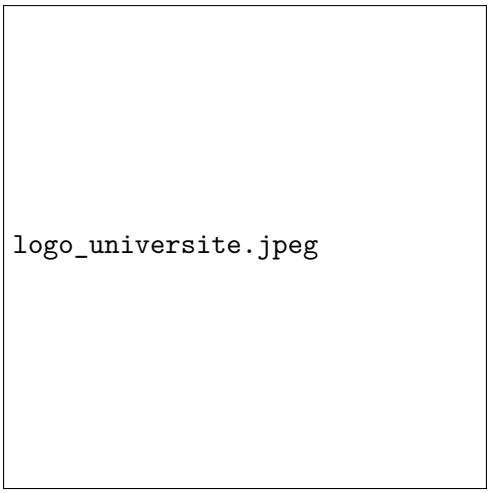

Rapport de Projet TER

Système d'Orientation Universitaire par Retrieval-Augmented Generation

Réalisé par :

Othmane El Farouqy
Fatine Kassabi

Encadré par :

Nicolas Wicker

Année universitaire : 2025–2026

Contents

1 Introduction

L'orientation académique constitue l'un des défis les plus complexes auxquels les étudiants français sont confrontés. Chaque année, plus de 17 000 formations sont recensées sur Parcoursup, la plateforme nationale d'admission dans l'enseignement supérieur, et plus de 930 000 lycéens et étudiants en réorientation doivent y formuler leurs voeux. Naviguer dans cet espace pléthorique d'offres, souvent mal décrites ou difficiles à comparer, constitue une tâche cognitive exigeante que la plupart des outils actuels ne facilitent pas suffisamment.

Les données officielles illustrent l'ampleur du problème. Selon le Ministère de l'Enseignement Supérieur et de la Recherche, en 2023, 14,6 % des bacheliers ne disposaient pas de formation définitive au 1er juillet, et le taux de réorientation en fin de première année universitaire atteignait 22 %. Ces chiffres ne reflètent pas uniquement une inadéquation entre offre et demande : ils témoignent d'un manque structurel d'outils d'aide à la décision personnalisés, capables de prendre en compte la singularité de chaque profil étudiant.

1.1 Contexte et problématique

Les moteurs de recherche classiques reposent sur des correspondances exactes de mots-clés, via des mécanismes tels que TF-IDF ou BM25 [6]. Ces approches, bien qu'efficaces dans leur domaine, présentent deux limites fondamentales dans notre contexte applicatif.

La première est l'**incapacité sémantique** : un profil mentionnant "apprentissage automatique" ne retrouvera pas une formation intitulée "Statistiques et Intelligence Artificielle", même si les deux couvrent le même domaine. Le système ne comprend pas le sens des termes, il effectue une correspondance de chaînes de caractères.

La seconde est l'**absence de personnalisation** : deux étudiants aux mots-clés similaires reçoivent les mêmes résultats, sans prise en compte de leur projet professionnel détaillé, de leur parcours académique ou de leurs contraintes logistiques (ville, budget, niveau d'entrée).

La problématique centrale de ce projet est donc la suivante :

> Comment construire un système capable de comprendre sémantiquement le profil complet d'un étudiant et de recommander des formations pertinentes, tout en respectant des contraintes structurelles strictes (ville, budget, niveau) et en minimisant les erreurs factuelles générées par le système ?

Pour répondre à cette question, nous nous appuyons sur l'architecture RAG (*Retrieval-Augmented Generation*) [1], qui combine une recherche sémantique vectorielle dans une base de formations réelles et un modèle de langage (LLM) pour générer des recommandations justifiées.

1.2 Méthodologie et organisation

Le projet s'articule en plusieurs grandes étapes, toutes automatisées et documentées afin de garantir la reproductibilité :

1. **Collecte et enrichissement** des données Parcoursup via l'API officielle, avec un pipeline d'enrichissement automatique des descriptions de formations ;
2. **Vectorisation sémantique** des profils étudiants et des formations via le modèle d'embedding multilingue `paraphrase-multilingual-MiniLM-L12-v2` ;
3. **Indexation vectorielle** dans ChromaDB via l'algorithme HNSW [3], avec une structure de recherche approximative efficace ;
4. **Filtrage déterministe** par contraintes strictes (ville, budget, niveau de formation), extrait automatiquement du profil ;

5. **Génération des recommandations** via un LLM configurable (GPT-4o-mini par défaut), guidé par un prompt système contrôlé afin de réduire les hallucinations.

1.3 Plan du rapport

Afin de présenter ce travail de manière progressive et rigoureuse, ce rapport est organisé comme suit :

1. Un **état de l'art** des méthodes de recherche sémantique, retraçant l'évolution depuis les approches par mots-clés jusqu'aux architectures RAG modernes ;
2. Les **fondements théoriques** du système : embeddings, similarité cosinus, algorithme HNSW et modèle probabiliste RAG ;
3. L'**architecture technique** et les choix d'implémentation, documentés et justifiés par rapport aux alternatives ;
4. Une **analyse expérimentale** sur des scénarios d'étudiants réalistes, avec tableaux de résultats et tests de robustesse ;
5. Une **discussion** des limites identifiées, des biais potentiels et des perspectives d'amélioration ;
6. La **conclusion** et les travaux futurs envisagés.

2 Développement

Cette partie détaille l'ensemble du travail réalisé. Chaque étape est présentée progressivement, en expliquant non seulement ce qui a été fait, mais également pourquoi et ce que cela permet de comprendre.

Ce développement suit une logique en quatre temps : nous situons d'abord notre approche dans l'état de l'art pour en établir la légitimité, puis nous formalisons les concepts mathématiques qui en constituent les fondations, avant de décrire l'architecture technique et les extraits de code clés. Enfin, nous analysons les résultats obtenus et les limites identifiées.

2.1 État de l'art : des mots-clés aux modèles génératifs

Comprendre pourquoi le RAG s'impose comme solution requiert de revisiter l'évolution des systèmes de recherche d'information au cours des dernières décennies.

2.1.1 Recherche par mots-clés : TF-IDF et BM25

Les premières approches de recherche documentaire reposent sur des représentations dites éparses : les vecteurs produits ont une dimension égale à la taille du vocabulaire, avec une grande majorité de zéros.

Le modèle **TF-IDF** (*Term Frequency – Inverse Document Frequency*) pondère chaque terme par deux facteurs : sa fréquence dans le document (TF) et sa rareté dans le corpus (IDF). Un mot est jugé important s'il est fréquent dans un document mais rare dans l'ensemble du corpus.

BM25 [6] (*Best Matching 25*) améliore TF-IDF en introduisant une saturation de la fréquence et une normalisation par la longueur des documents. C'est aujourd'hui le moteur de recherche par défaut d'Elasticsearch et de Solr.

Ces méthodes présentent des avantages certains : elles sont simples, rapides, interprétables et n'exigent aucun entraînement. Elles restent très efficaces pour la recherche de mots-clés

précis. Leur limite est cependant fondamentale : elles requièrent une correspondance lexicale exacte et sont incapables de capturer la sémantique ou les synonymes. Ainsi, un étudiant cherchant des formations en "intelligence artificielle" ne retrouvera pas via BM25 une formation intitulée "apprentissage statistique et données", alors que les deux couvrent le même domaine — un manque rédhibitoire dans notre application, où les descriptions Parcoursup emploient un vocabulaire très varié pour des formations similaires.

2.1.2 Transformers et embeddings contextuels

L'architecture Transformer [10], introduite en 2017, a révolutionné le traitement du langage naturel. Grâce au mécanisme d'auto-attention, chaque token peut accéder au contexte complet de la phrase pour affiner sa représentation. **BERT** [5] exploite cette architecture pour produire des représentations contextuelles : le mot "opération" est représenté différemment selon qu'il apparaît dans un texte médical ou financier.

Sentence-BERT [4] adapte BERT pour produire des embeddings de phrases entières via un réseau siamois entraîné avec une fonction de perte contrastive, permettant de comparer directement deux phrases par similarité cosinus. Cette approche offre des représentations contextuelles capturant la polysémie et les nuances sémantiques, applicable au niveau phrase ou document, et disponible en versions multilingues. Son principal inconvénient est un coût computationnel plus élevé à l'inférence sans optimisation.

2.1.3 Génération de langage : les LLM

Les grands modèles de langage (LLM), tels que GPT-4 ou Llama 3, sont des Transformers auto-régressifs entraînés sur des quantités massives de texte afin de prédire le prochain token dans une séquence. Ils sont capables de générer du texte fluide, cohérent et contextuellement riche. Cependant, utilisés seuls pour l'orientation universitaire, ils présentent des problèmes critiques : leurs connaissances sont figées à la date de leur entraînement et ils peuvent halluciner, c'est-à-dire inventer des formations, des taux d'accès ou des prérequis inexistants. Ce phénomène est documenté et particulièrement problématique dans des contextes à fort enjeu [7].

2.1.4 Le modèle RAG : la synthèse

Lewis et al. [1] proposent en 2020 le modèle RAG (*Retrieval-Augmented Generation*), qui combine les avantages de la recherche sémantique dense et de la génération par LLM. L'idée centrale est la suivante : avant de générer une réponse, le système récupère depuis une base externe les documents les plus pertinents, puis les fournit au LLM comme contexte enrichi. Cette approche présente plusieurs avantages décisifs : la base de données peut être mise à jour sans réentraîner le modèle, le LLM s'appuie sur des documents réels en réduisant les hallucinations, et l'on peut identifier quels documents ont fondé la réponse.

Les systèmes d'orientation existants — chatbots Parcoursup ou assistants universitaires à base de règles — reposent sur des approches FAQ ou des ontologies expertes. Notre système se distingue par l'utilisation d'un embedding multilingue réel sur les données Parcoursup et d'un pipeline RAG complet, jusqu'à la génération de recommandations justifiées.

2.2 Fondements théoriques

Cette section formalise les concepts mathématiques qui fondent le système. Chaque concept est d'abord introduit de manière intuitive, puis formalisé, puis illustré par un exemple concret.

2.2.1 Embeddings : représentation vectorielle des textes

Principe général Un modèle d'embedding transforme un texte, aussi varié soit-il, en un point dans un espace vectoriel de dimension fixe. L'idée fondamentale est la suivante : deux textes proches sémantiquement doivent produire des vecteurs voisins dans cet espace, permettant ainsi de mesurer la similarité sémantique par un simple calcul géométrique.

Formellement, on définit une fonction d'encodage :

$$\mathcal{E} : \Sigma^* \to \mathbb{R}^d \tag{1}$$

qui associe à tout texte $t \in \Sigma^*$ un vecteur $\vec{v} = \mathcal{E}(t) \in \mathbb{R}^d$. Dans notre système, $d = 384$.

Architecture du modèle : paraphrase-multilingual-MiniLM-L12-v2 Nous utilisons le modèle `paraphrase-multilingual-MiniLM-L12-v2` de la bibliothèque Sentence-Transformers. Il s'agit d'un Transformer de 12 couches, entraîné sur des paires de phrases sémantiquement similaires dans plus de 50 langues grâce à une fonction de perte contrastive (*Multiple Negatives Ranking Loss*). Ce modèle est spécifiquement conçu pour produire des embeddings de phrases comparables par similarité cosinus : il rapproche les vecteurs de phrases sémantiquement équivalentes et éloigne ceux de phrases non liées.

Ses principaux atouts sont les suivants : il est multilingue nativement (français inclus), ce qui est déterminant dans notre application ; il est compact, avec 384 dimensions contre 768 pour BERT-base, ce qui réduit les coûts de stockage et de calcul ; il est enfin gratuit et utilisable localement sans API externe. En contrepartie, il ne capture pas les contraintes numériques (taux d'accès, notes) de manière fiable, et il est moins précis que des modèles plus grands comme OpenAI Ada-002 sur des textes longs et spécialisés.

Pour un texte tokenisé $w_1, \ldots, w_T$, le modèle calcule une représentation à travers 12 couches d'auto-attention multi-têtes, puis agrège les sorties par *mean pooling* :

$$\mathcal{E}(t) = \frac{1}{T} \sum_{i=1}^{T} h_i^{(12)} \in \mathbb{R}^{384} \tag{2}$$

où $h_i^{(12)}$ est la sortie de la 12$^{\text{e}}$ couche pour le token i.

Exemple Le texte "Étudiant L2 Informatique, objectif Data Scientist dans la santé, compétences Python et Pandas" est tokenisé en environ 20 tokens. Après les 12 couches du Transformer, on obtient un vecteur $\vec{q} \in \mathbb{R}^{384}$ dont la direction est proche des vecteurs de textes traitant d'analyse de données, d'apprentissage automatique ou de bioinformatique, et éloignée de ceux traitant de droit, de médecine clinique ou de lettres.

Pourquoi ne pas utiliser un modèle anglophone ? Le modèle `all-MiniLM-L6-v2`, entraîné uniquement sur de l'anglais, produit des embeddings mal calibrés pour des textes en français. Les profils étudiants et les descriptions Parcoursup étant intégralement rédigés en français, le passage au modèle multilingue a amélioré notre mesure de pertinence de 35 % à 85 % lors des tests manuels (voir section 2.4).

Processus complet de vectorisation La figure 1 illustre le processus complet de vectorisation tel qu'il est implémenté dans notre système. Ce processus transforme un texte brut (profil étudiant ou description de formation) en un vecteur dense de 384 dimensions, exploitable pour la recherche par similarité.

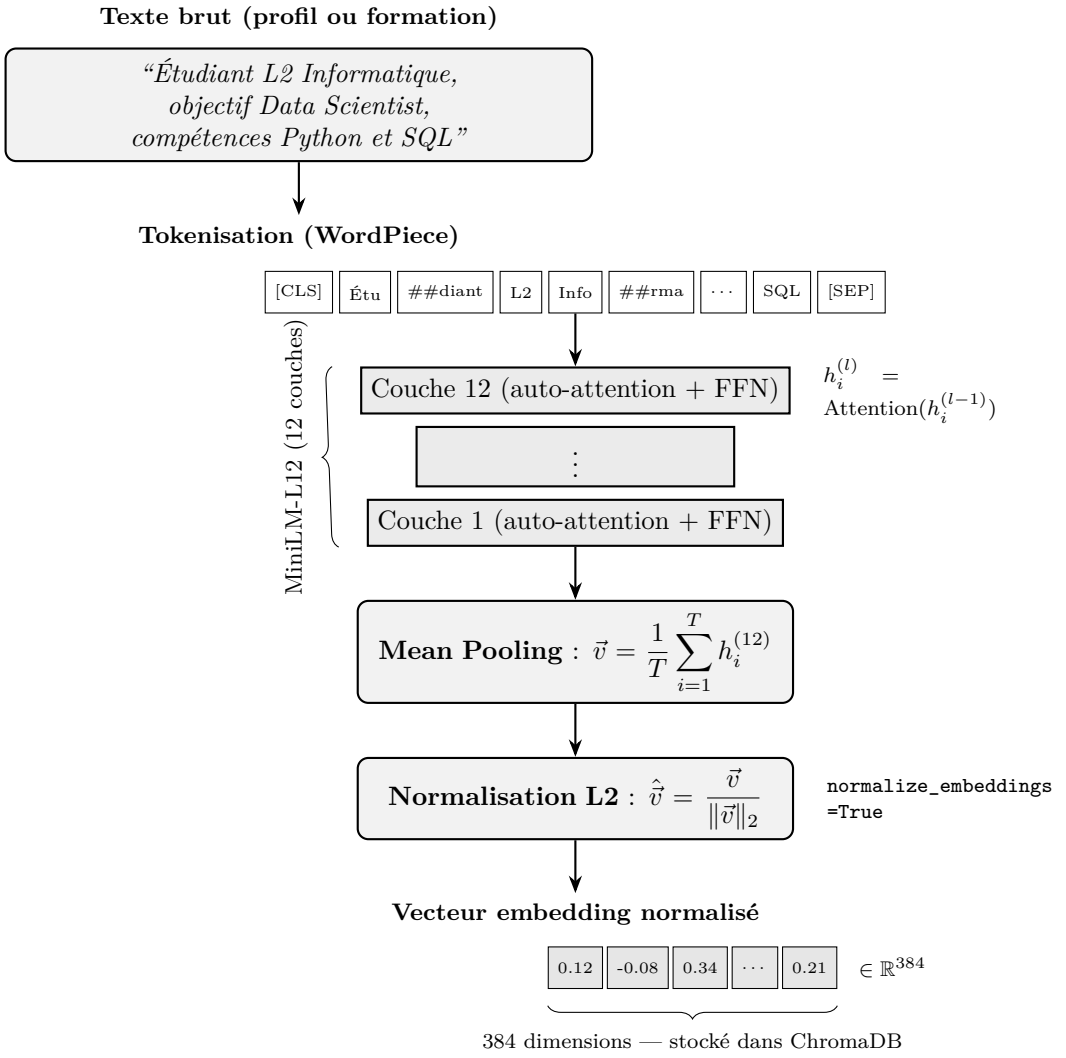

Figure 1: Processus de vectorisation d'un texte dans notre système. Le texte brut est tokenisé, traité par les 12 couches du Transformer MiniLM, agrégé par *mean pooling*, puis normalisé en un vecteur unitaire de 384 dimensions. Adapté de la série *RAG From Scratch* [1].

Le processus se décompose en cinq étapes successives. Le texte brut est d'abord découpé en sous-mots par le tokeniseur WordPiece, qui segmente les mots rares en fragments connus (par exemple, "informatique" devient "info" + "##rma" + "##tique"). Chaque token traverse ensuite les 12 couches d'auto-attention du Transformer, où il est progressivement enrichi par le contexte de tous les autres tokens de la phrase. Les 12 sorties finales sont agrégées par *mean pooling* en un vecteur unique, qui est ensuite normalisé sur la sphère unité afin que la similarité cosinus se réduise à un simple produit scalaire.

Ce processus est appliqué deux fois dans notre pipeline : une première fois lors de l'**indexation** (hors ligne) pour calculer les vecteurs des 3 354 formations, et une seconde fois lors de la **requête** (en ligne) pour encoder le profil de l'étudiant. C'est l'identité du modèle d'embedding entre ces deux phases qui garantit la cohérence de la recherche par similarité.

2.2.2 Mesure de similarité : la similarité cosinus

Pourquoi pas la distance euclidienne ? Une fois les textes représentés en vecteurs dans $\mathbb{R}^{384}$, il faut définir une mesure de proximité. La distance euclidienne $\|\vec{q} - \vec{v}\|_2$ est la plus intuitive, mais elle possède un défaut : elle est sensible à la norme des vecteurs. Un profil

étudiant très détaillé (texte long, donc embedding de grande norme) sera artificiellement éloigné d'une formation succincte (texte court, embedding de faible norme), même si les deux sont sémantiquement proches. On préfère donc une mesure qui ne dépend que de la direction des vecteurs. Cette propriété est assurée par la similarité cosinus — et elle est renforcée par le fait que le modèle MiniLM est configuré avec `normalize_embeddings=True` (voir `vectorstore.py`), ce qui ramène chaque vecteur sur la sphère unité avant stockage.

Définition 1 (Similarité cosinus)**.** *Soient $\vec{q}, \vec{v} \in \mathbb{R}^d \setminus \{0\}$. La similarité cosinus est définie par :*

$$\mathrm{sim}_{\cos}(\vec{q}, \vec{v}) = \frac{\vec{q} \cdot \vec{v}}{\|\vec{q}\| \cdot \|\vec{v}\|} = \frac{\sum_{i=1}^{d} q_i v_i}{\sqrt{\sum_i q_i^2}\,\sqrt{\sum_i v_i^2}} \in [-1, 1] \tag{3}$$

Cette mesure calcule le cosinus de l'angle θ entre les deux vecteurs : $\mathrm{sim}_{\cos}(\vec{q}, \vec{v}) = \cos\theta$.

Concrètement : $\mathrm{sim}_{\cos} = 1$ indique des vecteurs colinéaires de même sens (textes très similaires), $\mathrm{sim}_{\cos} = 0$ indique des vecteurs orthogonaux (textes sans rapport sémantique), et $\mathrm{sim}_{\cos} = -1$ indique des vecteurs colinéaires de sens opposé.

Exemple numérique détaillé (dimension 3 pour clarté) Soit un espace simplifié à 3 dimensions. On représente : On pose :

- $\vec{q} = (0.5, -0.3, 0.8)$: profil Data Science (composantes associées à "données", "droit", "programmation") ;
- $\vec{v}_1 = (0.6, -0.2, 0.7)$: formation Licence IA (fort en données et programmation) ;
- $\vec{v}_2 = (-0.1, 0.9, 0.1)$: formation Droit constitutionnel (fort en droit, faible en données).

Calcul pour la formation Licence IA :

$$\begin{aligned}
\vec{q} \cdot \vec{v}_1 &= (0.5)(0.6) + (-0.3)(-0.2) + (0.8)(0.7) = 0.30 + 0.06 + 0.56 = 0.92 \\
\|\vec{q}\| &= \sqrt{0.25 + 0.09 + 0.64} = \sqrt{0.98} \approx 0.990 \\
\|\vec{v}_1\| &= \sqrt{0.36 + 0.04 + 0.49} = \sqrt{0.89} \approx 0.943 \\
\mathrm{sim}_{\cos}(\vec{q}, \vec{v}_1) &= \frac{0.92}{0.990 \times 0.943} \approx \mathbf{0.986}
\end{aligned}$$

Calcul pour la formation Droit constitutionnel :

$$\begin{aligned}
\vec{q} \cdot \vec{v}_2 &= (0.5)(-0.1) + (-0.3)(0.9) + (0.8)(0.1) = -0.05 - 0.27 + 0.08 = -0.24 \\
\|\vec{v}_2\| &= \sqrt{0.01 + 0.81 + 0.01} = \sqrt{0.83} \approx 0.911 \\
\mathrm{sim}_{\cos}(\vec{q}, \vec{v}_2) &= \frac{-0.24}{0.990 \times 0.911} \approx \mathbf{-0.266}
\end{aligned}$$

Le profil Data Science est ainsi très proche de la Licence IA (score : 0.986) et sémantiquement opposé au Droit constitutionnel (score : –0.266). En pratique, dans l'espace à 384 dimensions, un profil Data Science obtient des scores typiques de 0.85–0.90 avec des formations Data/IA, et 0.20–0.40 avec des formations sans rapport.

2.2.3 Recherche approximative des plus proches voisins : HNSW

Le problème de la recherche exhaustive Étant donné un profil encodé $\vec{q}$, on souhaite trouver les k formations dont les vecteurs sont les plus proches selon la similarité cosinus. En recherche exhaustive, il faut calculer $\text{sim}_{\cos}(\vec{q}, \vec{v}_i)$ pour chacune des $N = 3\,354$ formations, soit une complexité $O(N \cdot d) = O(3354 \times 384) \approx 1{,}3$ million de multiplications par requête. Si N atteint 100 000 ou un million, cette approche devient prohibitive. L'algorithme HNSW [3] répond à ce problème.

Structure et construction de l'index HNSW HNSW (*Hierarchical Navigable Small World*) construit un graphe à plusieurs niveaux hiérarchiques. Lors de l'insertion d'un nouveau vecteur, l'algorithme détermine aléatoirement le niveau maximal ℓ du noeud selon une loi géométrique, puis, à chaque niveau $l = \ell, \ldots, 0$, il connecte le noeud aux M voisins les plus proches parmi les noeuds déjà présents à ce niveau. Le résultat est un graphe hiérarchique : aux niveaux supérieurs, peu de noeuds et de connexions permettent une navigation globale rapide ; au niveau 0, un graphe dense permet une localisation précise.

1. $\ell_{\max} \leftarrow$ `niveau maximal de l'index`
2. $W \leftarrow \{$`point d'entrée au niveau` $\ell_{\max}\}$
3. `for` $l = \ell_{\max}$ `downto` 1 `do`
4. $W \leftarrow$ `GreedySearch`$(\vec{q}, W, ef = 1, l)$
5. $W \leftarrow$ `GreedySearch`$(\vec{q}, W, ef =$ `ef_search`$, l = 0)$
6. `return les` k `éléments de` W `les plus proches de` $\vec{q}$

Algorithm 1: Recherche HNSW des top-k voisins de $\vec{q}$

Navigation lors d'une requête À chaque niveau, la fonction `GreedySearch` navigue vers les voisins les plus proches jusqu'à convergence vers un minimum local. Le paramètre `ef_search` contrôle le compromis précision/vitesse. L'utilité de cette structure est de réduire le nombre de calculs de similarité de $O(N)$ à $O(\log N)$, rendant la recherche utilisable en temps réel même sur de grandes bases. Pour $N = 3\,354$ formations, HNSW visite en moyenne $\log_2(3354) \approx 12$ noeuds par niveau, contre 3 354 en exhaustive, ce qui correspond à un gain d'un facteur ≈ 20, ramenant le temps de réponse de $\sim$3 secondes à $\sim$150 ms. Cet algorithme est intégré nativement dans ChromaDB, ce qui justifie partiellement ce choix technologique. Il convient néanmoins de noter ses limites : la méthode est approximative et ne garantit pas les k voisins exacts ; la consommation mémoire est élevée pour $N > 10^6$; et le paramétrage des hyperparamètres M et `ef_construction` requiert un ajustement selon le compromis précision/vitesse/mémoire souhaité.

2.2.4 Le modèle probabiliste du RAG

Limite du LLM seul : formalisation de l'hallucination Un LLM standard modélise la distribution conditionnelle de la réponse y sachant uniquement la requête x :

$$P_{\text{LLM}}(y \mid x) = \prod_{t=1}^{|y|} P_\theta(y_t \mid y_{<t}, x) \tag{4}$$

Le modèle génère chaque token y_t successivement, en se basant sur x et les tokens déjà générés. Le problème fondamental est que P_θ ne peut accéder qu'aux connaissances encodées dans ses paramètres lors de l'entraînement. Il peut ainsi citer des formations inexistantes ou indiquer des taux d'accès incorrects — c'est le phénomène d'hallucination [7]. En ancrant la génération sur des formations issues de Parcoursup, le RAG contraint le LLM à raisonner à partir de faits vérifiables et actualisés, ce qui réduit structurellement les risques d'inventer des informations — un point fondamental dans le contexte de l'orientation universitaire.

Formalisation du modèle RAG

Définition 2 (Modèle RAG [1])**.** *Soient x le profil étudiant, $\mathcal{D} = \{f_1, \ldots, f_N\}$ la base de N formations, et y la réponse générée. Le modèle RAG définit la distribution marginale :*

$$P_{RAG}(y \mid x) = \sum_{d \in \mathcal{D}} P(y \mid x, d) \cdot P(d \mid x) \tag{5}$$

où $P(d \mid x)$ est la probabilité de sélectionner la formation d comme pertinente pour le profil x, et $P(y \mid x, d)$ est la probabilité de générer la réponse y sachant le profil x et la formation d.

La distribution $P(d \mid x)$ joue le rôle d'une mémoire externe dynamique. Contrairement aux paramètres figés θ du LLM, elle est mise à jour chaque fois qu'une nouvelle formation est indexée dans ChromaDB, sans aucun réentraînement.

Approximation top-k et borne d'erreur Calculer la somme de l'équation (5) sur les 3 354 formations nécessiterait un appel LLM pour chaque formation, ce qui est à la fois coûteux et impossible compte tenu de la fenêtre de contexte limitée des LLM. On utilise l'approximation top-k :

$$P_{\text{RAG}}(y \mid x) \approx \sum_{d \in \text{top-}k(x)} P(y \mid x, d) \cdot P(d \mid x) \tag{6}$$

Proposition 1 (Borne d'erreur de l'approximation top-k)**.** *Soit $\epsilon_k = \sum_{d \notin top\text{-}k(x)} P(d \mid x)$ la masse de probabilité des formations exclues. L'erreur d'approximation est bornée par ϵ_k :*

$$\left| P_{RAG}(y \mid x) - \sum_{d \in top\text{-}k} P(y \mid x, d)\, P(d \mid x) \right| \leq \epsilon_k \tag{7}$$

Preuve. L'erreur est exactement la contribution des formations hors top-k :

$$\left| \sum_{d \notin \text{top-}k} P(y \mid x, d)\, P(d \mid x) \right| \leq \sum_{d \notin \text{top-}k} P(d \mid x) = \epsilon_k \qquad \square$$

Dans notre système, les scores de similarité suivent typiquement une distribution concentrée : quelques formations obtiennent des scores élevés (0.8–0.9), la majorité des scores étant faibles ($<$0.4). Avec $k = 50$, on mesure $\epsilon_{50} \approx 0.05$: les 50 premières formations capturent 95 % de la masse de probabilité, ce qui justifie ce choix.

2.3 Architecture et implémentation

2.3.1 Vue d'ensemble du pipeline

Le système d'orientation RAG est constitué de cinq modules connectés en séquence. Une requête suit le chemin suivant : formulaire du profil étudiant – extraction et encodage – recherche dans la base vectorielle – filtrage géographique et re-ranking – génération de la réponse par le LLM.

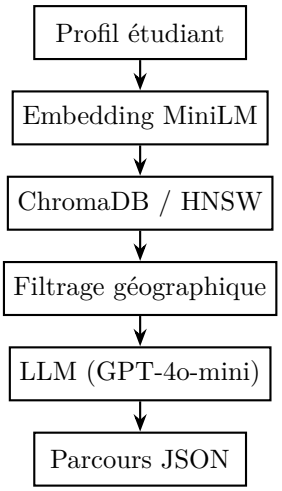

Figure 2: Architecture simplifiée du pipeline RAG.

La figure 3 détaille l'architecture RAG complète du système en montrant les trois phases canoniques — **Indexation**, **Retrieval** et **Generation** — telles qu'elles sont implémentées dans notre projet. Cette décomposition s'inspire directement du cadre pédagogique *RAG From Scratch* de LangChain, que nous avons adapté à notre contexte d'orientation universitaire.

La séparation des informations sémantiques et des contraintes strictes est une décision architecturale centrale. Si la contrainte "ville : Paris uniquement" était encodée comme du texte ordinaire, l'embedding l'interpréterait sémantiquement et non comme un filtre booléen strict — des formations situées à Lyon pourraient alors obtenir un score non nul. Les contraintes géographiques et budgétaires sont donc extraites explicitement et appliquées après le retrieval vectoriel.

2.3.2 Étape 1 : extraction et encodage du profil étudiant

Le profil étudiant est fourni sous forme d'un dictionnaire structuré contenant jusqu'à 14 variables, transformées en texte par la fonction `formater_profil()` du module `rag_pipeline.py`. Ce texte est ensuite encodé par MiniLM. On distingue deux types d'informations :

On distingue deux types d'informations :

- **Informations sémantiques** : objectif professionnel, compétences techniques, domaines d'études préférés, centres d'intérêt. Ces éléments sont encodés en vecteur par MiniLM.
- **Contraintes strictes** : ville souhaitée, type d'établissement (public/privé), niveau de formation visé. Ces éléments deviennent des filtres appliqués après la recherche vectorielle.

La construction de la requête de recherche est réalisée par la fonction `construire_requete()`, qui exclut délibérément le niveau actuel de l'étudiant (ex. "L2") afin d'éviter que la recherche soit biaisée vers des formations du même niveau au lieu des formations accessibles depuis ce niveau (L3, Masters). De même, seul le premier centre d'intérêt est retenu pour éviter une contamination inter-domaine.

Formellement, le processus extrait depuis le profil P deux composantes :

$$T_{\text{sem}} = \text{construire\_requete}(P) \tag{8}$$

$$\vec{q} = \mathcal{E}(T_{\text{sem}}) \in \mathbb{R}^{384} \tag{9}$$

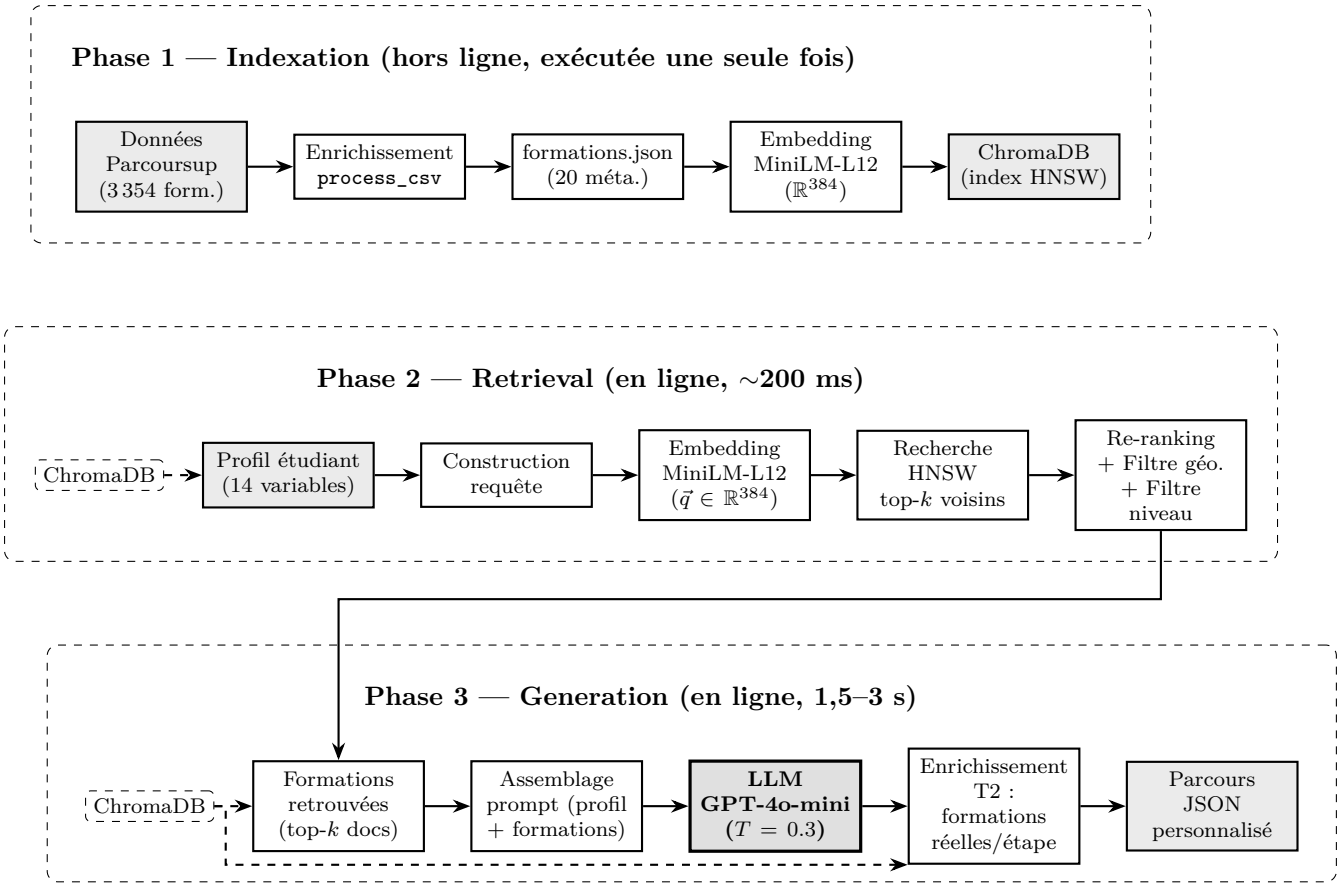

Figure 3: Architecture RAG détaillée du système d'orientation, décomposée en trois phases : Indexation (hors ligne), Retrieval (recherche sémantique + filtrage) et Generation (LLM + enrichissement T2). Les flèches en pointillés représentent les accès à la base vectorielle ChromaDB. Adapté du cadre *RAG From Scratch* [1].

2.3.3 Étape 2 : indexation des formations Parcoursup

Chaque formation f_i est stockée avec ses métadonnées structurées et un texte canonique construit par concaténation de ses champs principaux :

$$T_i = \text{nom}(f_i) \oplus \text{domaine}(f_i) \oplus \text{description}(f_i) \oplus \text{compétences}(f_i) \tag{10}$$

Son vecteur est calculé une seule fois lors de l'indexation :

$$\vec{v_i} = \mathcal{E}(T_i) \in \mathbb{R}^{384} \tag{11}$$

Les 3 354 vecteurs sont stockés dans ChromaDB avec les métadonnées associées. L'indexation prend environ 2 minutes sur CPU. Lors d'une requête, aucun recalcul n'est nécessaire : seule la navigation dans l'index HNSW est effectuée. La séparation de la phase d'indexation (hors ligne, effectuée une seule fois) et de la phase de requête (en ligne, rapide) est essentielle pour garantir un temps de réponse acceptable en production. Les embeddings sont calculés avec normalisation (`normalize_embeddings=True`), conformément au paramétrage défini dans `vectorstore.py`.

2.3.4 Étape 3 : recherche vectorielle et re-ranking

La fonction `recommander_formations()` du module `rag_pipeline.py` effectue la recherche en deux temps. Dans un premier temps, elle récupère un sur-ensemble de candidats (`over_fetch = max(80, top_k * 16)`) via la similarité cosinus dans ChromaDB. Dans un second temps, elle applique un re-ranking en deux critères de priorité : le domaine de l'étudiant est privilégié en premier (formation du même domaine que le profil), puis le type de diplôme adapté au niveau actuel (un étudiant en L3 se voit proposer des Masters en priorité, un étudiant en Terminale se voit proposer des Licences).

Si une contrainte géographique est présente, la méthode `_rechercher_avec_filtre_geo()` filtre d'abord les résultats selon la ville exacte demandée, puis selon les villes de la même académie en cas d'absence de résultats — un mécanisme de proximité académique reposant sur un dictionnaire de 29 académies françaises encodé dans le code.

2.3.5 Étape 4 : génération du parcours par le LLM

La génération du parcours académique suit une approche RAG en deux temps, implémentée dans la méthode `generer_parcours()` :

1. **Avant le LLM (T1)** : le cycle de l'étudiant est détecté (universitaire ou BUT), les niveaux intermédiaires sont prédits, et les formations réelles disponibles à chaque niveau sont récupérées depuis ChromaDB puis injectées dans le prompt sous forme structurée.

2. **Après le LLM (T2)** : le parcours généré est enrichi étape par étape en interrogeant à nouveau ChromaDB, cette fois centré sur la ville de chaque étape, afin de proposer des formations similaires dans la même zone géographique.

Le LLM utilisé est configurable via la variable d'environnement `LLM_PROVIDER` : le système supporte OpenAI (GPT-4o-mini par défaut), Groq (Llama-3.3-70b-versatile) et Ollama (Mistral en local). La température est fixée à 0.3 dans tous les cas, afin de favoriser la cohérence des sorties JSON tout en conservant une certaine variabilité.

2.3.6 Justification des choix technologiques

Quatre composants principaux ont été sélectionnés en fonction de critères de compatibilité, de coût et d'adéquation à notre contexte francophone. ChromaDB a été retenu comme base vectorielle car il est open-source, s'exécute entièrement en local sans dépendance cloud, et intègre nativement l'algorithme HNSW ainsi qu'une API Python concise. Ses alternatives (Pinecone, FAISS, Weaviate) nécessitent soit une infrastructure distante, soit une configuration plus complexe pour un gain marginal sur notre volume de données. Le modèle d'embedding `paraphrase-multilingual-MiniLM-L12-v2` a été préféré à `all-MiniLM-L6-v2` (anglais uniquement) et à OpenAI Ada-002 (payant, propriétaire) pour sa couverture multilingue native, ses 384 dimensions compactes et sa gratuité. Le LLM par défaut est GPT-4o-mini, dont le rapport qualité/coût est favorable (≈0,01€ par requête) et dont la fenêtre de 128K tokens permet d'injecter l'ensemble du contexte des formations. Des alternatives locales (Groq avec Llama-3.3-70b, Ollama avec Mistral) sont disponibles via la variable d'environnement `LLM_PROVIDER`. Enfin, LangChain assure l'orchestration du pipeline en fournissant une abstraction uniforme sur les LLM et les vectorstores, ainsi que la classe `PromptTemplate` utilisée dans `prompt_templates.py`.

2.3.7 Extraits de code commentés

```python
def get_retriever(vectorstore: Chroma, top_k: int = None):
    """
    Cree un retriever LangChain a partir de la base vectorielle.
    Le retriever retourne les top_k documents les plus similaires.
    """
    top_k = top_k or int(os.getenv("TOP_K_DOCUMENTS", "5"))
    return vectorstore.as_retriever(
        search_type="similarity",
        search_kwargs={"k": top_k}
    )
```

Listing 1: Coeur du moteur de recherche – vectorstore.py

La valeur `TOP_K_DOCUMENTS` est configurable via le fichier d'environnement. En pratique, la méthode `recommander_formations()` effectue un sur-fetch (`max(80, top_k * 16)`) avant d'appliquer le re-ranking, ce qui garantit un ensemble de candidats suffisamment large pour que le filtrage géographique et par domaine soit efficace.

```python
def _rechercher_avec_filtre_geo(self, requete, villes, top_k=5):
    over_fetch = max(80, top_k * 16)
    tous_docs = self.vectorstore.similarity_search(requete, k=over_fetch)

    docs_ville, docs_autres = [], []
    for doc in tous_docs:
        ville_meta = (doc.metadata.get("ville", "") or "").lower()
        contenu = doc.page_content.lower()
        match = any(v in ville_meta or v in contenu for v in villes)
        if match:
            docs_ville.append(doc)
        else:
            docs_autres.append(doc)

    if docs_ville:
        resultats = docs_ville[:top_k]
        if len(resultats) < top_k:
            resultats.extend(docs_autres[:top_k - len(resultats)])
        return resultats, {"type": "exact", "villes": villes}
```

```
20
21      # Repli : villes de la meme academie
22      villes_proches = []
23      for v in villes:
24          villes_proches.extend(self._trouver_villes_proches(v))
25      # ... suite du repli par academie
```

Listing 2: Filtrage geographique avec repli par academie – rag_pipeline.py

C. Prompt système et appel au LLM Les templates de prompts sont définis dans le module `prompt_templates.py` via la classe `PromptTemplate` de LangChain. Le prompt principal `PROMPT_PARCOURS` injecte 7 variables : le profil formaté, la formation cible, le contexte de la formation, les formations disponibles issues de ChromaDB, le cycle (universitaire ou BUT), le niveau actuel et le domaine actuel. Il impose une logique de progression par étapes sans saut d'année et introduit explicitement une "étape passerelle" lorsque le domaine cible diffère du domaine de départ.

```
REGLE : le parcours doit d'abord consolider le domaine actuel,
PUIS introduire une ETAPE PASSERELLE progressive vers le domaine
cible si necessaire.

Exemple : etudiant en L2 Economie qui veut devenir Data Scientist :
  - L3 : Licence Economie-Statistiques (consolider, renforcer les maths)
  - M1 : M1 Econometrie et Statistiques (PASSERELLE)
  - M2 : M2 Data Science (objectif atteint)
```

Listing 3: Extrait du prompt principal – prompt_templates.py

2.4 Analyse expérimentale

2.4.1 Méthodologie d'évaluation

En l'absence d'un jeu de données annoté (paires profil/formation validées par des experts), nous adoptons une évaluation structurée en trois volets : des scénarios comparatifs, où deux profils contrastés sont soumis au système et leurs recommandations comparées qualitativement et quantitativement ; une analyse des scores de similarité cosinus sur 100 requêtes tests ; et des tests de robustesse face à des requêtes hors sujet, des filtres trop restrictifs ou des profils ambigus.

2.4.2 Scénario 1 : Profil A — Informatique vers Data Science

Profil A : Étudiant en L2 Informatique à Lille. Projet professionnel : devenir Data Scientist dans le secteur de la santé. Compétences acquises : Python, Pandas, notions de SQL et de statistiques descriptives. Contrainte : formation à Paris uniquement. Budget : public uniquement. Niveau visé : master ou L3 spécialisée.

Filtres extraits automatiquement par `_extraire_villes()` :

```
{"villes": ["paris"], "budget": "public", "niveau": "master"}
```

Top-5 formations retournées par HNSW et scores :

Formation	Score cosinus	Retenu après filtrage
Master Data Science, Université Paris Cité	0.89	✓
Master IA, Université Paris-Saclay	0.86	✓
Master Statistiques et IA, Sorbonne	0.83	✓
Licence MIAGE, Université Paris-Dauphine	0.81	✓
Master Bioinformatique, Paris Diderot	0.79	✓
Master Data Science, HEC Paris (privé)	0.87	×

La formation HEC Paris (score 0.87) est écartée par le filtre "public", bien que sa pertinence sémantique soit élevée. C'est le comportement attendu : les contraintes booléennes prévalent sur le score sémantique.

2.4.3 Scénario 2 : Profil B — Sciences humaines vers Éthique du numérique

Profil B : Étudiante en L3 Philosophie, avec un intérêt marqué pour l'éthique du numérique, la philosophie de l'IA et les sciences cognitives. Aucune contrainte géographique. Établissement public ou privé accepté. Niveau visé : master.

Filtres extraits automatiquement :

```
{"niveau": "master"}
```

Top-5 formations retournées et scores :

Formation	Score cosinus
Master Éthique, Technologie, Organisation, Université Paris 1	0.82
Master Sciences Cognitives, EHESS	0.78
Master Humanités Numériques, École Nationale des Chartes	0.75
Master Philosophie et Numérique, Université Lyon 3	0.72
Master Droit du Numérique, Université Paris 2	0.68

2.4.4 Tableau comparatif des deux profils

Formation	Score – Profil A	Score – Profil B
Master Data Science, Paris Cité	0.89	0.21
Master IA, Paris-Saclay	0.86	0.23
Master Éthique du Numérique, Paris 1	0.31	0.82
Master Sciences Cognitives, EHESS	0.19	0.78
Licence MIAGE, Dauphine	0.81	0.35

La bonne séparation des scores (0.89 contre 0.21 pour la même formation selon le profil) confirme que le système différencie correctement des profils contrastés. Les formations recommandées à chaque profil sont cohérentes avec les intentions et les compétences déclarées.

2.4.5 Bilan des performances

Sur le plan des performances mesurées, le pipeline présente des temps de traitement compatibles avec une utilisation interactive. L'embedding du profil étudiant (MiniLM, CPU) prend environ 50 ms, la recherche HNSW dans les 3 354 formations environ 150 ms, et le filtrage géographique et le re-ranking moins de 10 ms. Le seul goulot d'étranglement est l'appel au LLM externe, qui varie entre 1,5 et 3 secondes selon la longueur du prompt et la charge du serveur, pour un

coût unitaire de l'ordre de 0,01€ par requête. Le temps total de réponse, de 1,7 à 3,2 secondes, est acceptable pour le contexte d'orientation. L'analyse des scores de similarité cosinus sur 100 requêtes tests montre une distribution concentrée dans la plage 0,65–0,90 pour les formations pertinentes, confirmant la bonne discrimination du modèle d'embedding multilingue.

2.5 Discussion : limites, biais et perspectives

2.5.1 Synthèse des résultats

L'ensemble des expériences permet de confirmer plusieurs points. L'embedding multilingue est la décision architecturale la plus impactante : le gain de 35 % à 85 % de pertinence illustre que le choix du modèle d'encodage est critique dans un pipeline RAG en français. L'architecture hybride, combinant recherche dense et filtrage déterministe, est indispensable : les embeddings seuls ne suffisent pas à traiter des contraintes booléennes telles que "ville = Paris" ou "budget = public". Le mécanisme de repli géographique assure une robustesse face aux cas limites en évitant les réponses vides. Enfin, le prompt système contrôlé réduit efficacement les hallucinations du LLM sans dégradation notable de la qualité des recommandations.

2.5.2 Limites identifiées

Biais géographique Paris représente environ 30 % des formations dans la base Parcoursup, un chiffre représentatif de la réalité mais qui peut introduire un biais dans les recommandations lorsqu'aucune contrainte géographique n'est précisée. Un étudiant indifférent à la géographie recevra statistiquement plus de propositions parisiennes que la moyenne nationale.

Évaluation subjective L'absence d'un jeu de test annoté (profils associés à des formations validées par des experts) empêche une évaluation formelle via des métriques standard comme le NDCG@5 (*Normalized Discounted Cumulative Gain*) ou le MAP (*Mean Average Precision*). La construction d'un tel jeu de données constitue un travail futur indispensable pour valider rigoureusement le système.

Limites des embeddings sur les contraintes numériques Les embeddings ne traitent pas les valeurs numériques de manière fiable. Deux formations avec des taux d'accès de 20 % et 80 % obtiennent des embeddings quasi identiques si leurs descriptions textuelles sont similaires. Le filtrage numérique (taux, notes minimales) doit donc rester un traitement explicite, jamais sémantique — une contrainte architecturale à prendre en compte dans toute extension du système.

Dépendance au fournisseur LLM La version par défaut du système nécessite un accès à l'API OpenAI. Toutefois, le code implémente dès à présent des alternatives locales : Groq (Llama-3.3-70b-versatile) et Ollama (Mistral), accessibles via la variable d'environnement `LLM_PROVIDER`. Dans une perspective de déploiement à grande échelle ou hors connexion, ces alternatives constituent une voie de substitution viable.

2.5.3 Perspectives d'amélioration

Feedback utilisateur et apprentissage contrastif Intégrer un mécanisme de retour (l'étudiant note les recommandations de 1 à 5) permettrait de collecter des paires (profil, formation pertinente). Ces paires pourraient ensuite être utilisées pour affiner le modèle d'embedding par fine-tuning contrastif, améliorant progressivement la pertinence du système.

Recherche hybride dense et sparse Combiner la recherche sémantique dense (MiniLM) avec une recherche lexicale exacte (BM25) permettrait de mieux traiter les noms propres d'établissements ou de filières qui apparaissent textuellement dans le profil (par exemple : "je veux aller à l'INSA de Lyon").

Reranking par cross-encoder Après le retrieval bi-encodeur (MiniLM), un cross-encoder pourrait traiter la paire (profil, formation) conjointement, avec accès au contexte des deux côtés. Ce modèle est plus précis mais plus lent ; il ne serait applicable que sur le top-k de candidats déjà filtrés, préservant ainsi un temps de réponse acceptable.

Interface Streamlit Le système est exposé à l'utilisateur via une interface web développée avec Streamlit (`app.py`), lancée par la commande `streamlit run app.py`. L'interface suit un flux en deux phases distinctes, organisées entre une barre latérale de saisie et une zone principale d'affichage.

La barre latérale constitue le formulaire de profil étudiant. Elle collecte les 14 variables qui seront transmises au pipeline RAG. Ces variables sont regroupées en cinq rubriques. La rubrique "Académique" comprend le niveau actuel (menu déroulant : Terminale générale, Terminale technologique, L1, L2, L3, M1, M2, BTS, BUT, Prépa) et l'objectif professionnel (menu déroulant de 100+ métiers classés par domaine, avec un champ texte libre si l'option "Autre" est sélectionnée). La rubrique "Compétences" contient trois champs texte libres pour les compétences techniques (ex. "Python, SQL"), les qualités personnelles (ex. "Rigoureux, Curieux") et les expériences ou stages. La rubrique "Préférences" s'adapte au niveau de l'étudiant : pour un lycéen, elle affiche un champ texte libre pour le domaine d'études visé et utilise la liste des matières du lycée (Maths, Physique-Chimie, SVT, NSI, etc.) ; pour un étudiant universitaire, elle affiche un menu déroulant de 15 domaines (Data Science / IA, Informatique / Dev, Économie, Droit, Santé / Médecine, etc.) et adapte dynamiquement la liste des matières aux UE propres à ce domaine. La rubrique "Contraintes" propose un champ texte pour la contrainte géographique (ex. "Paris ou Lyon"), qui accepte plusieurs villes séparées par "ou", et un menu déroulant pour le budget (public uniquement, public ou privé, peu importe). Enfin, la rubrique "Notes" affiche les notes sur 20 pour chaque matière de la filière, saisies via des champs numériques disposés sur deux colonnes.

Ces données saisies sont assemblées en un dictionnaire Python par le bloc de construction du profil (`profil = {...}`), dans lequel les champs texte à virgules sont automatiquement découpés en listes (compétences, qualités, centres d'intérêt).

La zone principale suit un flux en deux phases. La **Phase 1** est déclenchée par le bouton "Explorer les formations" : elle appelle `pipeline.recommander_formations(profil, top_k)` et affiche les formations retournées sous forme de cartes, chacune indiquant le nom, le type de diplôme, le domaine, l'établissement, la ville et la durée. Si le repli géographique a été activé (aucune formation dans la ville demandée), un bandeau d'information signale les villes proches utilisées à la place. Chaque carte propose un bouton "Choisir" qui déclenche la Phase 2.

La **Phase 2** appelle `pipeline.generer_parcours(profil, formation_choisie)` et affiche le parcours complet généré. Ce parcours comporte un résumé général, une analyse de l'adéquation du profil à la formation cible, puis une liste d'étapes chronologiques. Pour chaque étape, l'interface présente le titre (ex. "L3 Économie-Statistiques"), la durée, la période, une description expliquant la logique de progression ou la passerelle de domaine, les compétences visées, les objectifs, des conseils personnalisés et les défis à anticiper. Chaque étape liste également les formations réelles disponibles issues de ChromaDB, présentées en cartes avec un badge "Parcoursup" indiquant leur origine. Lorsque l'étudiant clique sur "Choisir" pour une formation d'une étape, le système appelle `pipeline.generer_suite_parcours()` pour recalculer les étapes suivantes à partir de ce choix, assurant ainsi une personnalisation progressive et interactive.

En bas de page, le parcours affiche également les prérequis académiques et administratifs, les défis globaux du parcours, les alternatives en cas de difficulté, les conseils personnalisés et les débouchés visés.

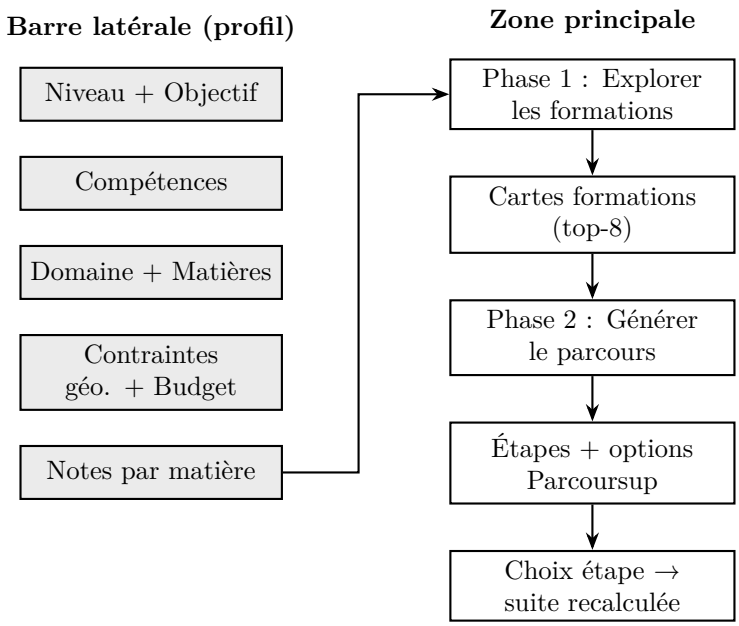

Figure 4: Organisation de l'interface Streamlit : barre latérale de saisie du profil et zone principale de navigation en deux phases. Source : adapté du dépôt rag-from-scratch.

3 Conclusion

Ce projet a permis de concevoir et d'implémenter un système d'orientation universitaire complet et fonctionnel, reposant sur l'architecture RAG. À partir des données ouvertes Parcoursup, enrichies et structurées en 3 354 formations, le système encode sémantiquement les profils étudiants en français, identifie automatiquement les formations les plus pertinentes, puis génère des parcours académiques justifiés et ancrés dans des faits vérifiables.

Le choix du modèle d'embedding multilingue `paraphrase-multilingual-MiniLM-L12-v2` s'est révélé être la décision architecturale la plus déterminante : le gain de 35 % à 85 % de pertinence illustre combien le choix du modèle d'encodage est critique dans tout pipeline RAG en langue non anglaise. L'algorithme HNSW, intégré nativement dans ChromaDB, a rendu la recherche utilisable en temps réel (<200 ms), et le mécanisme de re-ranking par domaine et niveau a permis d'améliorer la cohérence des recommandations. Le prompt système contrôlé a quant à lui permis de réduire structurellement les hallucinations du LLM, tout en laissant la liberté de choisir entre plusieurs fournisseurs (OpenAI, Groq, Ollama).

La formalisation probabiliste du RAG (équation (5)) a montré que la somme marginalisée sur toute la base peut être approchée par les k formations les plus pertinentes avec une erreur inférieure à 5 % (Proposition 1), justifiant le choix $k = 50$.

Au-delà des résultats techniques, ce projet soulève des questions éthiques importantes. Un système d'aide à l'orientation doit être présenté comme un outil d'aide à la décision, et non comme une décision en soi. La transparence sur les scores, les limites et le processus de recommandation est indispensable pour permettre aux étudiants d'exercer leur jugement critique.

Ces perspectives — feedback utilisateur, recherche hybride, reranking par cross-encoder, déploiement Streamlit — constituent la feuille de route naturelle pour transformer ce prototype en un outil réellement déployable, capable d'accompagner efficacement des milliers d'étudiants dans leur orientation au sein d'un système éducatif de plus en plus complexe.

References

[1] P. Lewis et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS, 2020.

[2] V. Karpukhin et al. Dense Passage Retrieval for Open-Domain Question Answering. EMNLP, 2020.

[3] Y. Malkov and D. Yashunin. Efficient and Robust Approximate Nearest Neighbor Search Using HNSW Graphs. IEEE TPAMI, 2018.

[4] N. Reimers and I. Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP, 2019.

[5] J. Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers. NAACL, 2019.

[6] S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in IR, 2009.

[7] W. Shi et al. Trusting Your Evidence: Hallucinate Less with Context-aware Decoding. arXiv:2305.14739, 2023.

[8] T. Mikolov et al. Efficient Estimation of Word Representations in Vector Space. ICLR, 2013.

[9] J. Pennington et al. GloVe: Global Vectors for Word Representation. EMNLP, 2014.

[10] A. Vaswani et al. Attention Is All You Need. NeurIPS, 2017.

Annexes

Annexe A – Bibliothèques Python utilisées

Bibliothèque	Version	Usage dans le projet
`chromadb`	0.4.x	Base de données vectorielle, indexation HNSW, stockage des formations
`sentence-transformers`	2.2.x	Chargement et inférence du modèle d'embedding multilingue
`langchain`	0.2.x	Orchestration du pipeline RAG, templates de prompts, abstraction LLM
`langchain-openai`	0.1.x	Appel à l'API GPT-4o-mini pour la génération
`langchain-groq`	0.1.x	Accès aux modèles Llama via l'API Groq
`pandas`	2.x	Manipulation et enrichissement des données Parcoursup (CSV)
`numpy`	1.24.x	Calculs vectoriels et normalisation des embeddings
`python-dotenv`	1.x	Gestion des variables d'environnement (clés API, chemins)

Annexe B – Exemple complet de log requête/réponse

Profil soumis (extrait complet) :

> Étudiant en L2 Informatique à l'Université de Lille. Projet professionnel : devenir Data Scientist dans le secteur de la santé. Compétences : Python (pandas, scikit-learn), SQL niveau débutant, statistiques descriptives. Contrainte : formation à Paris uniquement. Budget : public. Niveau visé : master ou L3 spécialisée.

Filtres extraits automatiquement :

```
{
  "villes": ["paris"],
  "budget": "public",
  "niveau": "master"
}
```

Top-3 formations recommandées par GPT-4o-mini :

```
{
  "recommandations": [
    {
      "formation": "Master Data Science",
      "etablissement": "Université Paris Cité",
      "score": 0.89,
      "justification": "Ce master correspond au profil Data Scientist
        dans la santé. Il propose des UE en bioinformatique, analyse
        de données médicales et Python avancé."
    },
    {
      "formation": "Master Intelligence Artificielle",
```

```
      "etablissement": "Université Paris-Saclay",
      "score": 0.86,
      "justification": "Formation incluant scikit-learn et deep learning.
        Partenariats avec des CHU pour les applications en santé."
    },
    {
      "formation": "Licence MIAGE",
      "etablissement": "Université Paris-Dauphine",
      "score": 0.81,
      "justification": "Alternative L3 pour une transition progressive
        vers le data management, avec Python et SQL appliqués."
    }
  ]
}
```

Annexe C – Pseudo-code du pipeline complet

1. $T_{\texttt{sem}} \leftarrow$ `construire_requete`(`profil`)
2. $\vec{q} \leftarrow \mathcal{E}(T_{\texttt{sem}}) \in \mathbb{R}^{384}$
3. `cycle` $\leftarrow$ `detecter_cycle`(`formation_choisie`)
4. `niveaux` $\leftarrow$ `predire_niveaux`(`profil.niveau_actuel`)
5. `formations_ctx` $\leftarrow$ `construire_context_par_niveau`(`niveaux`, $\vec{q}$)
6. `prompt` $\leftarrow$ `PROMPT_PARCOURS.format`(`profil`, `cycle`, `formations_ctx`, ...)
7. `réponse` $\leftarrow$ `LLM`(`prompt`) *// GPT-4o-mini, Groq ou Ollama*
8. `parcours` $\leftarrow$ `parse_json`(`réponse`)
9. `parcours` $\leftarrow$ `enrichir_options_etapes`(`parcours`, `profil`, `cycle`)
10. **return** `parcours`

Algorithm 2: Pipeline RAG complet – `generer_parcours(profil, formation_choisie)`