de langage

Le choix d'utiliser l'API Gemini (spécifiquement `gemini-2.5-flash-lite-preview-06-17`) plutôt qu'un LLM auto-hébergé a été une décision motivée par deux facteurs principaux :

- **Contraintes matérielles et temporelles :** Déployer et faire tourner des modèles de langage de grande taille (LLM) en local requiert une puissance de calcul significative, incluant des cartes graphiques (GPU) avec une mémoire vidéo très importante (souvent 24 Go de VRAM ou plus pour des modèles performants). Les ressources matérielles étaient limitées et ne permettaient pas de supporter efficacement l'inférence de LLM de la taille et des capacités requises pour des tâches de résumé et d'extraction de thèmes sur un corpus aussi vaste. L'utilisation d'une API cloud comme Gemini a éliminé ce besoin en hardware coûteux et complexe à gérer. Avec le GPU le plus puissant mis a disposition par l'école, par exemple, la génération d'un résumé prenait en moyenne cinq minutes avec le modèle `qwen3:0.6b` qui est un modèle déjà petit et largement moins efficace de ce qu'il est possible d'obtenir en ligne. Pour 5'060 livres, le temps de traitement total aurait été de dix-sept jours consécutifs.
- **Contrainte de coût et d'efficacité** En optant pour une solution basée sur le Google Cloud, il a été possible d'utiliser un modèle relativement conséquent et rapide et d'utiliser des crédits fournis par Google pour un essai ce qui a permis de n'engager aucun frais.

Bien que les LLM locaux offrent des avantages en termes de confidentialité des données et de contrôle total, ces considérations étaient secondaires par rapport à la faisabilité technique et budgétaire pour ce projet de bachelor. Néanmoins, le code permettant le traitement des livres est parfaitement compatible avec l'utilisation de Ollama et la classe permettant son utilisation est déjà implémentée.

Détermination de la taille des chunks

La taille des chunks, fixée à *28'000* caractères, représente un choix significatif qui impacte directement la performance et la qualité des résumés. Cette valeur a été déterminée après plusieurs expérimentations et justifications :

- **Maximisation du contexte par appel LLM :** Les LLM ont une fenêtre contextuelle limitée, c'est-à-dire la quantité maximale de texte qu'ils peuvent traiter en une seule fois. Le choix de 28'000 caractères vise à maximiser la quantité d'informations contenues dans chaque chunk, permettant ainsi au modèle d'avoir un contexte aussi riche que possible pour générer un résumé cohérent et pertinent pour cette partie du livre. Un contexte plus large réduit le risque de perdre des informations cruciales ou des relations entre différentes parties d'un même segment.
- **Réduction du nombre d'appels API et des coûts :** En utilisant des chunks plus grands, le nombre total d'appels à l'API Gemini est **considérablement réduit**. Chaque appel à une API cloud a un coût associé. En minimisant le nombre d'appels tout en fournissant suffisamment de contexte à chaque fois, nous avons optimisé l'efficacité économique du traitement, ce qui était essentiel étant donné notre budget limité pour l'utilisation du LLM.
- **Gestion de la complexité hiérarchique :** Pour des livres longs, un découpage en très petits chunks peut entraîner un très grand nombre de résumés intermédiaires, rendant la phase de fusion finale plus complexe pour le LLM et potentiellement plus sujette à la perte de cohérence globale. Des chunks plus grands signifient moins de résumés intermédiaires à fusionner, ce qui peut **améliorer la cohérence du résumé final** et simplifier la tâche du LLM dans la phase de synthèse.
- **Compromis entre la qualité et la latence :** Bien que des chunks excessivement grands puissent potentiellement submerger le modèle ou augmenter la latence des réponses, la taille de 28'000 caractères a été trouvée comme un bon compromis pour le modèle Gemini utilisé. Elle permettait d'équilibrer la richesse du contexte avec une latence acceptable et une qualité de résumé satisfaisante pour les besoins du projet.

Ces choix techniques reflètent une approche pragmatique pour construire un système fonctionnel et performant avec les ressources allouées, tout en maximisant la qualité des résultats obtenus.

5.3.6 Statistiques sur les livres et temps de traitement

La diversité des livres du corpus, a un impact direct sur le processus de traitement. Alors que certains livres sont relativement courts, d'autres présentent des volumes de texte considérables, comme en témoignent les valeurs minimale et maximale des tokens visible en table 5.2 Cette variabilité implique que le temps de génération nécessaire pour traiter chaque livre est également très hétérogène. Les ouvrages plus longs, nécessitant un découpage en un plus grand nombre de chunks et, par conséquent, davantage d'appels à l'API du modèle de langage, ont logiquement exigé des temps de traitement plus importants. La répartition du nombre de token par livre se trouve en figure 5.6

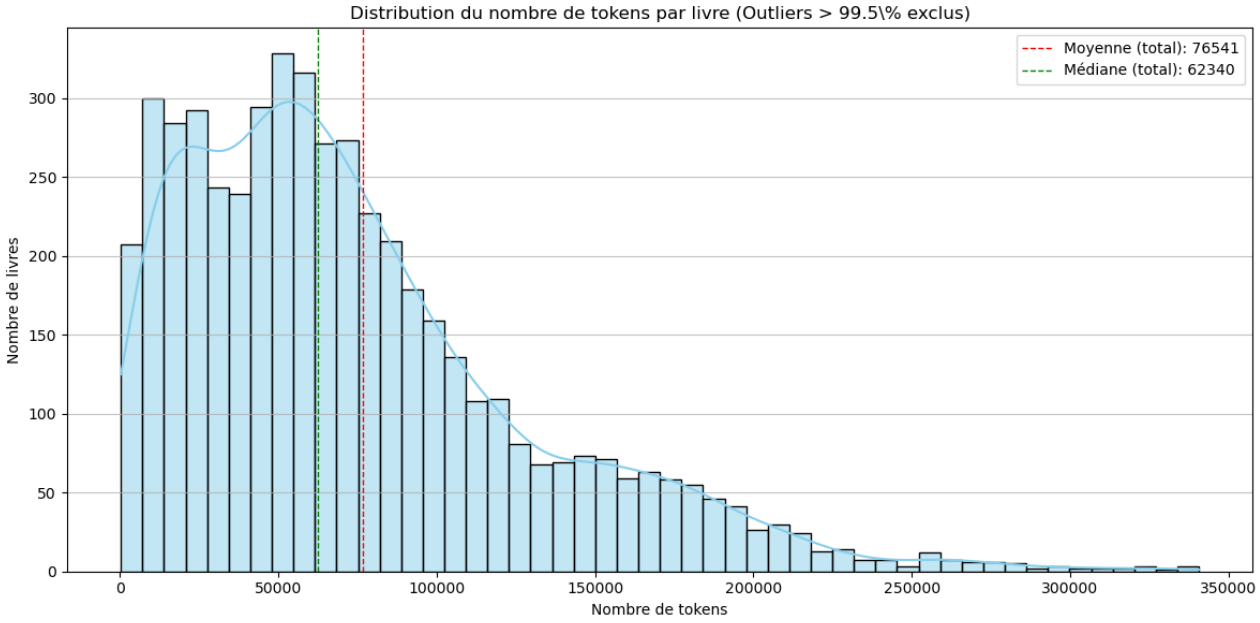


Figure 5.6 – *Distribution du nombre de tokens par livre*

Table 5.2 – *Statistiques de distribution du nombre de tokens par livre.*

Statistique	Valeur
Nombre total de livres analysés	5060
Valeur minimale de tokens	304
Valeur maximale de tokens	2588097
Moyenne de tokens	76541.42
Médiane de tokens	62339.5

Le temps de traitement moyen d'un livre avec les paramètres utilisé est de 40 secondes, ce qui correspond à un peu plus de deux jours de processing consécutifs pour les 5060 livres traités.

Chapitre 6

Implémentation de l'application de recommandation

Après avoir détaillé les phases de sélection, d'acquisition et de traitement sémantique du corpus de livres au chapitre précédent, ce chapitre est dédié à la présentation de l'architecture technique complète du système de recommandation. Il s'agit d'une application full-stack moderne et conteneurisée, conçue pour offrir une expérience utilisateur fluide et performante.

Avant de décrire chaque service, la figure 6.1 ci-dessous schématise le pipeline de données de bout en bout. Ce schéma illustre le parcours complet d'un livre : depuis sa source brute au sein du Projet Gutenberg, en passant par son analyse sémantique par un LLM pour générer des résumés et des thèmes, jusqu'à l'envoi dans l'application principale qui gérera la génération des embeddings à l'aide du worker de pgai. Les détails de fonctionnement du `pgai_worker` sont détaillés plus bas. L'architecture repose sur une séparation claire des responsabilités entre le backend et le frontend. Le backend, développé en Python avec le framework `FastAPI` et une couche d'abstraction de base de données `SQLAlchemy` asynchrone, interagit avec une base de données `PostgreSQL` optimisée par `TimescaleDB` pour la gestion efficace des informations des livres et des embeddings. Le frontend, quant à lui, est bâti avec `Vue.js` et le framework `Vuetify`, offrant une interface utilisateur moderne et réactive. L'ensemble de ces composants est orchestré via `Docker Compose`, assurant un déploiement et une gestion simplifiée. Le code de cette application est disponible sur Github publiquement à l'adresse suivante : https://github.com/Sainane/book_recommendation.git

6.1 Architecture

Le projet consiste en une application de recommandation moderne construite sur une architecture microservices conteneurisée avec Docker. Le système exploite les capacités des modèles de langage (LLM) pour générer des embeddings vectoriels et fournir des recommandations intelligentes aux utilisateurs.

L'architecture, illustrée dans la Figure 6.2, repose sur cinq composants principaux déployés dans des conteneurs Docker distincts : un service backend développé avec FastAPI pour l'API REST, une interface utilisateur frontend basée sur Vue.js/Node.js, une base de données TimescaleDB (PostgreSQL 17) optimisée pour la gestion des embeddings, un service Ollama hébergeant les modèles de langage, et un worker pgai responsable de la vectorisation automatique des données.

Le worker pgai orchestre le processus d'embedding en interrogeant périodiquement la base de données pour détecter les nouveaux champs nécessitant une vectorisation, en sollicitant le service Ollama pour générer les embeddings correspondants, puis en mettant à jour la base de données avec ces nouvelles représentations vectorielles. Cette approche permet une génération automatisée et scalable des embeddings, essentielle pour la qualité des recommandations du système

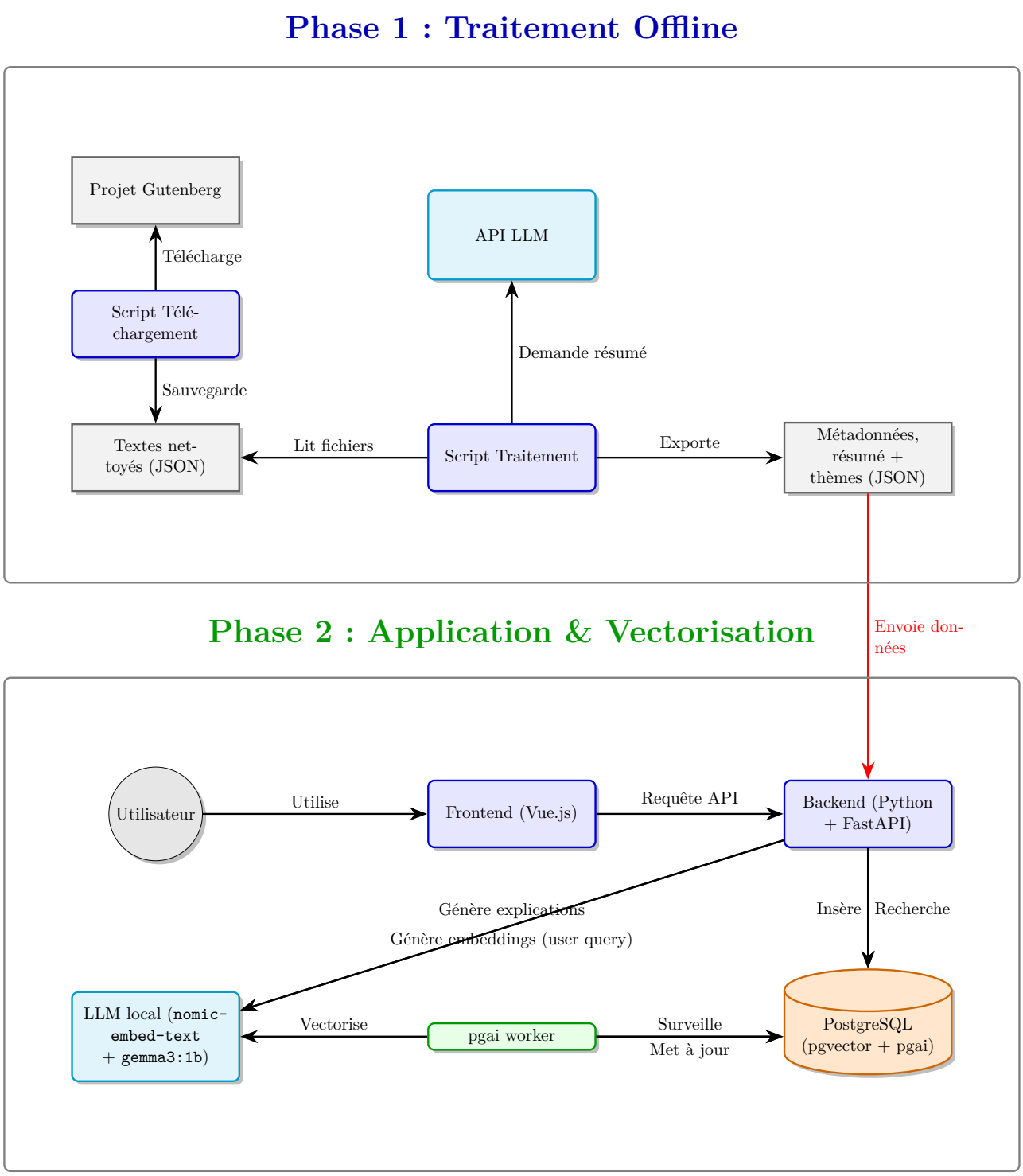


Figure 6.1 – *Pipeline de traitement d'un livre : Du téléchargement jusqu'au embedding et utilisation dans l'application*

6.2 Backend

Le service backend est le coeur logique de l'application. Il fait le pont entre l'interface utilisateur (frontend), la base de données et le service Ollama pour la génération d'embedding de query.

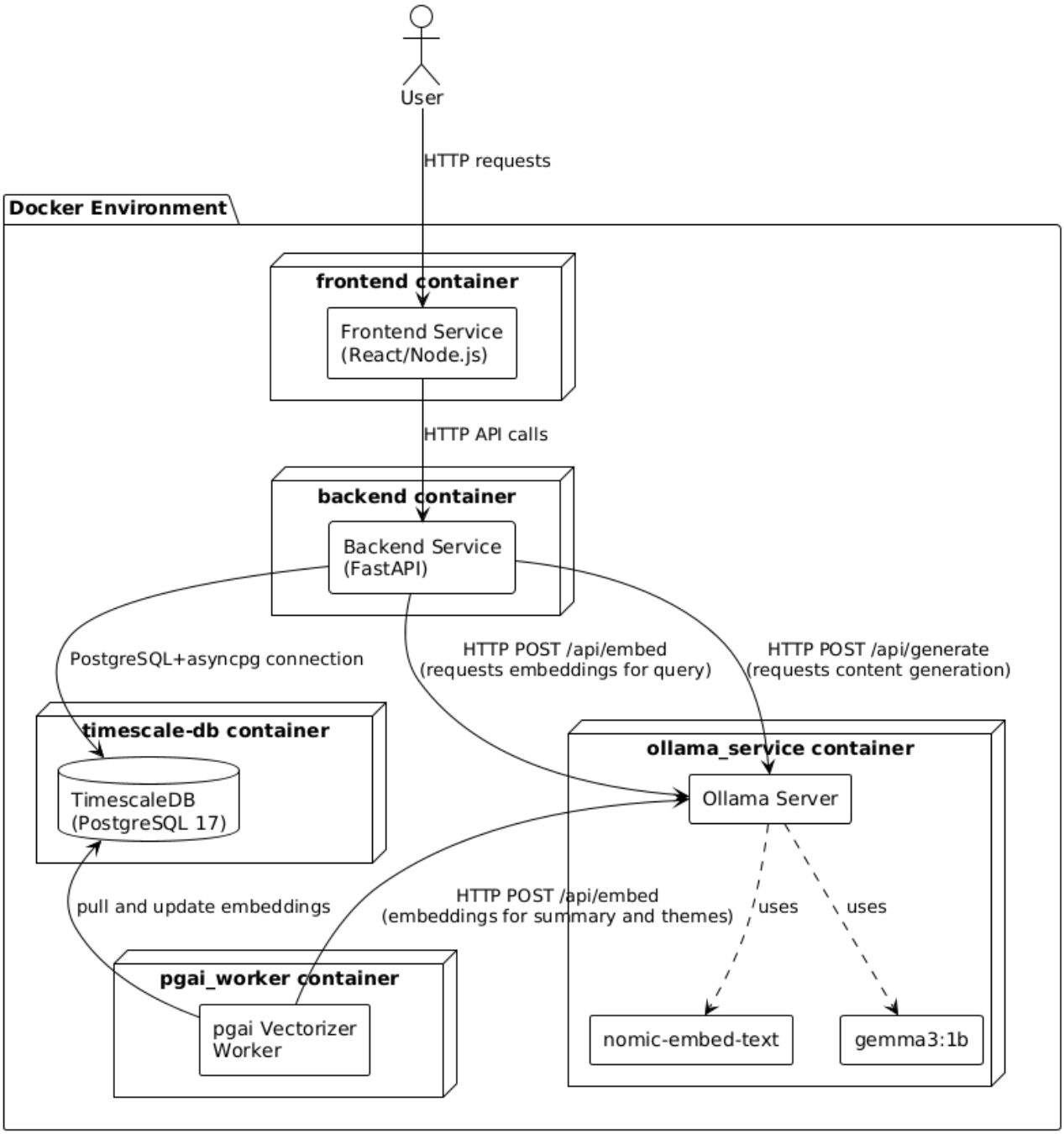


Figure 6.2 – *Architecture de l'application*

6.2.1 Technologies utilisées

FastAPI : Un framework web moderne, rapide et performant pour la construction d'APIs avec Python 3.8+. [52] FastAPI fournit des performances élevées, grâce à son fondement sur Starlette et Pydantic. L'utilisation de type hints avec Python facilite le développement et réduit les bugs. FastAPI fournit également un documentation interative généré automatiquement.

Python : Le langage de programmation principal du backend, choisi pour sa polyvalence, son vaste écosystème de bibliothèques et son efficacité pour les applications de machine learning.

SQLAlchemy Async : Un ORM (Object Relational Mapper) et un toolkit SQL pour Python. [59] L'utilisation du pilote asyncpg assure une communication asynchrone avec la base de données PostgreSQL, évitant ainsi le blocage des opérations d'E/S et maintenant la haute performance de FastAPI. Cela permet de définir des modèles de base de données, d'effectuer des opérations CRUD (Create, Read, Update, Delete) et de gérer des sessions de manière asynchrone.

Alembic : Un outil de migration de base de données léger, spécifiquement conçu pour SQLAlchemy. Il est indispensable pour gérer les modifications de schéma de base de données de manière contrôlée et versionnée.

Extension python `pgai` Une intégration transparente avec les ORMs (comme SQLAlchemy), permettant de définir des colonnes vectorielles dans vos modèles et d'effectuer des requêtes de similarité vectorielle.

6.2.2 Fonctionnalités

Le service backend est responsable de :

- La gestion des requêtes HTTP provenant du frontend.
- La mise en oevre de la logique métier.
- L'interaction avec la base de données TimescaleDB via SQLAlchemy pour la persistance et la récupération des données, y compris la gestion des données vectorielles avec pgai.

6.3 Frontend

Le service frontend est l'interface utilisateur de l'application Il est construit sur un empilement technologique moderne, centré sur Vue.js et enrichi par la bibliothèque de composants Vuetify.

6.3.1 Technologies utilisées

Vue.js : Un framework JavaScript progressif pour la construction d'interfaces utilisateur. Vue.js est réputé pour sa facilité d'apprentissage, sa flexibilité et ses excellentes performances.

Vuetify : Un framework de composants UI complet pour Vue.js, qui implémente le Material Design de Google. Vuetify accélère considérablement le développement de l'interface utilisateur en fournissant une vaste collection de composants pré-construits, élégants et responsives.

Node.js : L'environnement d'exécution JavaScript côté serveur.

6.3.2 Fonctionnalités

Le service frontend assure les responsabilités suivantes :

- **Rendu de l'interface utilisateur** : Afficher l'application web dans le navigateur de l'utilisateur, créant une expérience fluide et intuitive.
- **Gestion des interactions utilisateur** : Capturer et réagir aux entrées de l'utilisateur (clics, saisies, etc.) et mettre à jour l'interface en conséquence.
- **Communication avec le backend** : Effectuer des requêtes HTTP asynchrones (via des bibliothèques comme Axios) vers l'API du **service backend** pour envoyer des données (ex : formulaires) et récupérer les informations nécessaires à l'affichage.

6.4 Service Ollama

Le service ollama agit comme un point de terminaison local pour l'exécution de modèles de langage de grande taille (LLMs) et, plus spécifiquement dans notre cas, de modèles d'embedding et de génération de texte. Son intégration via Docker permet une gestion facile et une isolation de l'environnement d'exécution des modèles.

6.4.1 Modèle d'embedding

Le modèle d'embedding utilisé dans ce projet est `nomic-embed-text`. Le modèle `nomic-embed-text` est reconnu pour ses performances élevées dans la génération d'embeddings textuels, surpassant des modèles comme OpenAI text-embedding-ada-002 et text-embedding-3-small sur diverses tâches de contexte court et long. [41, 23]

6.4.2 Modèle de génération de texte

En plus du modèle d'embedding, le service Ollama intègre également le modèle `gemma3:1b` pour la génération d'explications textuelles. Ce modèle est utilisé spécifiquement pour générer des explications détaillées des similarités entre deux livres en analysant leurs résumés, titres et thèmes respectifs. Les détails du prompt utilisé et la méthodologie de génération sont présentés dans la section 6.8.3.

6.4.3 Fonctionnalités

Le service Ollama, en conjonction avec les modèles nomic-embed-text et gemma3:1b, remplit des fonctions critiques dans la pipeline de recommandation. Il reçoit des requêtes du **pgai_worker** et également du backend pour les embeddings de requêtes. En réponse, il calcule et renvoie les vecteurs d'embedding correspondants via le modèle nomic-embed-text. Parallèlement, il traite les demandes de génération d'explications de similarité via le modèle gemma3:1b, en analysant les métadonnées des livres pour produire des justifications textuelles compréhensibles des recommandations.

6.5 Base de données

Le service db est la couche de persistance fondamentale de notre application. Il est basé sur l'image Docker `timescale/timescaledb-ha:pg17`, qui fournit une instance de PostgreSQL 17 augmentée par l'extension TimescaleDB. [60] Ce choix technologique est stratégique car il transforme notre base de données relationnelle standard en une puissante "base de données vectorielle" grâce à l'intégration des extensions PostgreSQL `pgvector` et `pgai`. L'extension `pgvector` est la brique de base, ajoutant le type de données `VECTOR` et les opérateurs essentiels pour la recherche de similarité du plus proche voisin, permettant ainsi le stockage et l'interrogation efficaces des embeddings numériques. L'extension `pgai` s'appuie sur `pgvector` pour fournir des fonctions SQL directement orientées IA, facilitant l'intégration avec des modèles de langage externes et la gestion des flux de données pour des applications comme la génération augmentée par récupération (RAG). Ensemble, ces technologies permettent à la base de données de gérer non seulement les données relationnelles, mais aussi les représentations vectorielles qui sont générées et mises à jour automatiquement. L'extension pgai permet la définition de `vectorizer` qui permet, en combinaisont avec le modèle d'embeding local d'Ollama et le service pg_ai worker, de créer et mettre à jour automatiquement les données devant être "embed" dans la base de donnée.

6.5.1 Schéma de la base de données

Le schéma de la base de donnée est visible en figure 6.3

`users` (Utilisateurs)

- **Objectif :** Gérer les informations relatives aux utilisateurs de la plateforme.
- **Champs :**
  - `id` (entier) : Identifiant unique de l'utilisateur (clé primaire).

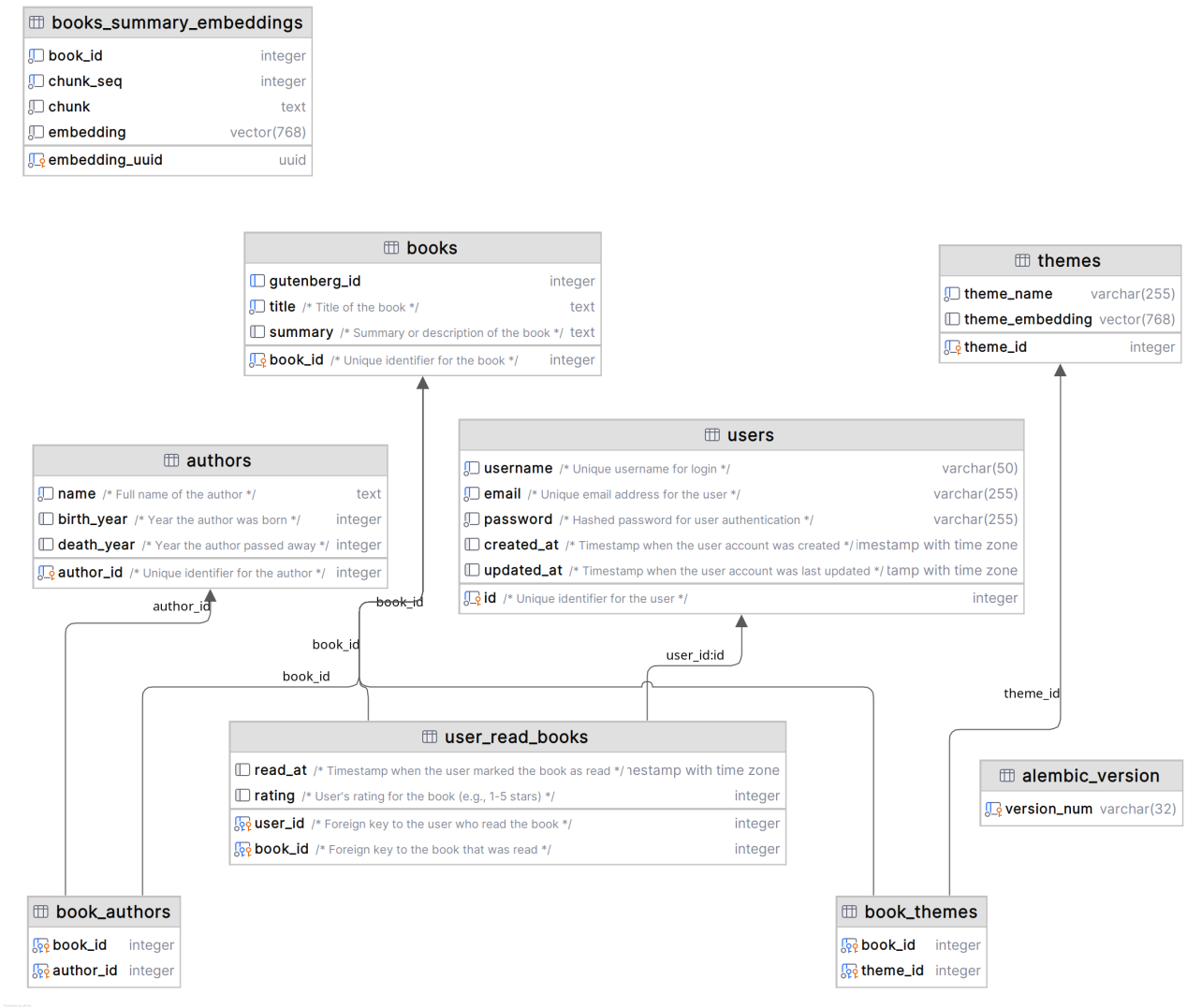


Figure 6.3 – *Schema de la database*

- `username` (varchar(50)) : Nom d'utilisateur unique.
- `email` (varchar(255)) : Adresse email unique de l'utilisateur.
- `password` (varchar(255)) : Hachage du mot de passe pour l'authentification.
- `created_at` (timestamp avec fuseau horaire) : Horodatage de la création du compte utilisateur.
- `updated_at` (timestamp avec fuseau horaire) : Horodatage de la dernière mise à jour du compte utilisateur.

`books` (Livres)

- **Objectif :** Stocker les informations principales sur les livres.
- **Champs :**
  - `book_id` (entier) : Identifiant unique du livre (clé primaire).
  - `gutenberg_id` (entier) : Identifiant du livre dans le projet Gutenberg.
  - `title` (texte) : Titre du livre.
  - `summary` (texte) : Résumé ou description du contenu du livre.

`authors` (Auteurs)

- **Objectif :** Contenir les informations sur les auteurs des livres.
- **Champs :**
  - `author_id` (entier) : Identifiant unique de l'auteur (clé primaire).
  - `name` (texte) : Nom complet de l'auteur.
  - `birth_year` (entier) : Année de naissance de l'auteur.
  - `death_year` (entier) : Année de décès de l'auteur (si applicable).

`themes` (Thèmes)

- **Objectif :** Gérer les thèmes associés aux livres et leurs représentations vectorielles.
- **Champs :**
  - `theme_id` (entier) : Identifiant unique du thème (clé primaire).
  - `theme_name` (varchar(255)) : Nom du thème.
  - `theme_embedding` (vector(768)) : Vecteur (embedding) représentant le thème.

`books_summary_embeddings` (Embeddings des résumés de livres)

- **Objectif :** Stocker les embeddings des résumés de livres pour des applications de recherche ou de recommandation.
- **Champs :**
  - `book_id` (entier) : Identifiant du livre (clé étrangère vers `books`).
  - `chunk_seq` (entier) : Séquence du fragment (chunk) si le résumé est divisé.
  - `chunk` (texte) : Contenu du fragment de texte.
  - `embedding` (vector(768)) : Vecteur de plongement (embedding) du fragment.
  - `embedding_uuid` (uuid) : Identifiant unique universel pour l' embedding.

`user_read_books` (Livres lus par les utilisateurs)

- **Objectif :** Enregistrer les livres lus par les utilisateurs, y compris la date de lecture et l'évaluation.
- **Champs :**
  - `user_id` (entier) : Identifiant de l'utilisateur (clé étrangère vers `users`).
  - `book_id` (entier) : Identifiant du livre (clé étrangère vers `books`).
  - `read_at` (timestamp avec fuseau horaire) : Horodatage de la lecture du livre.
  - `rating` (entier) : Évaluation du livre par l'utilisateur (de 1 à 5 étoiles).
  - **Clé primaire composite :** (`user_id`, `book_id`) pour assurer l'unicité de la lecture d'un livre par un utilisateur.

`book_authors` *(Auteurs de livres - table de jonction)*

- **Objectif :** Établir la relation "plusieurs-à-plusieurs" entre les livres et les auteurs (un livre peut avoir plusieurs auteurs, un auteur peut écrire plusieurs livres).
- **Champs :**
  - `book_id` (entier) : Identifiant du livre (clé étrangère vers `books`).
  - `author_id` (entier) : Identifiant de l'auteur (clé étrangère vers `authors`).
  - **Clé Primaire Composite :** (`book_id`, `author_id`).

`book_themes` *(Thèmes de livres - table de jonction)*

- **Objectif :** Établir la relation "plusieurs-à-plusieurs" entre les livres et les thèmes (un livre peut aborder plusieurs thèmes, un thème peut être présent dans plusieurs livres).
- **Champs :**
  - `book_id` (entier) : Identifiant du livre (clé étrangère vers `books`).
  - `theme_id` (entier) : Identifiant du thème (clé étrangère vers `themes`).
  - **Clé Primaire Composite :** (`book_id`, `theme_id`).

9. `alembic_version` *(Historique des migrations alembic)*

- **Objectif :** Suivre l'historique des migrations de la base de données gérées par Alembic.
- **Champs :**
  - `version_num` (varchar(32)) : Numéro de version de la migration actuelle.

Relations entre les Tables

Les relations entre les tables sont établies via des clés étrangères, assurant l'intégrité référentielle de la base de données :

- `users` est liée à `user_read_books` par `user_id`.
- `books` est liée à :
  - `user_read_books` par `book_id`.
  - `book_authors` par `book_id`.
  - `book_themes` par `book_id`.
  - `books_summary_embeddings` par `book_id`.
- `authors` est liée à `book_authors` par `author_id`.
- `themes` est liée à `book_themes` par `theme_id`.

6.6 Service `pgai` vectorizer worker

Le service `pgai_worker` est un composant dédié et découplé, essentiel pour l'automatisation de la génération d'embeddings et leur synchronisation avec la base de données. Il utilise l'image Docker `timescale/pgai-vectorizer-worker:latest`, conçue spécifiquement pour interagir avec l'extension `pg_ai` de PostgreSQL. Le rôle principal de ce worker est de décharger le service backend et la base de données de la tâche de transformation de texte en vecteurs et son automatisation. Il fonctionne en interrogeant périodiquement la base de données TimescaleDB (avec un intervalle de temps configurable) pour identifier les nouvelles données qui nécessitent une vectorisation. Une fois les données identifiées, le pgai_worker envoie ces données textuelles au service Ollama pour obtenir les embeddings correspondants à l'aide du modèle `nomic-embed-text`. Après avoir reçu les vecteurs générés, le worker les réinsère ou les met à jour dans les colonnes `VECTOR` appropriées de la base de données, assurant ainsi que les données textuelles sont toujours accompagnées de leurs représentations sémantiques. Cette approche asynchrone garantit que la génération d'embeddings ne perturbe pas la performance des opérations en temps réel de l'application.

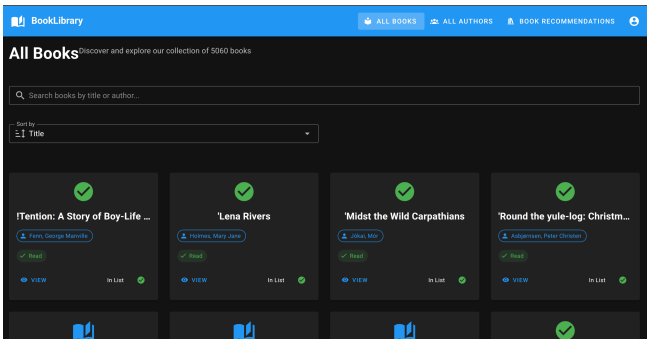


Figure 6.4 – *Section pour voir tous les livres disponibles*

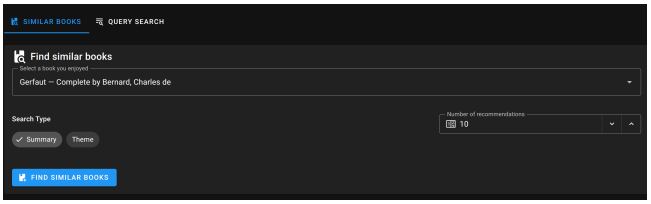


Figure 6.5 – *Recommendation par rapport un un autre livre*

6.7 Populer la base de données

Le processus d'ingestion des données traitées dans la base de données est simplifié. Une fois les livres traités par le module de *processing*, les fichiers JSON de sortie (conformes au schéma `ProcessorOutput`) peuvent être envoyés à l'API du backend via le point de terminaison `/books` et la requête CRUD de type `POST`. Ce processus d'ingestion gère intelligemment les dépendances :

- Si un auteur mentionné dans le fichier n'existe pas encore dans la table `authors`, une nouvelle entrée est automatiquement créée pour cet auteur.
- De même, pour chaque thème identifié, s'il n'est pas déjà présent dans la table `themes`, il est créé.
- Pour les livres, la logique d'ingestion vérifie l'existence de l'identifiant `gutenberg_id`. Si un livre avec le même `gutenberg_id` existe déjà dans la base de données, l'ingestion est ignorée afin d'éviter les doublons et de garantir l'unicité des oeuvres venant de Gutenberg dans le catalogue.

6.7.1 Récupérer les données déjà traitées

Comme mentionnée précédemment, 5060 livres on été téléchargés et traité par la pipeline. Dans le projet un fichier .bak provenant d'un dump de la base de données est fourni. Un script bash permettant l'initialisation de la base de donnée ou la restoration de la base de donnée est également fourni. Le dump contient toutes les schémas décrits plus haut, toutes les données, y les Vectorizer et les embeddings associés.

6.8 Aperçu de l'application

Cette application de recommandation de livres offre une expérience complète aux utilisateurs, combinant des fonctionnalités de gestion de bibliothèque, de découverte de livres et de personnalisation intelligente basée sur l'intelligence artificielle. L'interface utilisateur, développée avec Vue.js et Vuetify, est conçue pour être intuitive et réactive sur diverses plateformes.

6.8.1 Fonctionnalités utilisateur et frontend

L'application place l'utilisateur au centre de l'expérience grâce à une série de fonctionnalités accessibles via une interface moderne et dynamique.

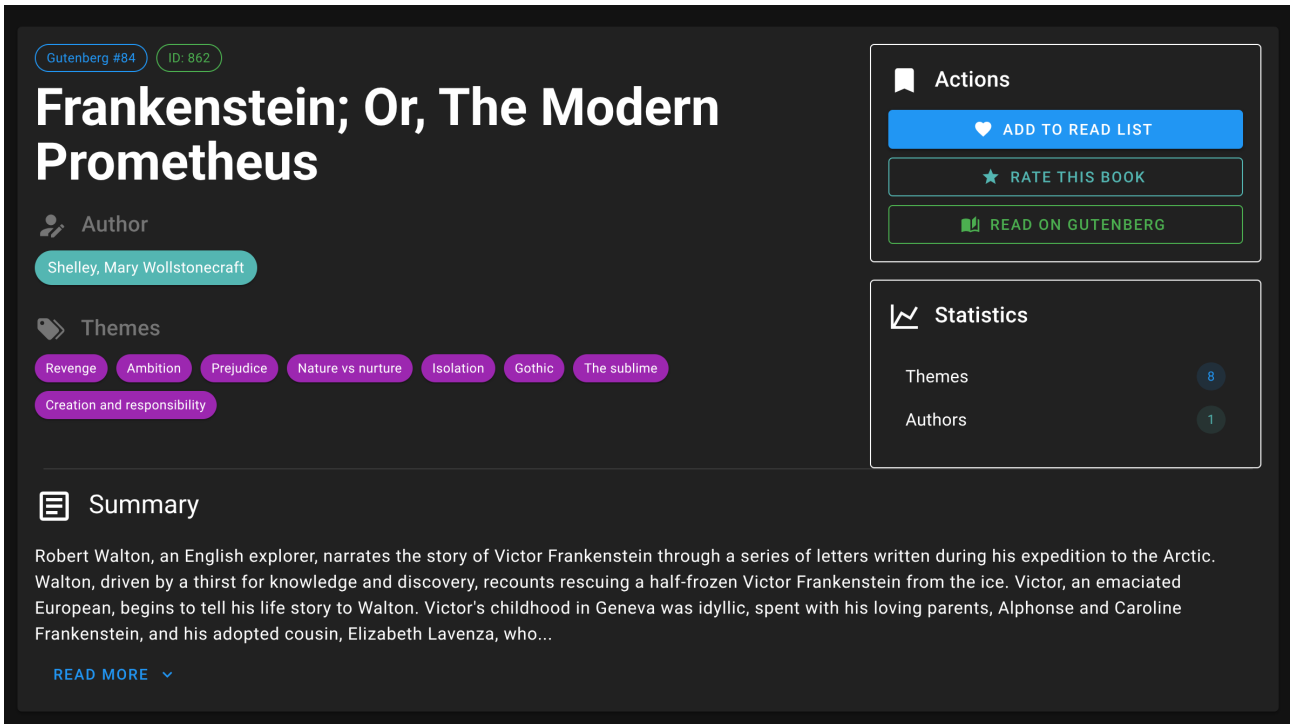


Figure 6.6 – *Page de détail du livre Frankenstein*

Authentification et gestion de profil :
- *Connexion et inscription* : Les nouveaux utilisateurs peuvent facilement s'inscrire et les utilisateurs existants se connecter pour accéder à leurs fonctionnalités personnalisées.
- *Gestion du profil* : Chaque utilisateur dispose d'un profil détaillé où il peut consulter ses informations personnelles (username, email), et sa liste de livres.
- L'authentification est gérée avec un token jwt

Gestion de la bibliothèque personnelle :
- *Liste de lecture* : Les utilisateurs peuvent ajouter et retirer des livres de leur liste de lecture.
- *Notation de livres* : Après avoir lu un livre, les utilisateurs peuvent lui attribuer une note de 1 à 5.

Exploration du catalogue :
- *Détails des livres* : Une page dédiée pour chaque livre présente toutes les informations pertinentes : titre, auteur(s), résumé, thèmes associés et la possibilité de le noter ou de l'ajouter à une liste (voir figure 6.6).
- *Détails des auteurs* : Les utilisateurs peuvent consulter des profils d'auteurs incluant les autres livres de la plateforme dont ils sont auteur
- *Recherche avancée* : L'application propose une fonction de recherche permettant de trouver des livres par titre ou auteur.

6.8.2 Fonctionnalités de recommandation

Au coeur de l'application se trouvent des fonctionnalités de recommandation, tirant parti de l'intelligence artificielle pour offrir des suggestions pertinentes et personnalisées. Un aperçu de l'interface de recommandation en Figure 6.7.

Recommandations par similarité de contenu :
- *Similarité de livre à livre* : Une page dédiée permet de découvrir des livres similaires à un ouvrage spécifié. La similarité est calculée en comparant les *embeddings* des résumés ou des thèmes des livres, ce qui permet d'identifier des oeuvres ayant un sens ou un esprit proche, au-delà de simples correspondances de mots-clés. Un exemple de résultat d'une telle recherche est visible en Figure 6.8.

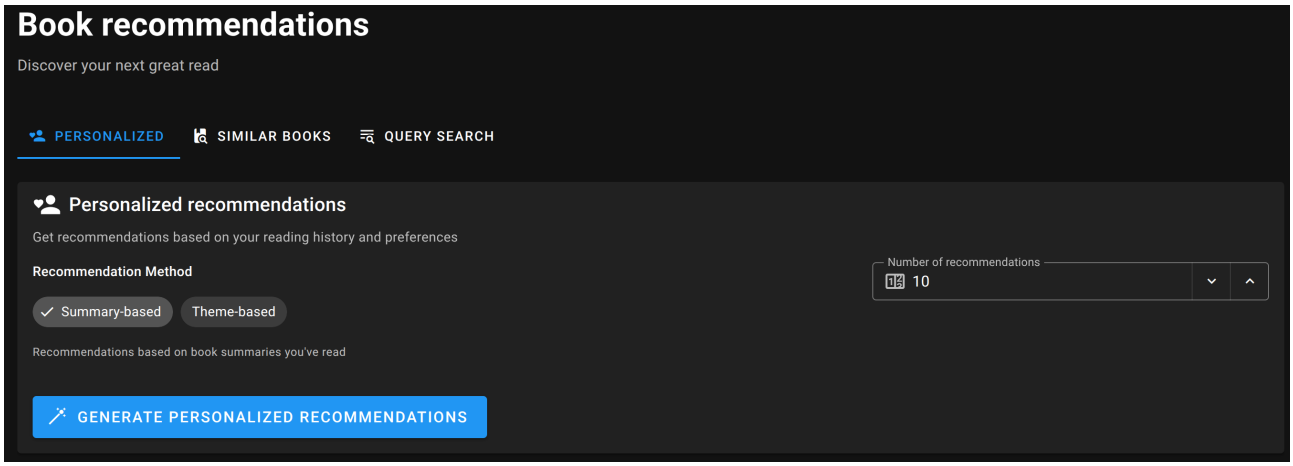


Figure 6.7 – *Page de recommendations*

— *Recherche par requête textuelle et thèmes* : Les utilisateurs peuvent utiliser une requête en langage naturel (par exemple, "livres sur l'aventure et l'exploration de mondes fantastiques") ou une sélection de thèmes prédéfinis. La requête ou les thèmes sont convertis en embeddings et utilisés pour rechercher des livres dont le résumé et les thèmes correspondent le plus sémantiquement.

Recommandations personnalisées via le profil utilisateur :

— *Basé sur les livres lus* : Depuis la même page de recommandations, en étant connecté, les utilisateurs peuvent générer des recommandations basées sur leur historique de lecture. L'application analyse les thèmes dominants ou les résumés des livres qu'ils ont lus pour proposer des ouvrages qui s'alignent avec leurs goûts littéraires établis.

6.8.3 Explication des recommandations

Dans la section de recommandation "similarité livre à livre" de l'application, les utilisateurs ont la possibilité d'obtenir une explication détaillée de chaque suggestion. En cliquant sur un bouton dédié, le système déclenche la génération d'une justification pour la recommandation. Cette explication est produite par un LLM local, spécifiquement le modèle *gemma3:1b*, via une intégration avec le service Ollama. Cette approche permet de fournir une transparence accrue sur les raisons d'une recommandation, en explicitant les liens sémantiques, thématiques ou stylistiques identifiés par le modèle entre le livre de référence et l'uvre suggérée. En utilisant un modèle local tournant dans un Docker, la recommandation prend environ trente secondes pour commencer à être générée. Le résultat est "streamé" pour voir la génération en temps réel. Un aperçu de cette interface est visible en figure 6.9. Outre la génération d'explications dynamiques par le LLM, la disponibilité des résumés et des thèmes pour chaque livre dans l'application contribue également de manière significative à l'explicabilité des recommandations pour l'utilisateur. Les résumés offrent un aperçu rapide de l'intrigue et des points clés du livre, permettant à l'utilisateur de comprendre pourquoi un ouvrage est pertinent. Les thèmes, quant à eux, fournissent une vue conceptuelle des sujets abordés, facilitant l'identification de correspondances thématiques avec leurs propres intérêts ou avec les livres de référence. Cette transparence inhérente aux données traitées enrichit l'expérience utilisateur et renforce la confiance dans le système de recommandation.

Approche et prompt utilisé pour générer les explications

Pour générer l'explication de la similarité entre deux livres, le système exploite les capacités du modèle LLM local `gemma3:1b` hébergé via Ollama. Cette approche vise à fournir des justifications détaillées et contextuelles en tirant parti de la compréhension sémantique des LLMs. Le choix de ce modèle a été guidé par un compromis entre performance et contraintes de puissance computationnelle. `gemma3:1b` (1 milliard de paramètres) a été sélectionné car il représente l'un des plus petits modèles capables de

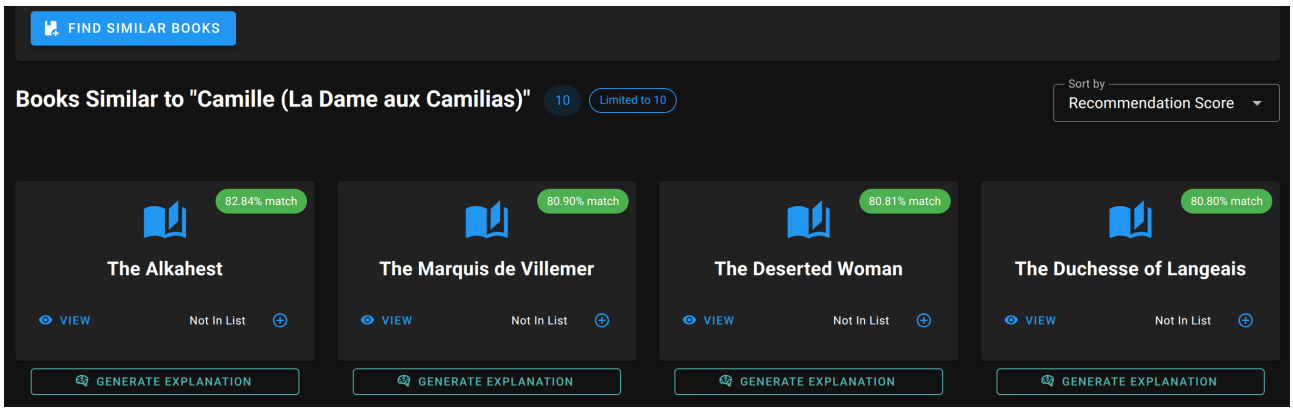


Figure 6.8 – *Exemple de recommandations de livres similaires à Camille, de Alexandre Dumas*

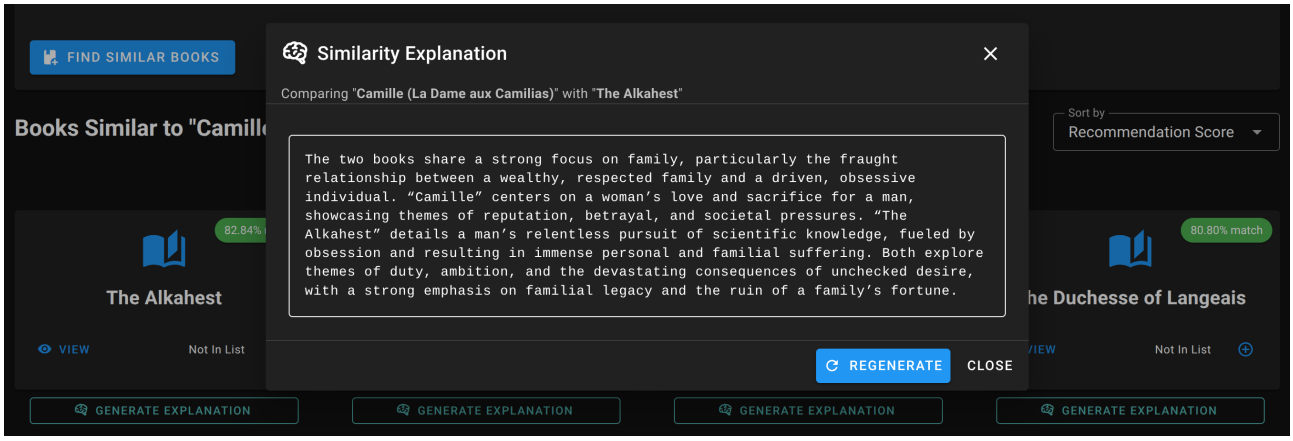


Figure 6.9 – *Résultat de l'explicaion de similarité générée par LLM entre le livre Camille (La Dame aux Camilias) et The Alkahest*

produire des résultats concluants pour des tâches de raisonnement et de génération d'explications complexes. Des expérimentations avec des modèles encore plus petits, comme smollm:135m (135 millions de paramètres), n'ont pas donné de résultats satisfaisants, les rendant inadaptés pour cette fonctionnalité exigeante en compréhension sémantique. L'environnement local du projet, bien que bénéficiant des avantages d'Ollama pour le déploiement de LLM, présente des limitations en termes de ressources matérielles disponibles, rendant impraticable l'utilisation de modèles plus volumineux. L'explication est générée en soumettant au LLM un prompt soigneusement élaboré qui inclut les informations suivantes pour les deux livres comparés :

- Le titre de chaque livre.
- Un résumé complet pour chaque livre (préalablement généré lors de la phase de traitement des données).
- La liste des thèmes associés à chaque livre (également extraits lors du traitement).

Le prompt est structuré pour demander au LLM de mettre en évidence les similarités et les points de connexion entre les deux ouvrages, en se basant sur leurs résumés et leurs thèmes. Le prompt utilisé est présenté en 6.1

```
Given the following two books:

{title_1}:
Themes: {book_1_themes}
Summary: {summary_1 or "No summary available"}

{title_2}:
Themes: {book_2_themes}
Summary: {summary_2 or "No summary available"}

Based on their titles, themes, and summaries, provide a concise explanation of their similarities and
↪  differences.
Your response should refer to the books by their titles (e.g., "{title_1}" and "{title_2}").
Your response should contain ONLY this explanation, without any introductory phrases or concluding
↪  remarks.
```

Listing 6.1 – *Prompt utilisé pour la génération d'explication de similarité entre deux livres*

6.8.4 Implémentation du calcul de similarité

Le coeur des fonctionnalités de recommandation sémantique repose sur la capacité à transformer le texte (résumés de livres, thèmes, requêtes utilisateur) en représentations numériques appelées *embeddings*. Ces vecteurs permettent de mesurer la similarité conceptuelle entre différents éléments, au-delà d'une simple correspondance lexicale. Les embeddings sont calculés pour différentes entités :

- **Résumés de livres** : Chaque résumé de livre est converti en un embedding pour permettre la recherche de livres similaires en fonction de leur contenu textuel.
- **Thèmes de livres** : Chaque thème associé à un livre est également converti en un embedding. Pour un livre ayant plusieurs thèmes, une agrégation est nécessaire. Actuellement, l'agrégation des embeddings de thèmes pour un livre donné s'effectue par une moyenne simple des embeddings dans la requête SQL).
- **Requêtes utilisateur** : Les requêtes textuelles des utilisateurs et les listes de thèmes spécifiés par l'utilisateur sont également embeddées (et agrégées par moyenne pour les thèmes) afin de pouvoir les comparer aux embeddings des livres.
- **Profil utilisateur pour la recommandation personnalisée** : Pour générer des recommandations basées sur l'historique de lecture d'un utilisateur, une approche similaire d'agrégation est appliquée. L'embedding du profil utilisateur est calculé comme la oyenne des embeddings des résumés (ou des thèmes) de tous les livres que l'utilisateur a lus. Cela crée une représentation vectorielle de ses préférences globales, permettant de trouver des livres similaires qui n'ont pas encore été lus.

Les calculs de similarité sont ensuite effectués directement dans la base de données PostgreSQL, en utilisant l'opérateur de distance cosinus (`<=>`) fourni par `pgvector`. Cet opérateur est privilégié car il mesure l'angle entre les vecteurs, ce qui est particulièrement adapté pour évaluer la similarité sémantique indépendamment de la magnitude des vecteurs.

6.8.5 Améliorations futures et perspectives

Plusieurs pistes d'amélioration peuvent être envisagées pour affiner la pertinence des recommandations :

- **Optimisation de l'application et montée en charge** Si les données venaient à devenir beaucoup plus importantes, il deviendrait alors pertinent d'indexer les embeddings. De plus, actuellement, tous les embeddings demandé par le worker, les queries, et les explications sont générés par le même serveur Ollama. Ce fait peut déjà s'avérer bloquant dans le cas où l'on souhaite chercher un livre à l'aide d'une query alors qu'une quantité massive d'embeddings est en train d'être générée. Pour réellement supporter de multiples utilisateurs simultanés, il faudrait dupliquer les instances de votre service Ollama, en dédiant certaines à la génération d'embeddings et d'autres à la génération de contenu, tout en mettant en place un système de répartition de charge.
- **Prise en compte des notes utilisateur dans les agrégations d'embeddings** : Actuellement, l'embedding agrégé du profil utilisateur est une simple moyenne des embeddings des livres lus. Une amélioration significative consisterait à pondérer cette moyenne par les notes attribuées par l'utilisateur. Par exemple, un livre noté 5/5 pourrait avoir un poids plus important dans le calcul de l'embedding agrégé qu'un livre noté 3/5. Cela permettrait de mieux refléter les préférences de l'utilisateur, en donnant plus d'influence aux lectures qu'il a réellement appréciées. Des techniques de pondération basées sur la note (ex : note normalisée) pourraient être explorées lors de l'agrégation des vecteurs.
- **Filtrage collaboratif basé sur les notes** : L'intégration des notes utilisateurs ouvre la voie au développement futur de méthodes de recommandation par filtrage collaboratif. Le filtrage collaboratif, qu'il soit centré sur l'utilisateur (trouver des utilisateurs aux goûts similaires) ou sur l'élément (trouver des livres similaires basés sur les notes d'autres utilisateurs), pourrait compléter les recommandations basées sur les embeddings sémantiques. Les notes de 1 à 5 sont des données précieuses qui, une fois collectées en volume, peuvent être exploitées pour identifier des corrélations entre utilisateurs ou livres, offrant ainsi une perspective différente et potentiellement plus nuancée des préférences. L'architecture actuelle avec la persistance des notes est déjà prête pour l'implémentation de tels algorithmes en tant que future amélioration.
- **Recommandation hybride** : La combinaison du filtrage basé sur le contenu (via les embeddings) et du filtrage collaboratif (via les notes) permettrait de construire un système de recommandation hybride. Ces systèmes sont souvent les plus performants car ils capitalisent sur les forces de chaque approche, palliant leurs faiblesses respectives (par exemple, le "problème du démarrage à froid" pour le filtrage collaboratif, ou le manque de personnalisation directe pour le filtrage par contenu pur).

Chapitre 7

Analyse des résultats

7.1 Analyse et caractéristiques des données

La qualité et les spécificités des données sont primordiales pour le fonctionnement efficace d'un système de recommandation. Notre application utilise une base de données de livres provenant principalement d'auteurs du 19ème siècle, dont la source principale est le Projet Gutenberg.

7.1.1 Gestion de la variabilité des résumés pour l'évaluation de la pertinence

Une observation clé concernant les données du Projet Gutenberg est la présence fréquente de multiples éditions pour un même ouvrage. Des classiques tels que *Frankenstein* de Mary Shelley ou *Alice au pays des merveilles* de Lewis Carroll sont souvent disponibles en plusieurs versions (par exemple, différentes traductions, éditions annotées, ou formats). Bien qu'il s'agisse du même titre littéraire, ces éditions peuvent avoir des résumés (ou introductions) légèrement, voire significativement, distincts, en fonction de la source ou de la personne qui les a ajoutés.

Cette variabilité des résumés associés à un même livre, s'est avérée particulièrement utile pour l'évaluation qualitative de la pertinence de nos recommandations. En effet, la présence de plusieurs embeddings pour un même livre (correspondant à ses différentes éditions et résumés) nous offre un terrain d'essai intéressant. Le système recommande différentes éditions d'un même ouvrage comme étant similaires entre elles, même lorsque leurs résumés d'entrée sont variés. Cela valide la capacité des LLM à générer des résumés et des thèmes qui capturent le sens profond du livre, indépendamment des variations textuelles mineures des résumés originaux. Plus spécifiquement, il a été observé que des oeuvres comme *Frankenstein* et *Alice au pays des merveilles*, bien que présentes sous diverses éditions avec des résumés initiaux distincts, sont toujours reconnues comme étant fortement similaires par le système. Ces résultats attestent de la robustesse sémantique du modèle d'embedding utilisé et de la pertinence des informations générées par le LLM, prouvant leur capacité à identifier les nuances sémantiques essentielles.

7.2 Validation de la pertinence des recommandations

Étant donné les ressources limitées pour une évaluation quantitative à grande échelle, le choix s'est porté sur une approche qualitative afin de démontrer la pertinence des recommandations générées par le système. Cette section présente des exemples concrets qui illustrent l'efficacité des méthodes basées sur les embeddings pour les différentes stratégies de recommandation implémentées. L'objectif est de montrer que les résultats produits sont sémantiquement cohérents et intuitivement pertinents pour un utilisateur.

7.2.1 Méthodologie d'évaluation qualitative

Pour évaluer la pertinence des recommandations, plusieurs exemples de similarité par livre, par résumé et par thèmes est étudiée. Un exemple de recherche par query de résumé et thème est également montrée.

Tous les résultats de similarité proviennent d'un calcul de similarité cosinus dont la valeur est comprise entre 0 et 1. Plus le score est proche de 1, plus le livre est similaire.

Bien que cette méthode ne fournisse pas de métriques statistiques, elle permet d'avoir un premier aperçu de la pertinence des résultats.

Similarité livre à livre (par résumé)

Livre à comparer : *The Aliens* de Leinster, Murray

Résultats générés :

Table 7.1 – *Livres similaires à "The Aliens"*

Titre	Auteur	Similarité
The Galaxy Primes	Fritz Leiber	0.8129
The Wailing Asteroid	Murray Leinster	0.8076
Vampires of Space	Murray Leinster	0.7955
A Trip to Venus : A Novel	John Munro	0.7856

Analyse : Les livres suggérés dans cette recommandation sont majoritairement des oeuvres de science-fiction, ce qui correspond bien au genre de *The Aliens* de Murray Leinster. Il est intéressant de noter que deux des quatre recommandations sont également écrites par Murray Leinster lui-même (*The Wailing Asteroid* et *Vampires of Space*), ce qui renforce la cohérence stylistique et thématique au sein de l'oeuvre d'un même auteur.

The Galaxy Primes de Fritz Leiber, bien qu'écrit par un auteur différent, s'inscrit également dans le genre de la science-fiction, suggérant que le système identifie des similarités au-delà des seuls auteurs. La présence de *A Trip to Venus : A Novel* de John Munro, un ouvrage potentiellement plus ancien ou moins connu, indique que le système peut repérer des similarités thématiques ou de genre même avec des oeuvres de différents niveaux de popularité ou époques, tant qu'elles partagent des caractéristiques fondamentales (ici, l'exploration spatiale et la rencontre avec l'inconnu), ce qui est un avanatage de la recommandation basée sur le contenu par rapport au filtrage collaboratif. Ces résultats démontrent que le système de recommandation parvient à identifier des proximités de genre et d'auteur, offrant des suggestions pertinentes pour les amateurs de science-fiction.

Similarité livre à livre (par résumé)

Livre à comparer : *The Strange Case of Dr. Jekyll and Mr. Hyde* de Robert Louis Stevenson

Résultats générés :

Analyse : Les oeuvres suggérées affichent une cohérence thématique marquée avec *Dr. Jekyll and Mr. Hyde*. La première entrée, une autre édition de l'oeuvre de référence, montre un score de similarité élevé, ce qui renforce la confiance dans la capacité des LLM à générer des résumés pertinents. Les titres recommandés présentent des éléments de mystère, de dualité morale, de quête d'immortalité ou de transformations, qui sont des thèmes centraux du roman de Stevenson. L'inclusion de *The Picture of Dorian Gray* et *Septimius Felton* suggère que le système est capable d'identifier des oeuvres explorant des questionnements sur l'identité, la conscience et le surnaturel. *The Adventure of the Devil's Foot*, une enquête de Sherlock Holmes, introduit une dimension policière compatible avec l'ambiance sombre et

Table 7.2 – *Livres similaires à "The Strange Case of Dr. Jekyll and Mr. Hyde"*

Titre	Auteur	Similarité
The Strange Case of Dr. Jekyll and Mr. Hyde	Robert Louis Stevenson	0.9081
The Adventure of the Devil's Foot	Arthur Conan Doyle	0.7198
The Picture of Dorian Gray	Oscar Wilde	0.7190
Guy Fawkes ; or, The Gunpowder Treason	William Harrison Ainsworth	0.7164
Septimius Felton, or, the Elixir of Life	Nathaniel Hawthorne	0.7110

troublante du roman original. Les similarités détectées corroborent la capacité du système à regrouper des récits partageant une atmosphère, une période et des thèmes psychologiques complexes.

.

Similarité livre à livre (par thème)

Livre à comparer : *The Aliens* de Leinster, Murray

Résultats générés :

Table 7.3 – *Livres similaires à "The Aliens"*

Titre	Auteur	Similarité
The Fifth-Dimension Tube	Murray Leinster	0.8985
The Walrus Hunters : A Romance of the Realms of Ice	R.M. Ballantyne	0.8971
The Giant of the North : Pokings Round the Pole	R.M. Ballantyne	0.8949
An Earthman on Venus (Originally titled "The Radio Man")	Ralph Milne Farley	0.8922

Analyse : Cette liste de livres similaires à *The Aliens* de Murray Leinster, basée sur la similarité thématique, révèle des correspondances intéressantes et parfois inattendues. Comme précédemment, on retrouve une oeuvre de Murray Leinster lui-même, *The Fifth-Dimension Tube*. La présence d'un titre du même auteur renforce l'idée que ces ouvrages partagent des préoccupations thématiques et stylistiques similaires, typiques de la science-fiction d'exploration et d'aventure typique de cet auteur.

La récurrence des romans de R.M. Ballantyne, avec *The Walrus Hunters : A Romance of the Realms of Ice* et *The Giant of the North : Pokings Round the Pole*, est particulièrement frappante. Ballantyne est connu pour ses récits d'aventure et d'exploration dans des environnements extrêmes (comme l'Arctique). Bien que n'étant pas de la science-fiction, ces livres partagent des thèmes de survie, de découverte de territoires inconnus, de confrontation avec la nature sauvage et d'interaction avec des cultures différentes, qui peuvent être des sous-thèmes majeurs dans la science-fiction exploratoire comme *The Aliens*. Le système de recommandation semble donc identifier ces thèmes universels d'aventure et de confrontation avec l'inconnu, transcendant les genres strictes.

Enfin, l'inclusion de *An Earthman on Venus* de Ralph Milne Farley, un classique de la science-fiction pulp, est une correspondance directe en termes de genre et de thématique (exploration spatiale, rencontre avec des civilisations extraterrestres). Cette analyse montre que le système est capable de détecter des thèmes sous-jacents qui relient des oeuvres de genres apparemment très différents, offrant ainsi des recommandations diversifiées et potentiellement surprenantes.

Similarité livre à livre (par thème)

Livre à comparer : *The Strange Case of Dr. Jekyll and Mr. Hyde* de Robert Louis Stevenson

Résultats générés :

Table 7.4 – *Livres similaires à "The Strange Case of Dr. Jekyll and Mr. Hyde" par thèmes*

Titre	Auteur	Similarité
The Strange Case of Dr. Jekyll and Mr. Hyde	Robert Louis Stevenson	0.9132
Phantastes : A Faerie Romance for Men and Women	George MacDonald	0.9120
The Lost Princess of Oz	L. Frank Baum	0.9097
Four Weird Tales	Algernon Blackwood	0.9071

Analyse : Les récits proposés en recommandation, bien que pouvant différer en termes de style et parfois de public cible (notamment "*The Lost Princess of Oz*", orienté jeunesse), présentent une forte proximité thématique. Il est observé que tous explorent des éléments de surnaturel, de fantastique, de mondes parallèles ou de réflexions sur l'identité et les transformations, qui sont des motifs centraux de l'oeuvre de Stevenson. Une ambiance onirique et mystérieuse est également présente dans *Phantastes* et *Four Weird Tales*, ce qui renforce la pertinence de ces rapprochements. Ces résultats soulignent la capacité du système à capter des connexions sémantiques à travers des récits de genres variés partageant des thématiques profondes.

Similarité par requête utilisateur (résumé)

Requête : A dark and gothic novel featuring an immortal creature that drinks the blood of the living, with an eerie atmosphere and themes of seduction, fear, and the supernatural.

Résultats générés :

Table 7.5 – *Livres similaires à la requête sur un roman gothique vampirique*

Titre	Auteur	Similarité
Carmilla	Sheridan Le Fanu	0.7417
The Dunwich Horror	H. P. Lovecraft	0.7076
Dracula	Bram Stoker	0.6979
The Mysteries of Florence	George Lippard	0.6787

Analyse : Les résultats obtenus en réponse à la requête montrent une capacité encourageante du système à identifier des oeuvres relevant de la littérature gothique et vampirique. En première position, *Carmilla* de Sheridan Le Fanu, qui est une référence fondatrice du mythe vampirique, connue pour son atmosphère sensuelle et inquiétante, constitue un choix pertinent. Elle précède même *Dracula* dans le classement, bien que ce dernier apparaisse dans le top 3 avec un score proche de 0.70, confirmant la pertinence sémantique de la requête.

The Dunwich Horror de Lovecraft, bien qu'appartenant davantage à l'horreur cosmique qu'au gothique vampirique pur, partage plusieurs éléments d'ambiance : créatures monstrueuses, surnaturel, et atmosphère pesante. *The Mysteries of Florence*, un roman gothique américain, contient également des motifs de décadence, d'ombre morale et de mystère qui peuvent expliquer sa présence.

Dans l'ensemble, la qualité des premiers résultats témoigne d'une performance du système en contexte de requête sémantique libre. Il est important de souligner que les deux oeuvres emblématiques du genre sont bien identifiées (*Dracula* et *Carmilla*), renforçant ainsi la confiance dans la capacité du moteur de recherche à répondre à des intentions complexes en langage naturel. Cependant, la dispersion thématique des résultats suivants indique qu'un affinement de la requête pourrait encore renforcer la précision de la recherche.

Similarité par requête utilisateur (par liste de thèmes)

Requête (liste de thèmes) :

— Gothic fiction
— Vampirism
— Immortality
— Blood lust
— Seduction and danger
— Fear of the unknown
— Supernatural creatures

Résultats générés :

Table 7.6 – *Livres similaires à la requête thématique sur le vampirisme et le gothique*

Titre	Auteur	Similarité
Carmilla	Sheridan Le Fanu	0.7554
The Tale of Terror : A Study of the Gothic Romance	Edith Birkhead	0.7537
Dracula	Bram Stoker	0.7468
In a Glass Darkly, v. 1/3	Sheridan Le Fanu	0.7452

Analyse : Les résultats de cette recherche thématique montrent une cohérence remarquable avec la requête construite autour de motifs gothiques et vampiriques.

En tête, *Carmilla* de Sheridan Le Fanu s'impose de nouveau comme l'un des récits fondateurs du mythe du vampire féminin. Elle partage un grand nombre de thèmes avec *Dracula*, notamment le vampirisme, la séduction malsaine et une atmosphère lourde de tension. L'oeuvre d'Edith Birkhead, *The Tale of Terror*, bien qu'essai critique plutôt que fiction, explore précisément les fondements historiques, littéraires et symboliques du roman gothique, ce qui justifie sa présence ici.

Dracula figure en troisième position avec un score élevé, confirmant que les thèmes choisis ciblent efficacement le roman. La proximité sémantique avec *In a Glass Darkly* (également de Le Fanu) est également logique : il s'agit d'un recueil de nouvelles fantastiques explorant les hallucinations, les dédoublements de réalité et les manifestations surnaturelles.

Recommandations Basées sur un Profil Utilisateur (par résumé)

Profil utilisateur : Utilisateur ayant lu tous les livres d'*Alice au pays des merveilles* disponibles sur la plateforme

Résultats générés :

Table 7.7 – *Recommandations pour un lecteur d'Alice au pays des merveilles basées sur les résumés des oeuvres*

Titre	Auteur	Similarité
The Nursery "Alice"	Lewis Carroll	0.9160
The Fairy Nightcaps	Frances Elizabeth Barrow	0.7736
Ernest Maltravers Volume 04	Edward Bulwer-Lytton	0.7633
The cats' Arabian nights, or, King Grimalkum	Abby Morton Diaz	0.7624
The Surprising Adventures of the Magical Monarch of Mo and His People	L. Frank Baum	0.7615

Analyse : Les recommandations générées pour ce profil utilisateur révèlent une cohérence thématique remarquable avec l'univers d'*Alice au pays des merveilles*.

Le score exceptionnel de *The Nursery "Alice"* (0.9160) s'explique parfaitement : il s'agit d'une adaptation simplifiée d'*Alice au pays des merveilles* réalisée par Lewis Carroll lui-même pour un public encore plus jeune. Cette recommandation valide la robustesse du système, qui identifie correctement les oeuvres directement liées à l'univers de référence.

Les autres suggestions maintiennent une forte cohérence avec les caractéristiques fondamentales des récits de Lewis Caroll. *The Fairy Nightcaps* de Frances Elizabeth Barrow et *The cats' Arabian nights* s'inscrivent dans la catégorie des contes merveilleux pour enfants, genre auquel appartient *Alice*.

The Surprising Adventures of the Magical Monarch of Mo de L. Frank Baum (créateur du *Magicien d'Oz*) présente une pertinence intéressante. Baum, contemporain de la postérité carrollienne, développe des univers fantastiques similaires où règnent la magie, l'absurde bienveillant et l'aventure merveilleuse. Cette recommandation démontre la capacité du système à identifier des auteurs partageant une vision comparable du conte fantastique.

La présence d'Ernest Maltravers de Bulwer-Lytton constitue la recommandation dont la pertinence est la plus discutable au sein de cette liste. Bien qu'il soit possible que le système ait identifié des liens narratifs ténus, tels que des rebondissements inattendus ou des rencontres avec des personnages excentriques , cette suggestion met surtout en lumière une limite inhérente à l'approche. Il est très probable est que l'histoire suit un personnage s'appelant "Alice Darvil." Le résumé de ce livre contient son nom à de nombreuses reprises et le fait que le profil contient uniquement des livre d'ALice aux pays de merveilles, la moyenne fait sur les livres renforce sans doute la similarité avec le nom Alice qui est trouvé dans le résumé de ce livre.

Dans l'ensemble, ces résultats attestent de la pertinence du système pour identifier des oeuvres appartenant au même registre littéraire (littérature jeunesse fantastique du XIXe siècle) et partageant des codes narratifs similaires comme le merveilleux ou l'absurde. La recommandation capture efficacement l'essence ludique et imaginative qui caractérise l'expérience de lecture d'*Alice au pays des merveilles*. Néanmoins, l'exemple de Ernest Maltravers met en avant le fait que des livres peuvent avoir des rapprochement peu pertinents s'ils ont des noms propres similaires, par exemple.

Recommandations Basées sur un Profil Utilisateur (par thèmes)

Profil utilisateur : Utilisateur ayant lu tous les livres d'*Alice au pays des merveilles* disponibles sur la plateforme

Résultats générés :

Table 7.8 – *Recommandations thématiques pour un lecteur d'Alice au pays des merveilles*

Titre	Auteur	Similarité
The Nursery "Alice"	Lewis Carroll	0.9483
Minnie ; or, The Little Woman : A Fairy Story	-	0.9218
Five Children and It	E. Nesbit	0.9217
The Book of Wonder	Lord Dunsany	0.9185
Mopsa the Fairy	Jean Ingelow	0.9169

Analyse : La recommandation basée sur les thèmes révèle une forte cohérence.

The Nursery "Alice" obtient un bon score, confirmant que l'approche thématique a correctement capturé les thèmes du livre du même auteur.

Les nouvelles recommandations révèlent une spécialisation plus fine vers la littérature fantastique pour enfants. *Five Children and It* d'E. Nesbit, classique de la littérature jeunesse britannique, partage avec *Alice* des thèmes centraux : enfants protagonistes, rencontres avec des créatures magiques, aventures dans des mondes parallèles, leçons de vie.

The Book of Wonder de Lord Dunsany et *Mopsa the Fairy* de Jean Ingelow sont des conte de type "merveilleux". Ces oeuvres explorent des thématiques communes avec Carroll : l'émerveillement face à l'inconnu, des mondes fantastiques, et la transformation des personnages à travers leurs aventures extraordinaires.

Minnie ; or, The Little Woman : A Fairy Story complète cette sélection en maintenant les motifs de la jeune protagoniste féminine évoluant dans un univers où les règles habituelles sont suspendues.

Comparée aux résultats basés sur les résumés, cette approche thématique élimine les recommandations moins directement liées (comme *Ernest Maltravers*) au profit d'oeuvres plus spécifiquement ancrées dans la fantasy pour enfants. Le système démontre ainsi sa capacité à identifier avec précision les macro-thèmes caractéristiques : merveilleux, enfance, créatures fantastiques, mondes alternatifs et quête initiatique, qui constituent le coeur thématique de l'univers Lewis Caroll.

7.2.2 Explicabilité des recommandations via LLM

Un aspect distinctif du système est sa capacité à fournir des explications textuelles pour justifier les recommandations, renforçant ainsi la confiance et la compréhension de l'utilisateur. Lors d'une recommandation livre à livre, l'utilisateur a la possibilité de demander une explication détaillée de la similarité, générée par un LLM local (`gemma3:1b`)

Par exemple, la Figure 6.9, située dans la section 6.8.3, illustre l'explication générée pour la similarité entre "Camille (La Dame aux Camélias)" et "The Alkahest". Comme le montre la figure, le LLM met en évidence des points de convergence thématiques et narratifs, tels que "le fort accent mis sur la famille, en particulier la relation tendue entre une famille riche et respectée et un individu déterminé et obsessionnel" ou encore "les thèmes du devoir, de l'ambition et des conséquences dévastatrices d'un désir incontrôlé". Cette capacité à articuler les raisons sémantiques profondes derrière une suggestion valide la puissance des LLM pour des systèmes de recommandation interprétables, allant au-delà de simples corrélations statistiques. Ce résultat a été généré à partir d'un livre recommandé en utilisant la similarité par résumé.

7.2.3 Conclusion sur la validation des recommandations

Les résultats obtenus à travers cette évaluation qualitative offrent des premières indications encourageantes quant à la pertinence des recommandations générées par le système. Qu'il s'agisse de comparaisons basées sur les résumés ou les thèmes, les livres suggérés présentent une forte cohérence sémantique avec l'oeuvre de référence. Les similarités évoquent des rapprochements pertinents sur les plans narratif, stylistique et émotionnel. Les deux mode de recommandations, par thèmes et par résumés donnent des résultats pertinents bien que parfois différents, ce qui laissent penser que ces deux modes se complètent bien. Il pourrait être intéressant de mettre en place une pondération en utilisant une combinaison des embeddings de résumé et de thèmes pour générer les recommandations.

Les explications générées par le LLM local, comme le montre l'exemple présenté semble assez pertinent et compréhensible.

Toutefois, cette démarche qualitative ne peut se substituer à une véritable validation rigoureuse sur la base de métriques quantitatives standards. De plus, une évaluation directe par des utilisateurs finaux apporterait une mesure bien plus fiable de la pertinence perçue des recommandations en contexte réel d'utilisation. Malheureusement, en raison des contraintes de ce projet (temps, ressources humaines et techniques), une telle validation utilisateur n'a pas pu être mise en oeuvre.

Cela étant dit, les premiers résultats laissent entrevoir un potentiel significatif pour l'approche choisie. Le système est capable de capter les similarités essentielles entre oeuvres littéraires, de plus, les explications fournies par le LLM semblent pertinentes.

Chapitre 8

Conclusion

Ce projet a permis de concevoir et de réaliser avec succès un prototype fonctionnel de système de recommandation de livres personnalisé, exploitant la puissance des grands modèles de langage (LLM). L'objectif initial était de surmonter les limites des approches traditionnelles, souvent restreintes aux métadonnées ou au filtrage collaboratif, en se fondant sur une analyse sémantique profonde du contenu textuel intégral des oeuvres littéraires

La première étape cruciale a été la constitution d'un corpus de plus de 5000 livres numériques, issus principalement du Projet Gutenberg, choisi pour son accès libre au texte intégral. Une robuste pipeline de traitement des données a ensuite été développée. Celle-ci a permis, dans un premier temps, de nettoyer et de préparer les textes bruts, puis d'appliquer une stratégie de résumé hiérarchique pour surmonter les contraintes de longueur des LLM. En s'appuyant sur l'API Gemini, cette pipeline a généré, pour chaque livre, un résumé abstractif et une liste de thèmes pertinents. Ces éléments, plus denses sémantiquement que le texte intégral, ont servi de base à la création de représentations vectorielles (embeddings)

Le développement s'est concrétisé par la mise en place d'une application full-stack à l'architecture microservices conteneurisée avec Docker. Le backend, en FastAPI, et le frontend, en Vue.js, communiquent avec une base de données PostgreSQL qui, grâce aux extensions `pgvector` et `pgai`, fonctionne comme une base de données vectorielle performante. Un aspect novateur de l'architecture est l'automatisation de la génération d'embeddings via un worker dédié (`pgai_worker`) et un service Ollama local, découplant ainsi la gestion des embeddings des opérations de l'application principale.

L'application finalisée offre une expérience utilisateur complète, incluant l'authentification, la gestion de bibliothèques personnelles et, surtout, plusieurs modes de recommandation : par similarité de livre, par requête en langage naturel ou par analyse du profil de lecture de l'utilisateur. Une évaluation qualitative des résultats a démontré la pertinence sémantique des suggestions, confirmant la capacité du système à identifier des liens thématiques, stylistiques et narratifs entre les oeuvres. De plus, pour la recommendation livre à livre, il est possible de demander à un LLM local de générer une explication de la recommandation et des similarités entre le livre original et celui recommandé.

Ce projet a ainsi validé la faisabilité et le potentiel d'une approche basée sur les LLMs pour la recommandation de livres, aboutissant à un système capable de fournir des suggestions plus nuancées et justifiables. Le prototype constitue une fondation solide sur laquelle de nombreuses améliorations peuvent être bâties

8.0.1 Etat par rapport au cahier des charges

Exigence du Cahier des Charges	Statut	Observations et Justifications
Objectifs Principaux		
Extraction et prétraitement du contenu textuel de livres pour analyse sémantique.	**Atteint**	Une pipeline de téléchargement et de nettoyage des livres du Projet Gutenberg a été développée et utilisée pour traiter plus de 5000 oeuvres.
Développement de méthodes d'analyse sémantique (thèmes, tonalité, style).	**Atteint**	Une stratégie de résumé hiérarchique avec un LLM (Gemini) a été implémentée pour générer des résumés abstractifs et des listes de thèmes pour chaque livre.
Création de représentations vectorielles (embeddings) efficaces.	**Atteint**	Les résumés et thèmes ont été transformés en embeddings via un modèle performant (nomic-embed-text) et stockés dans une base de données vectorielle (PostgreSQL + pgvector).
Conception et optimisation d'un moteur de recommandation basé sur le contenu sémantique.	**Atteint**	Un moteur de recommandation calculant la similarité cosinus sur les embeddings a été implémenté, offrant des recommandations par livre, par requête et par profil utilisateur.
Développement d'une interface utilisateur minimale pour tester le système.	**Atteint**	Une application full-stack avec un frontend en Vue.js a été développée, permettant l'exploration du catalogue, la gestion de profil et la visualisation des recommandations.
Exigences fonctionnelles		
Utilisation du texte intégral des oeuvres.	**Atteint**	Le projet se base sur le texte intégral des livres du Projet Gutenberg, nettoyé et traité pour l'analyse.
Analyse sémantique profonde via des modèles de langage avancés.	**Atteint**	L'API Gemini a été utilisée pour une analyse sémantique, et un LLM local (gemma3:1b) pour l'explication des recommandations.
Génération d'explications personnalisées justifiant chaque recommandation.	**Atteint**	Une fonctionnalité permet de générer une explication de la similarité entre deux livres via un LLM local, renforçant la transparence du système.
Traiter des requêtes textuelles libres décrivant les préférences de lecture (si le temps le permet).	**Atteint**	L'application permet aux utilisateurs de formuler une requête en langage naturel qui est convertie en embedding pour trouver des livres correspondants.

Suite à la page suivante

Table 8.1 suite

Exigence du Cahier des Charges	Statut	Observations et Justifications
Capacité à identifier des similarités conceptuelles et thématiques subtiles et pertinentes et bonne qualité des explications générées	**Atteint**	L'analyse qualitative (chapitre 7) a montré la capacité du système à regrouper des oeuvres sur la base de thèmes et d'atmosphères partagés, même entre des genres différents.
Exigences non fonctionnelles		
Génération des recommandations dans un délai raisonnable (< 10 secondes).	**Atteint**	Les requêtes de similarité dans la base de données sont très rapides. La génération d'explication via le LLM local prend environ 30 secondes, ce qui est acceptable pour une fonctionnalité non bloquante.
Traitement d'un corpus d'au moins 1000 livres.	**Atteint**	Le corpus final traité et intégré dans l'application contient 5060 livres.
Livrables		
Pipeline de traitement de texte et de génération d'embeddings.	**Livré**	Le module de téléchargement et traitement de livres a été développé, documenté et rendu public sur GitHub.
Prototype fonctionnel du moteur de recommandation avec documentation.	**Livré**	L'application de recommandation de livres est fonctionnelle, conteneurisée et disponible sur GitHub. Le code est documenté.
Interface utilisateur minimaliste.	**Livré**	Une interface utilisateur complète a été développée avec Vue.js et Vuetify, dépassant l'exigence d'une interface "minimaliste".
Rapport détaillé sur les méthodes, l'architecture et les résultats.	**Livré**	Le présent document constitue le rapport détaillé du projet.
Jeu de données structuré avec les embeddings et analyses générées.	**Livré**	Un dump de la base de données contenant les 5060 livres traités, leurs métadonnées, résumés, thèmes et embeddings est fourni avec le projet.

8.0.2 Évaluation

Les technologies choisies se sont avérées largement satisfaisantes. FastAPI et Python ont fourni un backend performant, avec SQLAlchemy Async assurant une communication asynchrone efficace avec la base de données. Vue.js et Vuetify ont facilité la création d'une interface utilisateur moderne et réactive. L'utilisation de Docker Compose a rationalisé le déploiement et la gestion des microservices. Un succès clé a été l'intégration de TimescaleDB avec pgvector et pgai, transformant efficacement PostgreSQL en une puissante base de données vectorielle capable de gérer les embeddings et de faciliter les recherches

sémantiques. La décision d'utiliser l'API Gemini pour le traitement des LLM, malgré le désir de LLM locaux, s'est avérée pragmatique en raison des contraintes matérielles et de temps, s'avérant rentable et efficace pour le grand corpus. Néanmoins, le projet fournit le nécessaire pour l'utilisation d'un LLM locaux. L'approche de résumé hiérarchique et la stratégie de découpage en morceaux de taille fixe ont été cruciales pour gérer les textes longs et optimiser les interactions avec les LLM, équilibrant la richesse contextuelle et l'efficacité des appels API.

8.1 Perspectives

Plusieurs axes d'amélioration clés ont été identifiés pour affiner la performance et la pertinence du système :

- **Pondération des recommandations entre résumés et thèmes :** Les deux modes de recommandations donnent des résultats intéressants et souvent différents. Il serait intéressant de faire un pondération des thèmes et résumés pour les recommandations. Cela demanderait un meilleure infrastructure de test pour évaluer la pondération idéale.
- **Pondération des profils utilisateurs :** Actuellement, le vecteur représentant le profil d'un utilisateur est une simple moyenne des vecteurs des livres qu'il a lus. Une amélioration consisterait à pondérer cette moyenne par les notes attribuées par l'utilisateur. Cette approche donnerait plus de poids aux livres fortement appréciés.
- **Intégration du filtrage collaboratif :** Le système collecte déjà les notes des utilisateurs, ce qui constitue une base de données idéale pour implémenter des techniques de filtrage collaboratif. En analysant les notes de l'ensemble des utilisateurs, il deviendrait possible de recommander des livres appréciés par des utilisateurs aux goûts similaires, complétant ainsi l'approche actuelle basée sur le contenu.
- **Système de recommandation hybride :** La combinaison du filtrage basé sur le contenu (via les embeddings) et du filtrage collaboratif (via les notes) permettrait de construire un système de recommandation hybride. Ces systèmes sont souvent les plus performants car ils capitalisent sur les forces de chaque approche, palliant leurs faiblesses respectives (par exemple, le ń problème du démarrage à froid ż pour le filtrage collaboratif, ou le manque de personnalisation directe pour le filtrage par contenu pur).
- **Évaluation quantitative et par les utilisateurs :** L'évaluation menée dans ce projet a été de nature qualitative. Une étape future essentielle serait de conduire une évaluation quantitative rigoureuse à l'aide de métriques standards (précision, rappel, F1-score, etc.) sur un jeu de données de test. De plus, la mise en place d'études utilisateurs (tests A/B, questionnaires de satisfaction) fournirait des retours directs et indispensables pour mesurer la pertinence et l'utilité perçues du système en conditions réelles.
- **Amélioration de la génération de résumés et de thèmes par l'évaluation** : Bien que la capacité des LLM à générer des résumés abstraits et des thèmes pertinents ait été qualitativement démontrée, une piste d'amélioration cruciale réside dans la mise en place d'une évaluation systématique et rigoureuse de la qualité de ces sorties. Actuellement, la pertinence est jugée par inspection humaine, ce qui n'est pas scalable. Pour affiner la précision et la cohérence des résumés et des listes de thèmes, il serait essentiel d'intégrer des métriques d'évaluation spécifiques, inspirées par des recherches récentes. Par exemple, des travaux comme "BooookScore" de Chang et al. [9] proposent des métriques automatiques basées sur des LLM pour évaluer la cohérence des résumés en détectant divers types d'erreurs. L'adoption de telles méthodes permettrait d'optimiser les prompts en utilisant les résultats de cette évaluation pour itérer et améliorer les prompts utilisés et de comparer les modèles sur la qualité des résumés et des thèmes générés. Ce processus n'a pas pu être effectué par contrainte de temps mais aussi matérielle, un telle évaluation avec affinage demandant un gros coût d'utilisation des LLMs.
- **Amélioration de l'application et de la sécurité :** Au-delà des algorithmes de recommandation, des améliorations fonctionnelles pourraient être apportées pour rendre l'application plus robuste et prête pour une mise en production. La gestion de l'authentification pourrait être ren-

forcée par l'implémentation d'un système de jetons de rafraîchissement (*refresh tokens*) pour des sessions plus longues et sécurisées. Sur le plan de l'expérience utilisateur, l'intégration de la pagination ou du défilement infini pour l'affichage des listes de livres améliorerait la performance et la fluidité de la navigation. Enfin, des stratégies de mise en cache (*caching*) pour les requêtes fréquentes, comme les livres les plus populaires ou les détails d'un auteur, pourraient être mises en place pour réduire la charge sur la base de données et accélérer les temps de réponse de l'application. De plus, pour une base de donnée plus importante, il deviendrait alors pertinent d'utiliser de l'indexation pour les embeddings.

8.1.1 Difficultés rencontrées

Le projet, malgré son succès final, n'a pas été exempt de défis inhérents à l'exploitation des technologies de pointe. Une difficulté majeure a résidé dans les contraintes matérielles et temporelles liées à l'utilisation des LLM. Le déploiement et l'exécution de modèles de langage de grande taille en local exigent une puissance de calcul significative, notamment des GPU avec une mémoire vidéo très importante, ce qui n'était pas toujours disponible. Par exemple, la génération d'un résumé prenait en moyenne cinq minutes avec un modèle local, ce qui aurait prolongé le temps de traitement total pour les 5060 livres à dix-sept jours consécutifs. Cette réalité a motivé le choix d'opter pour l'API Gemini de Google, afin de contourner ces limitations et de respecter le budget et les délais du projet. De plus, la variabilité des données sources, notamment les multiples éditions d'un même ouvrage dans le Projet Gutenberg avec des résumés initiaux parfois distincts, a constitué un défi pour l'évaluation qualitative de la pertinence des recommandations, bien que cela ait finalement validé la robustesse sémantique du système.

8.2 Bilan personnel

8.2.1 Connaissances acquises

Ce projet a offert de précieuses expériences d'apprentissage, notamment en :

- Acquisition d'une compréhension approfondie de la manière d'exploiter les LLMs pour l'analyse de texte complexe, allant au-delà de la simple correspondance de mots-clés pour capturer les nuances thématiques et stylistiques
- Implémentation de bases de données vectorielles : L'expérience pratique avec pgvector et pgai dans PostgreSQL a démontré la puissance de la combinaison des bases de données relationnelles traditionnelles avec les capacités vectorielles pour les applications d'IA.
- Exploitation et gestion des grands modèles de langage (LLM)
  - **LLM en environnement local** : Capacité à déployer, configurer et interagir avec des LLM directement sur des infrastructures locales. Cela inclut la maîtrise des plateformes comme Ollama, permettant d'exécuter des modèles d'embedding tels que 'nomic-embed-text', reconnu pour ses performances élevées dans la génération d'embeddings textuels et la gestion efficace de l'inférence des LLM en local.
  - **LLM en ligne via API** : Maîtrise de l'utilisation de services LLM basés sur le cloud, comme l'API Gemini, pour exploiter leurs capacités de compréhension et de génération de langage.
- **Architecture de microservices** : Renforcement des principes de modularité et de séparation des préoccupations dans un environnement conteneurisé, essentiel pour des applications maintenables et évolutives
- **Ingénierie des prompts** : Compréhension du rôle critique des prompts bien formulés pour guider le comportement des LLM et atteindre la qualité de sortie souhaitée pour la synthèse et l'extraction de thèmes.

8.2.2 Vécu du travail de Bachelor

Réaliser un projet de cette envergure, sur une période aussi longue et en toute autonomie, a été une première pour moi. J'ai trouvé qu'il s'agissait d'une expérience particulièrement enrichissante et satisfaisante, bien plus que les projets plus courts menés au cours de mon Bachelor. Le fait de pouvoir travailler un sujet en profondeur et de le concrétiser de A à Z a été extrêmement gratifiant. Ce processus a également été une opportunité unique de découvrir et d'explorer de nombreux outils techniques, d'approfondir mes compétences existantes et de cimenter mes connaissances théoriques dans des applications concrètes. Bien que le travail ait été réalisé de manière indépendante, le suivi régulier de la part de ma professeure a été très bénéfique. Ses conseils avisés et les points réguliers ont contribué à maintenir une bonne dynamique de travail et une vision claire des objectifs. Cette structure m'a permis de progresser efficacement et de surmonter les défis inhérents à un projet d'une telle envergure.

8.2.3 Remerciements

Je remercie la professeure Fatemi Nastaran pour sa supervision assidue et ses conseils avisés, ainsi que Christopher Meier, pour son soutien précieux. Je remercie mon père Giampaolo Tranchida et ma mère Francesca Tranchida pour la relecture de ce travail et leurs suggestions. Mes remerciements s'adressent également à tous mes collègues de la HEIG-VD et amis pour leur contribution technique, notamment via le test de l'application et leurs retours pertinents, ainsi que pour leur soutien moral indispensable.

Bibliographie

[1] Charu C. Aggarwal. *Recommender Systems. The Textbook*. Springer Cham, 2016. ISBN : 978-3-319-29659-3. DOI : 10.1007/978-3-319-29659-3.

[2] Xavier Amatriain. *Prompt Design and Engineering : Introduction and Advanced Methods*. 2024. arXiv : 2401.14423 [cs.SE]. URL : https://arxiv.org/abs/2401.14423.

[3] *APIs | Open Library*. URL : https://openlibrary.org/developers/api (visité le 21/05/2025).

[4] Liam Barkley et Brink van der Merwe. *Investigating the Role of Prompting and External Tools in Hallucination Rates of Large Language Models*. 2024. arXiv : 2410.19385 [cs.CL]. URL : https://arxiv.org/abs/2410.19385.

[5] Jason Brownlee. *A Gentle Introduction to the Bag-of-Words Model*. en-US. Oct. 2017. URL : https://www.machinelearningmastery.com/gentle-introduction-bag-words-model/ (visité le 15/05/2025).

[6] Robin Burke. « Knowledge-Based Recommender Systems ». In : *Encyclopedia of library and information systems* 69 (mai 2000).

[7] Hongliu Cao. *Recent advances in text embedding : A Comprehensive Review of Top-Performing Methods on the MTEB Benchmark*. arXiv:2406.01607 [cs] version : 2. Juin 2024. DOI : 10.48550/arXiv.2406.01607. URL : http://arxiv.org/abs/2406.01607 (visité le 30/04/2025).

[8] *Challenges of the Recommendation system | by Nidhi Shukla | Medium*. URL : https://medium.com/@nidhishukla2023/challenges-of-the-recommendation-system-f2406b3f2232 (visité le 23/05/2025).

[9] Yapei Chang et al. *BooookScore : A systematic exploration of book-length summarization in the era of LLMs*. arXiv:2310.00785 [cs]. Avr. 2024. DOI : 10.48550/arXiv.2310.00785. URL : http://arxiv.org/abs/2310.00785 (visité le 08/03/2025).

[10] *CMU Book Summary Dataset*. URL : https://www.cs.cmu.edu/~dbamman/booksummaries.html (visité le 23/05/2025).

[11] *Cosine similarity*. en. Page Version ID : 1287662903. Avr. 2025. URL : https://en.wikipedia.org/w/index.php?title=Cosine_similarity&oldid=1287662903 (visité le 23/05/2025).

[12] Jacob Devlin et al. *BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv : 1810.04805 [cs.CL]. URL : https://arxiv.org/abs/1810.04805.

[13] *Dot product*. en. Page Version ID : 1277638023. Fév. 2025. URL : https://en.wikipedia.org/w/index.php?title=Dot_product&oldid=1277638023 (visité le 11/03/2025).

[14] *embeddings-benchmark/mteb*. original-date : 2022-04-05T08:25:47Z. Mai 2025. URL : https://github.com/embeddings-benchmark/mteb (visité le 23/05/2025).

[15] *Euclidean distance*. en. Page Version ID : 1277294848. Fév. 2025. URL : https://en.wikipedia.org/w/index.php?title=Euclidean_distance&oldid=1277294848 (visité le 11/03/2025).

[16] Inc Facebook. *fastText : Library for fast text representation and classification*. 2016. URL : https://github.com/facebookresearch/fastText.

[17] *Factorisation matricielle | Machine Learning*. fr-x-mtfrom-en. URL : https://developers.google.com/machine-learning/recommendation/collaborative/matrix?hl=fr (visité le 16/05/2025).

[18] J.R. Firth. « A synopsis of linguistic theory 1930-1955 ». In : *Studies in Linguistic Analysis*. Reprinted in F.R. Palmer (Ed.), Selected Papers of J.R. Firth 1952-1959. London : Longman (1968). 1957, p. 1-32.

[19] *Frontier AI LLMs, assistants, agents, services | Mistral AI*. en. URL : https://mistral.ai/ (visité le 23/05/2025).

[20] *GloVe : Global Vectors for Word Representation*. URL : https://nlp.stanford.edu/projects/glove/ (visité le 15/05/2025).

[21] *Google Books APIs | Google for Developers*. URL : https://developers.google.com/books?hl=fr (visité le 23/05/2025).

[22] Google LLC. *Generate structured output (like JSON and enums) using the Gemini API*. 2025. URL : https://ai.google.dev/gemini-api/docs/structured-output?hl=fr.

[23] Kartik Gupta. « Nomic Embeddings A cheaper and better way to create embeddings ». In : *Medium* (mars 2024).

[24] Gutendex API. *Gutendex : A Free and Open API for Project Gutenberg Books*. 2024. URL : https://gutendex.com/.

[25] Chia-Hui Hsieh et al. « Design of a Serendipity-Incorporated Recommender System ». In : *Applied Sciences* 14.4 (fév. 2024), p. 821. DOI : 10.3390/app14040821. URL : https://www.mdpi.com/2079-9292/14/4/821.

[26] Lei Huang et al. « A Survey on Hallucination in Large Language Models : Principles, Taxonomy, Challenges, and Open Questions ». In : *ACM Transactions on Information Systems* 43.2 (jan. 2025), p. 1-55. ISSN : 1558-2868. DOI : 10.1145/3703155. URL : http://dx.doi.org/10.1145/3703155.

[27] Scott Isaacson et John Sebastian. *Book Recommendations on GoodReads.com*. Rapp. tech. CS229 Project. Stanford University, 2008. URL : https://cs229.stanford.edu/proj2008/IsaacsonSebastian-GoodReadsRecommendations.pdf.

[28] Gareth B. Johnson. *Gutendex : Web API for Project Gutenberg ebook metadata*. Accessed 2025-07-23. 2025. URL : https://github.com/garethbjohnson/gutendex.

[29] Michael Kelley. *Goodreads Launches Book Recommendation Feature*. Library Journal. Oct. 2011. URL : https://www.libraryjournal.com/story/goodreads-launches-book-recommendation-feature.

[30] Denis Kotkov, Shuaiqiang Wang et Jari Veijalainen. « A survey of serendipity in recommender systems ». In : *Knowledge-Based Systems* 111 (2016), p. 180-192. ISSN : 0950-7051. DOI : https://doi.org/10.1016/j.knosys.2016.08.014. URL : https://www.sciencedirect.com/science/article/pii/S0950705116302763.

[31] Matev Kunaver et Toma Porl. « Diversity in recommender systems A survey ». In : *Knowledge-Based Systems* 123 (2017), p. 154-162. ISSN : 0950-7051. DOI : https://doi.org/10.1016/j.knosys.2017.02.009. URL : https://www.sciencedirect.com/science/article/pii/S0950705117300680.

[32] Quoc V. Le et Tomas Mikolov. *Distributed Representations of Sentences and Documents*. 2014. arXiv : 1405.4053 [cs.CL]. URL : https://arxiv.org/abs/1405.4053.

[33] Copy link et al. *The history of Amazon's recommendation algorithm*. en. Section : News and features. Nov. 2019. URL : https://www.amazon.science/the-history-of-amazons-recommendation-algorithm (visité le 23/05/2025).

[34] Qijiong Liu et al. « ONCE : Boosting Content-based Recommendation with Both Open- and Closed-source Large Language Models ». In : *Proceedings of the 17th ACM International Conference on Web Search and Data Mining (WSDM '24)*. ACM, 2024, p. 452-461. DOI : 10.1145/3616855.3635845. URL : https://doi.org/10.1145/3616855.3635845.

[35] Yinhan Liu et al. *RoBERTa : A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv : 1907.11692 [cs.CL]. URL : https://arxiv.org/abs/1907.11692.

[36] Pasquale Lops, Marco De Gemmis et Giovanni Semeraro. « Content-based recommender systems : State of the art and trends ». In : *Recommender systems handbook*. Springer, 2011, p. 73-105.

[37] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. arXiv:1301.3781 [cs]. Sept. 2013. DOI : 10.48550/arXiv.1301.3781. URL : http://arxiv.org/abs/1301.3781 (visité le 15/05/2025).

[38] Jacob Murel et Eda Kavlakoglu. *What is content-based filtering ? | IBM*. en. Mars 2024. URL : https://www.ibm.com/think/topics/content-based-filtering (visité le 11/03/2025).

[39] Natural Language Toolkit. *NLTK : Natural Language Toolkit*. URL : https://www.nltk.org/ (visité le 16/07/2025).

[40] Ollama. *Structured outputs*. URL : https://ollama.com/blog/structured-outputs (visité le 16/07/2025).

[41] Ollama Team. *nomic-embed-text*. 2025. URL : https://ollama.com/library/nomic-embed-text.

[42] pgvector team. *Indexes — pgvector documentation*. 2025. URL : https://github.com/pgvector/pgvector#indexes.

[43] pgvector team. *pgvector documentation*. 2025. URL : https://github.com/pgvector/pgvector.

[44] *Présentation de la génération de candidats | Machine Learning | Google for Developers*. fr-x-mtfrom-en. URL : https://developers.google.com/machine-learning/recommendation/overview/candidate-generation?hl=fr (visité le 23/05/2025).

[45] *Project Gutenberg*. en. URL : https://www.gutenberg.org/ (visité le 23/05/2025).

[46] Project Gutenberg. *Offline Catalogs*. Accessed 2025-07-23. 2025. URL : https://www.gutenberg.org/ebooks/offline_catalogs.html.

[47] *Prompt Engineering Guide*. en. Avr. 2025. URL : https://www.promptingguide.ai/techniques (visité le 23/05/2025).

[48] Jing Qin. « A Survey of Long-Tail Item Recommendation Methods ». In : *Wireless Communications and Mobile Computing* 2021.1 (2021), p. 7536316. DOI : https://doi.org/10.1155/2021/7536316. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/7536316. URL : https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/7536316.

[49] *Qu'est-ce que l'apprentissage zero-shot ? | IBM*. URL : https://www.ibm.com/fr-fr/think/topics/zero-shot-learning (visité le 23/05/2025).

[50] Alec Radford et al. *Improving Language Understanding by Generative Pre-Training*. Rapp. tech. OpenAI, juin 2018. URL : https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[51] Rakuten Today. *How Kobo is using big data to keep booklovers reading*. Nov. 2016. URL : https://rakuten.today/blog/kobo-big-data.html.

[52] Sebastián Ramírez. *FastAPI : The Fast Modern Python Web Framework*. 2024. URL : https://fastapi.tiangolo.com/.

[53] Francesco Ricci, Lior Rokach et Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[54] Stephen Robertson. « Understanding Inverse Document Frequency : On Theoretical Arguments for IDF ». In : *Journal of Documentation - J DOC* 60 (oct. 2004), p. 503-520. DOI : 10.1108/00220410410560582.

[55] Deepjyoti Roy et Mala Dutta. « A systematic review and research perspective on recommender systems ». In : *Journal of Big Data* 9.1 (mai 2022), p. 59. ISSN : 2196-1115. DOI : 10.1186/s40537-022-00592-5. URL : https://doi.org/10.1186/s40537-022-00592-5 (visité le 30/04/2025).

[56] Avthar Sewrathan. *How to Automate Embedding Creation and Sync for RAG in PostgreSQL*. URL : https://www.tigerdata.com/blog/postgresql-as-a-vector-database-using-pgvector.

[57] Supabase. *Automatic embeddings*. 2025. URL : https://supabase.com/docs/guides/ai/automatic-embeddings.

[58] *Système de recommandation Amazon Personalize Amazon Web Services*. URL : https://aws.amazon.com/fr/personalize/ (visité le 23/05/2025).

[59] The SQLAlchemy Project. *Asynchronous I/O (asyncio) — SQLAlchemy 2.0 Documentation*. 2025. URL : https://docs.sqlalchemy.org/en/20/orm/extensions/asyncio.html.

[60] TigerData Documentation Team. *Installation Docker*. 2025. URL : https://docs.tigerdata.com/self-hosted/latest/install/installation-docker/.

[61] Timescale. *Automate AI embedding with pgai Vectorizer*. 2025. URL : https://github.com/timescale/pgai/blob/released/docs/vectorizer/overview.md.

[62] Timescale. *pgai architecture overview*. URL : https://github.com/timescale/pgai#basic-architecture-the-system-consists-of-an-application-you-write-a-postgresql-database-and-stateless-vectorizer-workers.

[63] Timescale. *Pgai : Giving PostgreSQL Developers AI Engineering Superpowers*. 2025. URL : https://www.tigerdata.com/blog/pgai-giving-postgresql-developers-ai-engineering-superpowers.

[64] Timescale. *PostgreSQL + TimescaleDB : 1000x Faster Queries, 90% Data Compression, and Much More*. 2025. URL : https://www.tigerdata.com/blog/postgresql-timescaledb-1000x-faster-queries-90-data-compression-and-much-more.

[65] Timescale. *RAG Is More Than Just Vector Search*. 2025. URL : https://www.tigerdata.com/blog/rag-is-more-than-just-vector-search.

[66] Timescale. *timescale/pgai : Power your RAG and Agentic applications with PostgreSQL*. 2025. URL : https://github.com/timescale/pgai.

[67] Hugo Touvron et al. *LLaMA : Open and Efficient Foundation Language Models*. 2023. arXiv : 2302.13971 [cs.CL]. URL : https://arxiv.org/abs/2302.13971.

[68] Ashish Vaswani et al. « Attention is All you Need ». In : *Advances in Neural Information Processing Systems*. T. 30. Curran Associates, Inc., 2017. URL : https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (visité le 15/05/2025).

[69] Yan Wang et al. *Enhancing Recommender Systems with Large Language Model Reasoning Graphs*. 2024. arXiv : 2308.10835 [cs.IR]. URL : https://arxiv.org/abs/2308.10835.

[70] *Welcome to Open Library | Open Library*. URL : https://openlibrary.org/ (visité le 23/05/2025).

[71] *What is the Cold Start Problem in Recommender Systems ?* en. Fév. 2025. URL : https://www.freecodecamp.org/news/cold-start-problem-in-recommender-systems/ (visité le 23/05/2025).

[72] *What is the Netflix Prize competition and its relevance to recommender systems ?* URL : https://milvus.io/ai-quick-reference/what-is-the-netflix-prize-competition-and-its-relevance-to-recommender-systems (visité le 23/05/2025).

[73] wumanandpat. *Project Gutenberg Catalog of 19th Century Authors*. 2023. URL : https://www.kaggle.com/datasets/wumanandpat/project-gutenberg-catalog-of-19th-century-authors.

[74] Shunyu Yao et al. *ReAct : Synergizing Reasoning and Acting in Language Models*. 2023. arXiv : 2210.03629 [cs.CL]. URL : https://arxiv.org/abs/2210.03629.

[75] Shunyu Yao et al. « Tree of Thoughts : Deliberate Problem Solving with Large Language Models ». In : *Advances in Neural Information Processing Systems*. Sous la dir. d'Alice H. Oh et al. T. 36. 2023, p. 24094-24115. URL : https://proceedings.neurips.cc/paper_files/paper/2023/file/1f192a1c6b33030908a069a217c42703-Paper-Conference.pdf.

[76] Gokul Yenduri et al. *Generative Pre-trained Transformer : A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions*. 2023. arXiv : 2305.10435 [cs.CL]. URL : https://arxiv.org/abs/2305.10435.

[77] Aakas Zhiyuli et al. « BookGPT : A General Framework for Book Recommendation Based on a Large Language Model ». In : *arXiv preprint arXiv:2305.15673* (2023). arXiv : 2305.15673 [cs.IR]. URL : https://arxiv.org/abs/2305.15673.

[78] Vassilis N. Ioannidis Zichen Wang. *How AWS uses graph neural networks to meet customer needs*. en. Mars 2022. URL : https://www.amazon.science/blog/how-aws-uses-graph-neural-networks-to-meet-customer-needs (visité le 23/05/2025).

Annexes

Annexe A

Exemple de résultat de processing


```
{
  "summary": "Robert Walton, an ambitious English explorer, embarks on a perilous journey to the
  ↪  Arctic, seeking knowledge and glory. He rescues Victor Frankenstein, who recounts his life
  ↪  story. Victor, born in Geneva, Switzerland, enjoyed a happy childhood with his loving parents
  ↪  and his adopted cousin and future wife, Elizabeth Lavenza. He also formed a close friendship
  ↪  with Henry Clerval. Victor's intellectual curiosity led him to the study of alchemy and
  ↪  eventually to the University of Ingolstadt, where he became obsessed with uncovering the secret
  ↪  of life. Driven by ambition, Victor succeeds in creating a sentient being, a gigantic monster,
  ↪  through a horrifying process. However, upon animating his creation, Victor is repulsed by its
  ↪  ghastly appearance and abandons it in horror. \n\nThe creature, left to fend for itself,
  ↪  experiences immense suffering and isolation. He learns about human society by observing the De
  ↪  Lacey family and gains language and empathy. Rejected by them due to his appearance, the
  ↪  creature's benevolence turns to despair and rage. He encounters Victor's younger brother,
  ↪  William, and murders him in a fit of anger, framing Justine Moritz, a young woman in the
  ↪  Frankenstein household, for the crime. Justine is unjustly convicted and executed, a tragedy
  ↪  that plunges Victor into guilt and despair. \n\nThe creature confronts Victor in the Alps,
  ↪  confessing to William's murder and Justine's framing. Driven by his own suffering and rejection,
  ↪  the creature demands that Victor create a female companion for him, threatening further violence
  ↪  if Victor refuses. Victor reluctantly agrees, embarking on a journey to Scotland with Clerval to
  ↪  fulfill this task. While Victor begins constructing the female creature in the Orkney Islands,
  ↪  he grapples with the immense moral implications of his actions. Ultimately, he destroys the
  ↪  half-finished creation, fearing the propagation of misery. The enraged creature vows revenge,
  ↪  threatening Victor's wedding night. \n\nTragically, the creature murders Elizabeth on Victor's
  ↪  wedding night. Consumed by grief and a thirst for vengeance, Victor pursues the creature across
  ↪  the globe, enduring immense hardship. He narrates his story to Robert Walton, and his pursuit
  ↪  ultimately leads him to the Arctic. Victor dies from exhaustion and torment, but not before
  ↪  urging Walton to continue the chase. The creature appears to Walton, expressing profound remorse
  ↪  and despair over his existence and his creator's death. He reveals his intention to immolate
  ↪  himself in the Arctic flames, seeking an end to his suffering and ensuring no other such being
  ↪  is created.",
  "title": "Frankenstein; Or, The Modern Prometheus",
  "authors": [
    {
      "birth_year": 1797,
      "death_year": 1851,
      "name": "Shelley, Mary Wollstonecraft"
    }
  ],
  "themes": [
    "creation",
    "abandonment",
    "revenge",
    "prejudice",
    "ambition",
    "guilt",
    "isolation",
```

```
    "nature vs nurture",
    "supernatural",
    "tragedy"
  ],
  "gutenberg_id": 84
}
```

Annexe B

Métadonnées du projet Gutenberg

B.1 Champs de Métadonnées du Projet Gutenberg

Table B.1 – *Description des champs de métadonnées du Projet Gutenberg*
Source : Adapté et traduit de Gutenberg.org

Champ (Field)	Explication
Creator	La personne ou l'entité qui a écrit le livre. Peut inclure des auteurs, contributeurs, éditeurs, illustrateurs et autres rôles.
Title	Le titre du livre. Un sous-titre est également inclus, s'il est disponible. Les sous-titres sont généralement sur une nouvelle ligne.
Alternate title	Tout titre alternatif, si connu. Peut être utilisé pour les titres traduits, le cas échéant.
Note	Notes ajoutées par l'équipe de catalogueurs. Les notes fournissent des informations supplémentaires au lecteur.
Reading level	Une estimation numérique de la facilité ou de la difficulté de lecture du livre. Utilise la méthode Flesch-Kincaid, qui examine des éléments tels que la longueur des phrases et le nombre de syllabes dans les mots. Un article Wikipédia décrit cette méthode : https://en.wikipedia.org/wiki/Flesch-Kincaid_readability_tests.
Original publication	Informations de publication de la ou des sources imprimées utilisées pour la numérisation, lorsque connues. Le Projet Gutenberg ne cherche pas à correspondre exactement aux sources imprimées et peut utiliser plusieurs sources pour un livre numérique donné. Si la provenance exacte d'un livre numérique est importante, une comparaison personnelle avec les sources imprimées connues peut être nécessaire. Il est parfois possible de contacter les producteurs via la ligne de Crédits. Ces informations sont disponibles pour les ouvrages publiés par le Projet Gutenberg depuis environ 2022.
Credits	Mentionne généralement qui a produit ce livre numérique et quelle(s) ressource(s) ont fourni la publication originale pour la numérisation.

Suite à la page suivante

Table B.1 – *Description des champs de métadonnées du Projet Gutenberg (suite)*

Champ (Field)	**Explication en français**
Language	La langue principale, ainsi que les autres langues utilisées dans le livre.
Category	Texte ou Son.
LoC Class	Classification de la Bibliothèque du Congrès (Library of Congress). Le Projet Gutenberg n'utilise que les 2 premières lettres. Cette classification peut aider à trouver des livres dans un domaine d'intérêt spécifique. Les enregistrements de classe LoC sont généralement ajoutés aux nouveaux livres numériques quelques semaines après leur publication.
Subject	Un ou plusieurs sujets. Les vedettes-matière sont généralement ajoutées aux nouveaux livres numériques quelques semaines après leur publication. Sélectionner un sujet permet d'accéder à une page de recherche pour d'autres livres numériques partageant le même sujet. C'est un excellent moyen de parcourir la collection du Projet Gutenberg.
Automated summaries	En septembre 2024, des résumés générés automatiquement ont été ajoutés pour la plupart des livres. Les nouveaux livres recevront des résumés environ une fois par mois. Ces résumés sont basés sur approximativement les 12 000 premiers mots des livres et sont uniquement en anglais, quelle que soit la langue du livre. Ils pourront être mis à jour à l'avenir avec l'amélioration des technologies d'automatisation.
EBook-No.	C'est le numéro de catalogue du Projet Gutenberg (ou numéro d'acquisition). Chaque livre numérique du PG en possède un différent.
Release Date	C'est la date à laquelle le livre numérique a été publié pour la première fois par le Projet Gutenberg. Les publications du Projet Gutenberg ne correspondent pas exactement à une édition imprimée, elles sont donc considérées comme de nouvelles publications. Les livres numériques du Projet Gutenberg sont fréquemment mis à jour, et les dates de ces mises à jour sont généralement indiquées dans le livre numérique lui-même.

Suite à la page suivante

Table B.1 – *Description des champs de métadonnées du Projet Gutenberg (suite)*

Champ (Field)	Explication en français
Copyright Status	Indique si le livre numérique est protégé par le droit d'auteur aux États-Unis. Si vous ne résidez pas aux États-Unis, vous devez vérifier les lois sur le droit d'auteur de votre pays avant de télécharger un livre numérique ! PG ne connaît pas le statut du droit d'auteur de ses livres numériques dans d'autres pays que les É.-U. Vous pouvez télécharger un livre numérique protégé pour votre usage personnel, mais devez contacter le détenteur des droits si vous souhaitez le redistribuer. Plus de 95% des livres numériques du Projet Gutenberg ne sont pas protégés par le droit d'auteur aux É.-U., généralement parce que la durée de leur protection a expiré. Il s'agit souvent d'ouvrages plus anciens. Voir la politique de développement des collections pour plus de détails.
Downloads	Le nombre approximatif de téléchargements au cours des 30 derniers jours (mis à jour quotidiennement).

B.2 Formats disponibles

Table B.2 – *Principaux formats de fichiers disponibles sur le Projet Gutenberg*
Source : Gutenberg.org

Format	Description
Lire en ligne (web)	HTML5 standard (HTML/XHTML hérités mis à jour). Consultable dans n'importe quel navigateur web.
EPUB3	Standard actuel pour les livres numériques (publication électronique version 3) par l'IDPF. Extension de fichier : `.epub`. Utilisé par la plupart des tablettes et liseuses.
EPUB (anciennes liseuses)	EPUB version 2 pour les appareils/logiciels plus anciens. Une version ń EPUB (sans images, anciennes liseuses) ż est également proposée pour réduire la taille du fichier.
Kindle	Format KF8 pour la série Amazon Kindle. Si des images sont présentes, des versions avec et sans images sont proposées.
Anciens Kindles	Format MOBI pour les anciens Kindles qui ne prennent pas en charge le KF8 ou les logiciels actuels.
Texte brut UTF-8	Fichier texte avec longueur de ligne fixe. Consultable dans les navigateurs ou les éditeurs de texte. Meilleur rendu avec une police à espacement fixe. Presque chaque livre numérique est disponible dans ce format. L'UTF-8 (Unicode) est presque toujours disponible. D'autres encodages (Latin1, ASCII) peuvent se trouver sous "Plus de fichiers...".
Télécharger HTML (zip)	Un paquet HTML complet sous forme de fichier ZIP, incluant HTML, images, et autres fichiers nécessaires pour une consultation hors ligne.