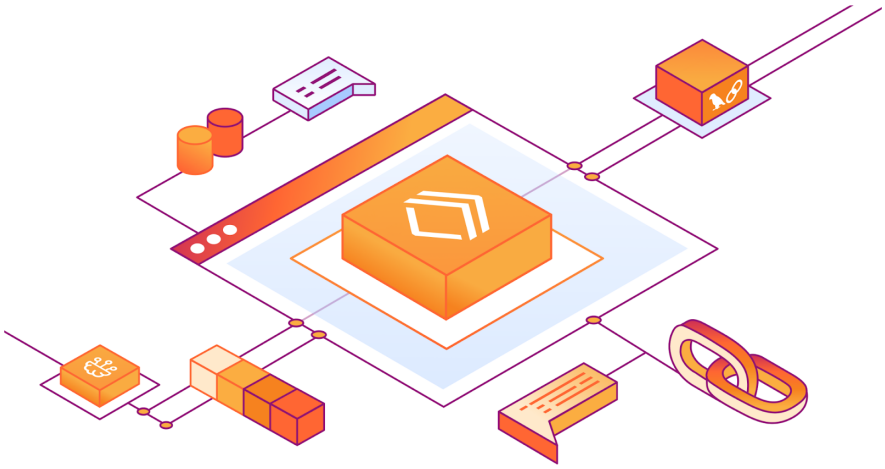
SafeTrain IA : Explorez vos documents en toute simplicité

Implémentation d'un chatbot avec un système de RAG pour une solution web interactive.



Auteurs : Bilal El Biyadi, Yvan Louamba
Tuteur : Youcef SKLAB
Spécialité : Ing 3 IA

15 Janvier 2025

Sommaire

1 Introduction

1.1 Contexte du projet

L'émergence des modèles de langage de grande taille, souvent désignés sous l'acronyme LLM (Large Language Model), a ouvert la voie à de multiples innovations dans le domaine du traitement automatique du langage. Les systèmes de génération de texte tels que GPT ou Gemini ont démontré leur capacité à produire des réponses de qualité pour une variété de tâches allant de la conversation de base à la rédaction d'articles techniques. Toutefois, lorsque l'on souhaite interroger précisément un ensemble de documents, un LLM standard peut se révéler insuffisant. Il est en effet limité à ses connaissances internes, souvent figées au moment de son entraînement, ce qui nuit à la pertinence des réponses dans des contextes où l'information évolue rapidement ou exige une grande précision (documents techniques, réglementaires, etc.).

Le concept de Retrieval-Augmented Generation (RAG) est alors apparu comme une solution novatrice pour pallier ces lacunes. En fusionnant la recherche d'informations dans une base documentaire et la génération de texte, il devient possible d'exploiter la puissance des LLM tout en s'appuyant sur des sources actualisées et adaptées aux besoins. C'est dans ce cadre que le projet SafeTrain a été conçu : il vise à proposer une plateforme web où l'utilisateur peut téléverser ses documents (uniquement PDF pour le moment), interroger l'IA et obtenir des réponses précises, basées non plus uniquement sur l'entraînement interne d'un modèle, mais aussi sur les données récentes issues de ses propres fichiers.

1.2 Problématique abordée

Le principal défi dans la conception d'un système de RAG réside dans la création d'une architecture efficace permettant la recherche d'informations et la génération de texte, tout en assurant la pertinence, la précision et la fiabilité des réponses fournies. Les LLM classiques, formés sur de larges corpus généralistes, peuvent manquer de précision ou d'exactitude lorsqu'on les interroge sur un sujet pointu. Ils sont également susceptibles d'« halluciner », c'est-à-dire de produire des informations incorrectes quand la question dépasse leur champ de connaissances.

Dans le cadre d'un projet où l'utilisateur charge lui-même un document (par exemple, un cahier des charges, un manuel technique ou un texte réglementaire), il est essentiel que le système ne s'égare pas. SafeTrain doit donc répondre à deux enjeux majeurs : d'une part, fournir un mécanisme de recherche à même de retrouver les parties pertinentes d'un document, et d'autre part, générer des réponses structurées et justes en s'appuyant sur ces éléments. Il faut en outre que l'ensemble reste accessible par le biais d'une interface web moderne, afin pouvoir être utiliser par des personnes ne maîtrisant pas forcément les environnements de développement Python et les notebooks.

1.3 Présentation générale du projet

SafeTrain a été conçue pour offrir à chacun la possibilité d'explorer et d'interagir directement avec ses documents PDF grâce à la technologie RAG. L'objectif est de rendre cette technologie accessible à tous via une interface conviviale et intuitive, utilisable depuis n'importe quel navigateur, sans nécessiter de compétences techniques particulières.

Les principales features de Safetrain sont :

- Permettre aux utilisateurs de dialoguer en temps réel avec le contenu de leurs documents et d'en extraire rapidement l'information recherchée.
- Exploitation d'un système d'agents configurables qui permet aux utilisateurs de créer, personnaliser et sauvegarder les paramètres clés d'un LLM basé sur RAG afin d'optimiser l'interaction avec leurs documents.
- Tester et comparer différentes techniques de RAG avancées (late chunking, reranking, hybrid search) afin d'améliorer la pertinence des réponses.

2 Définitions et Terminologies Clés

2.1 LLM (Large Language Model)

Un Large Language Model (LLM) est un modèle de traitement du langage naturel (NLP) entraîné sur d'immenses quantités de texte. Grâce à l'apprentissage profond (deep learning), ces modèles parviennent à comprendre (dans une certaine mesure) les requêtes des utilisateurs. En effet, ils sont capables de cerner la structure, la grammaire et le sens général de la langue. En outre, ils sont capables de générer du texte cohérent, produire des réponses, des résumés ou des traductions en s'appuyant sur ce qu'ils ont appris.

Concrètement, ils acquièrent une compréhension statistique du langage en analysant et en mémorisant des patterns, ce qui leur permet de générer des phrases cohérentes ou d'analyser et de classer du contenu. Un LLM assimile les règles de syntaxe, de grammaire et de sémantique à partir de ce vaste corpus d'entraînement, ce qui se traduit par la capacité de répondre à des questions ou de rédiger des textes avec une fluidité remarquable. Toutefois, ces modèles s'appuient sur une base de connaissances fondamentalement figée à la date où ils ont été entraînés. Dès lors, ils ne peuvent pas, sans mise à jour spécifique (réentraînement ou fine-tuning), intégrer les faits ou informations postérieurs à leur date de coupure. De plus, comme mentionné précédemment, il arrive parfois qu'un LLM "invente" des faits ou réponde de manière erronée (hallucination), faute de mécanismes internes robustes de vérification. Ainsi, les LLM bénéficient d'une puissance de génération et une polyvalence impressionnantes, mais présentent des limites en termes de données récentes, de fiabilité et d'explicabilité.

2.2 RAG (Retrieval-Augmented Generation)

Le *R*etrieval-Augmented Generation (RAG) vise à pallier certaines limites des LLM en y intégrant un module de recherche capable de récupérer, en temps réel, des informations depuis

une base de connaissances externe. Le fonctionnement repose d'abord sur la requête de l'utilisateur, qui déclenche un processus de recherche dans des documents. Les informations les plus adaptées, contextualisées et à jour sont ensuite extraites et transmises à la partie "génération" du modèle. Ainsi, au lieu de s'appuyer uniquement sur un bagage statique appris au préalable, le RAG exploite des ressources externes pour affiner et justifier ses réponses. Cette approche offre une actualisation permanente des contenus, réduit considérablement le risque d'hallucinations (réponses inventées) et permet de mieux tracer et justifier la provenance des informations fournies en les sourçant très clairement. Le RAG se révèle donc très efficace pour les entreprises ou organisations souhaitant utiliser leurs données internes sans devoir entièrement réentraîner un grand modèle pour chaque mise à jour.

2.3 Pipeline RAG

2.3.1 Le Concept d'Embedding

Les embeddings jouent un rôle crucial dans la pipeline RAG (Retrieval-Augmented Generation) en permettant une représentation vectorielle des mots, phrases ou documents dans un espace mathématique multidimensionnel. Contrairement à une représentation traditionnelle basée sur l'orthographe des mots ou leur indexation dans un dictionnaire, un embedding encode chaque entité textuelle sous la forme d'un vecteur de nombres réels. Par exemple, le mot « océan » peut être représenté par un vecteur tel que

$$(0.12, -0.03, 0.95, \dots)$$

Cette représentation vectorielle permet de capturer les relations sémantiques entre les différents éléments textuels. Par exemple, les phrases « J'adore les pommes » et « Je préfère les clémentines » auront des embeddings proches, reflétant leur similarité thématique, tandis qu'une phrase sur le nucléaire sera spatialement éloignée en raison de son contenu sémantiquement distinct.

L'utilisation des embeddings est essentielle pour le fonctionnement efficace de la pipeline RAG, en particulier pour la phase de récupération des informations pertinentes. Le *Retriever*, première composante de la pipeline, convertit la requête de l'utilisateur en un embedding. Cette transformation permet de comparer la question posée avec les documents de la base de connaissances en mesurant la similarité entre leurs vecteurs correspondants, souvent à l'aide de métriques telles que la distance cosinus ou le produit scalaire. Les documents dont les vecteurs sont les plus proches dans l'espace vectoriel sont alors identifiés comme les plus pertinents.

Cette capacité de comparer les significations plutôt que les simples correspondances lexicales permet une recherche plus intelligente et flexible. Par exemple, pour une requête telle que « Quelle est la recette d'un bon gâteau au chocolat ? », le *Retriever* peut identifier des documents pertinents même si ceux-ci utilisent des termes différents, comme un article de blog sur la pâtisserie, un extrait de livre de recettes ou un tutoriel de cuisine. Les embeddings permettent ainsi de capturer les nuances sémantiques, rendant la recherche plus robuste et contextuellement appropriée.

2.3.2 Étape 1 : Le Retriever (Récupération d'Informations)

La pipeline RAG se compose de deux étapes principales : la récupération d'informations pertinentes (*Retriever*) et la génération de la réponse (*Generator*).

La première étape de la pipeline RAG, le *Retriever*, consiste à transformer la requête de l'utilisateur en un embedding. Cette représentation vectorielle permet de comparer efficacement la question avec les documents de la base de connaissances. En calculant la similarité entre les vecteurs, le *Retriever* identifie les documents les plus pertinents. Par exemple, pour la question « Quelle est la recette d'un bon gâteau au chocolat ? », le *Retriever* pourrait sélectionner des documents tels qu'un article de blog sur la pâtisserie, un extrait de livre de recettes ou un tutoriel de cuisine. Ces documents sont ensuite récupérés en fonction de leur proximité vectorielle avec la requête initiale.

2.3.3 Étape 2 : Le Generator (Génération de la Réponse)

Une fois les documents pertinents récupérés, la deuxième étape implique le *Generator*, généralement un modèle de langage avancé comme GPT. Les textes extraits des documents sélectionnés sont concaténés ou résumés pour créer un contexte riche et pertinent. Le *Generator* utilise ce contexte, en conjonction avec la question de l'utilisateur, pour élaborer une réponse cohérente et précise. Par exemple, en se basant sur les informations spécifiques provenant des documents récupérés, le modèle peut générer une réponse détaillée telle que : « Les nouveaux parents reçoivent 20 jours de congé parental payés, ainsi qu'une couverture santé étendue pour l'enfant jusqu'à ses 2 ans. »

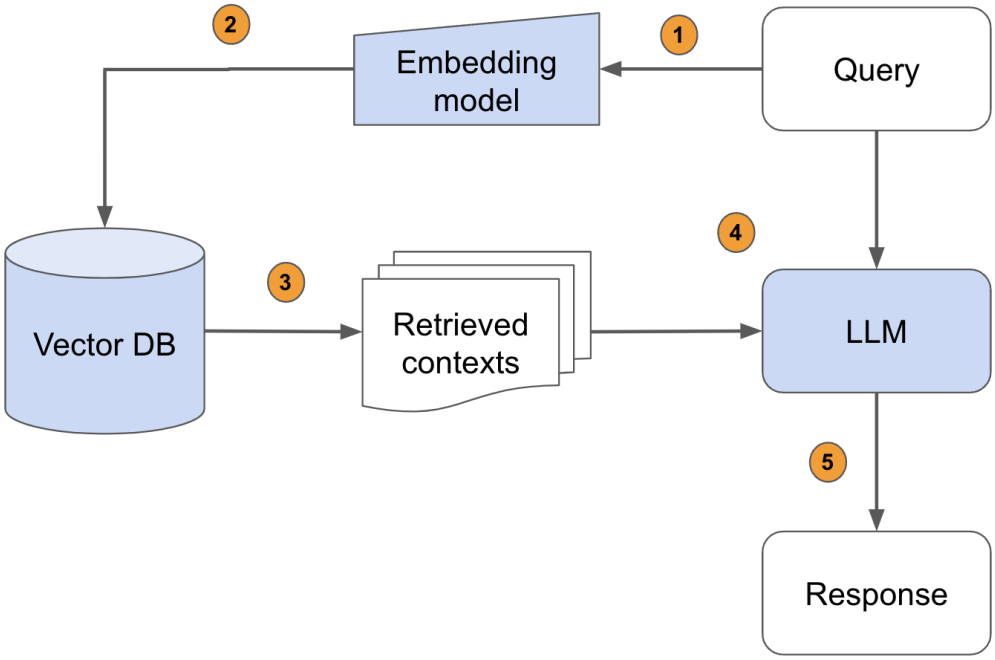


Figure 1: Schéma conceptuel d'une pipeline RAG

2.4 Les techniques de RAG avancés

Lorsqu'on met en place un système RAG (Recherche + IA Générative) de manière « classique », on se contente généralement d'extraire des passages via une simple recherche sémantique, puis de les transmettre au modèle de langage. Bien que cela fournisse déjà un certain niveau de pertinence, on constate souvent que cette approche seule est insuffisante pour couvrir un large éventail de cas d'usage et pour garantir des résultats de haute qualité.

En effet, il est fréquent de rencontrer des problèmes comme des réponses hors contexte, des informations manquantes ou, à l'inverse, une surabondance de contenu qui noie l'information utile. Pour remédier à ces limitations, on déploie des techniques de RAG avancées, qui permettent d'améliorer la précision, la pertinence et la concision des réponses générées.

Il existe de nombreuses stratégies avancées dans ce domaine. Toutefois, pour illustrer les principaux mécanismes d'amélioration, nous allons nous concentrer sur trois d'entre elles :

- **Late Chunking**
- **Reranking**
- **Hybrid Search**

Chacune de ces techniques vise un objectif précis, qu'il s'agisse d'affiner le découpage initial, de réordonnancer les résultats renvoyés, ou de compléter la recherche vectorielle par d'autres méthodes lexicales. Leur mise en œuvre fournit à la fois plus de robustesse et de flexibilité à un pipeline RAG, en offrant des moyens de répondre de façon plus fiable aux questions posées.

2.4.1 Late chunking

Le *late chunking* consiste à reporter le découpage du corpus en fragments (ou *chunks*) au moment où la requête est effectuée. Plutôt que de diviser le contenu en segments de taille fixe lors de l'indexation, on peut :

- **Adapter la taille des segments à la requête :** si la question est très spécifique (par exemple une référence légale), il peut être préférable de générer de plus petits segments, afin d'isoler la réponse exacte et d'éviter de diluer l'information dans de trop gros blocs. À l'inverse, pour des questions plus générales, un segment de plus grande taille pourra apporter davantage de contexte.
- **Prendre en compte la nature du contenu :** certains documents, comme les textes de loi ou les manuels techniques, se prêtent à un découpage logique (articles, sections, chapitres), qui n'a pas forcément la même granularité que des textes narratifs ou explicatifs.
- **Optimiser la cohérence textuelle :** en retardant le *chunking*, on peut appliquer des stratégies plus avancées pour préserver la cohérence d'un même sujet. Par exemple, une phrase ou un paragraphe qui complète une idée peut être conservé avec le bloc principal, plutôt que d'être artificiellement séparé.

Cette approche permet de disposer d'un ensemble de segments plus pertinents et mieux contextualisés pour le modèle de langage, améliorant ainsi les performances de *retrieval augmented generation*.

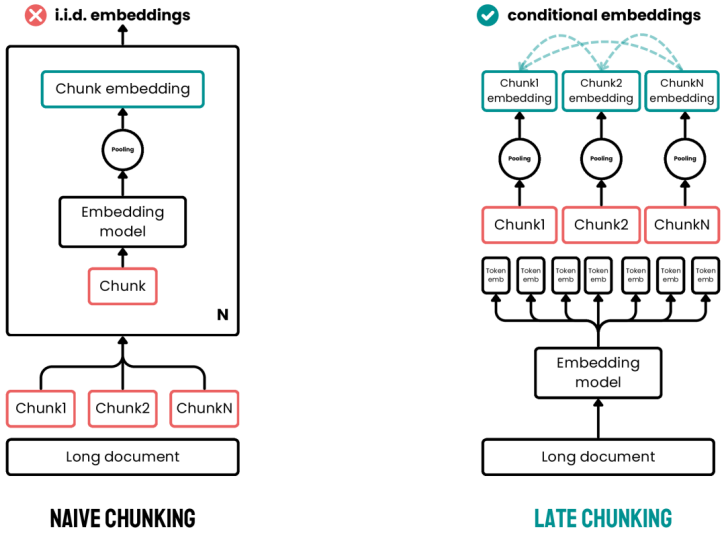


Figure 2: Schéma conceptuel du concept de late chunking

2.4.2 Reranking

Le *reranking* consiste à ajouter une étape supplémentaire de ré-ordonnancement des documents (ou des segments) renvoyés par le module de recherche sémantique. Concrètement :

- **Utilisation d'un second modèle de langage :** après la recherche initiale basée sur des *embeddings*, un autre modèle, généralement plus spécialisé ou plus finement entraîné, évalue la pertinence de chaque segment au regard de la requête.
- **Combinaison de scores :** ce second modèle ne se limite pas à la dimension sémantique stricte. Il peut par exemple prendre en compte un *score d'affinité linguistique*, c'est-à-dire la proximité lexicale ou la façon dont les mots clés apparaissent dans le segment, pour affiner davantage le classement.
- **Réduction du bruit :** cette étape permet de prioriser les résultats réellement utiles et de mettre à l'écart ceux qui, bien que sémantiquement voisins, n'offrent pas la meilleure réponse. Le *reranking* agit ainsi comme un filtre intelligent, garantissant un niveau de confiance plus élevé pour les segments les mieux notés.

En complétant la recherche initiale, le *reranking* augmente la pertinence des résultats finaux présentés au module de génération de texte.

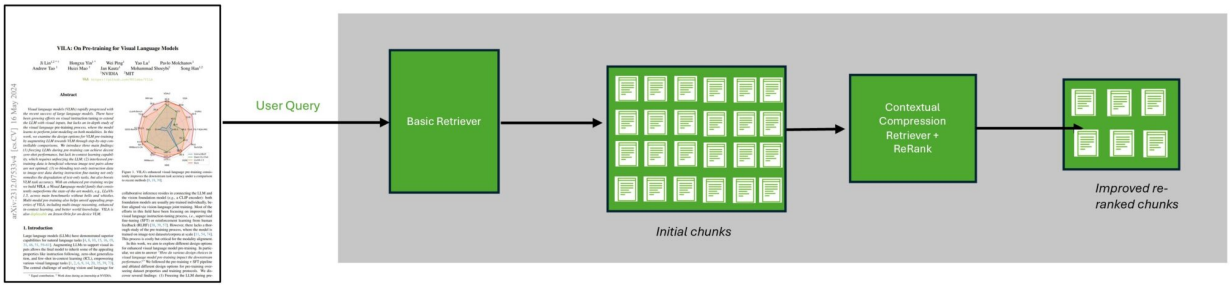


Figure 3: Schéma conceptuel du concept de reranking

2.4.3 Hybrid search

L'*hybrid search* combine deux méthodes de recherche complémentaires :

- **Recherche vectorielle (basée sur les embeddings) :** chaque segment est transformé en un vecteur dans un espace de représentation sémantique. Les segments les plus proches du vecteur de la requête sont considérés comme les plus pertinents.
- **Recherche lexicale (basée sur la concordance de mots-clés) :** on utilise également des algorithmes de recherche textuelle « classique » (e.g., TF-IDF, BM25) pour capturer l'information précise relative aux mots-clés exacts présents dans la requête.

Cette combinaison répond à des besoins variés :

- **Gestion des requêtes floues :** la recherche vectorielle est particulièrement efficace pour capter le sens global d'une question ou pour répondre à des requêtes « ouvertes » (questions conceptuelles, paraphrases, synonymes, etc.).
- **Références textuelles exactes :** la recherche lexicale est plus adaptée lorsque l'utilisateur cherche une citation légale ou une expression exacte, où les mots doivent correspondre précisément.
- **Complémentarité des approches :** associer ces deux modalités permet de couvrir un large éventail de cas d'usage. Par exemple, si la requête contient à la fois une référence clé (un numéro d'article, un nom spécifique) et une dimension sémantique plus vague, l'*hybrid search* s'assure que ces deux types d'informations soient pris en compte.

Grâce à l'*hybrid search*, on obtient ainsi un ensemble de documents à la fois sémantiquement alignés avec la requête et capables de fournir des correspondances exactes de mots-clés.

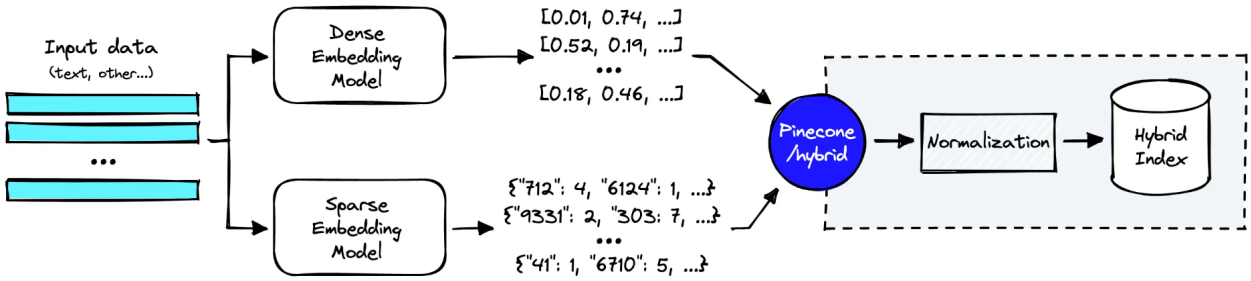


Figure 4: Schéma conceptuel du concept de hybrid search

2.5 Pourquoi un RAG plutôt qu'un LLM Basique ?

À la différence d'un LLM basique, qui se limiterait à ses propres connaissances internes, un RAG rend possible une adaptation quasi immédiate à des documents nouveaux ou spécialisés. Il fournit une meilleure robustesse face aux interrogations sur des textes précis, limitant ainsi les risques de réponses fantaisistes ou inexactes. Dans des secteurs comme le juridique ou l'industrie lourde, les textes de référence et réglementations changent régulièrement. Pouvoir mettre à jour la base documentaire sans réentraîner tout un modèle est un atout considérable.

Par ailleurs, le RAG facilite la mise en place de processus de vérification et de traçabilité des sources, offrant ainsi une transparence accrue sur l'origine des informations et renforçant la confiance dans la qualité des données fournies. En outre, cette approche permet de personnaliser le contenu documentaire en fonction des besoins spécifiques des utilisateurs ou des secteurs d'activité.

Enfin, l'architecture modulaire d'un système RAG favorise l'intégration harmonieuse avec d'autres outils d'analyse et de gestion de contenu, ce qui ouvre la voie à des solutions sur mesure et évolutives, adaptées aux exigences toujours changeantes du marché et aux défis spécifiques de chaque industrie.

3 Stack Technique

3.1 Frontend

Pour le développement de la partie frontend de SafeTrain, le choix a été porté sur Next.js, TypeScript et Tailwind CSS pour leurs avantages complémentaires. Next.js offre des fonctionnalités puissantes comme la génération de pages statiques et dynamiques, essentielles pour optimiser les performances et le référencement. TypeScript garantit une meilleure robustesse du code grâce au typage statique, réduisant ainsi les erreurs lors du développement. Tailwind CSS, quant à lui, permet une conception rapide et réactive grâce à son système de classes utilitaires, tout en maintenant une grande flexibilité pour la personnalisation du design. Le maquettage de l'interface utilisateur a été réalisé en amont sur Figma, ce qui a permis de définir une base visuelle claire et cohérente pour le développement.

3.2 Backend

Le backend se divise entre la logique métier et la gestion des données. L'application Node.js (ou les routes internes de Next.js) pilote les tâches d'authentification, de routage et de coordination avec l'infrastructure IA. Neon DB, un service Postgres « serverless », sert à stocker et gérer les informations structurées comme les profils des utilisateurs ou l'historique des requêtes. Pour les fichiers PDF eux-mêmes, Amazon AWS S3 est utilisé pour stoquer les PDF dans le Cloud.

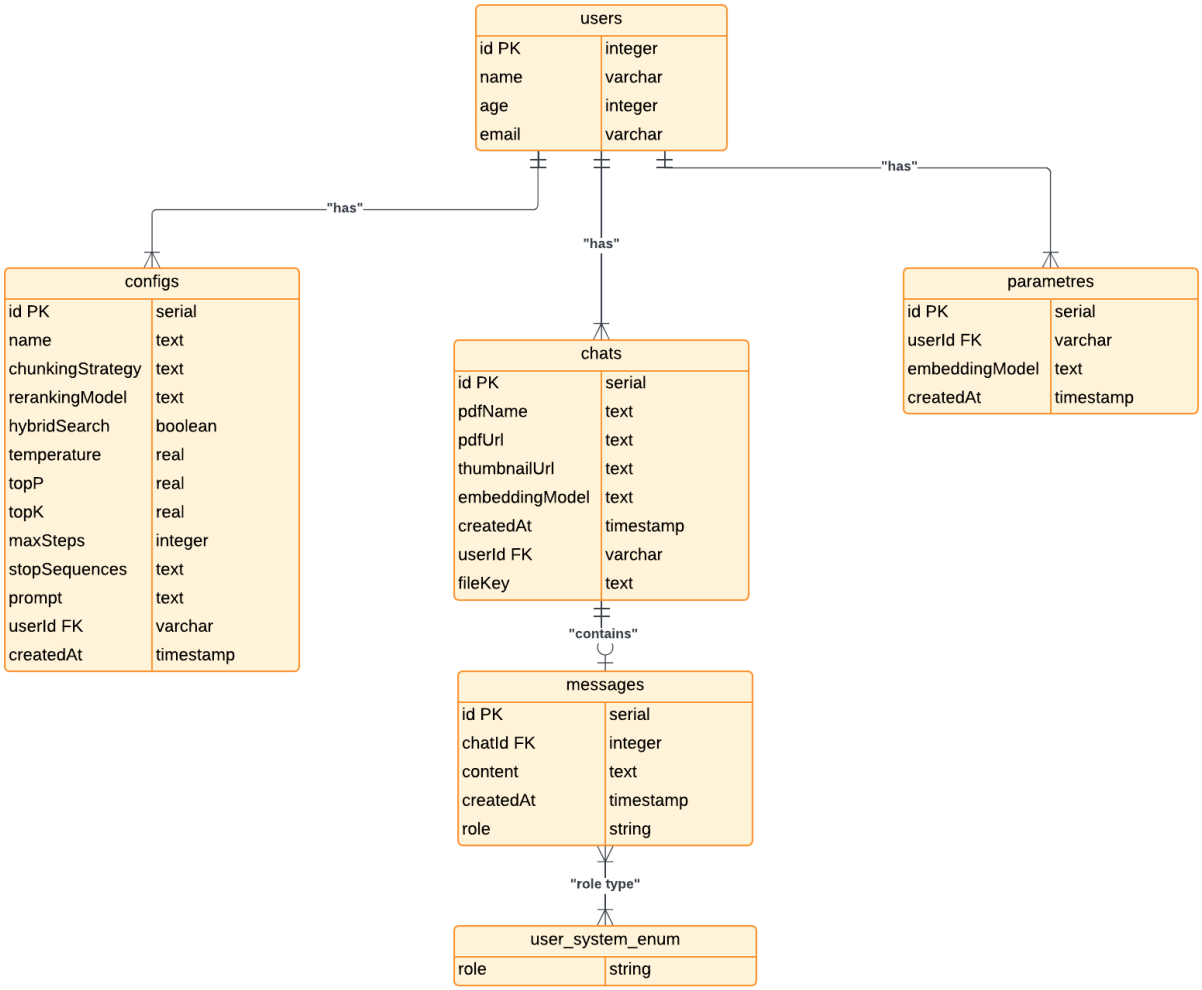


Figure 5: Schéma conceptuel de la base de données NeonDB

Concernant le schéma de la base de donnée NeonDB, il contient :

- **Table `users`** : centralise les informations des utilisateurs, avec des champs pour l'identifiant, le nom, l'âge et l'email (unique).

- **Table `chats`** : gère les interactions liées aux fichiers PDF, incluant des champs pour le nom du PDF, son URL, le modèle d'embedding utilisé et l'identifiant de l'utilisateur associé.
- **Table `messages`** : enregistre les messages échangés dans chaque chat, avec des informations sur le contenu, le rôle (utilisateur ou système), et la date de création.
- **Table `configs`** : stocke les configurations personnalisées des utilisateurs, comme les stratégies de découpage, les modèles de reranking, les paramètres des modèles IA (température, top-p, top-k) et les séquences d'arrêt.
- **Table `parametres`** : contient les préférences utilisateur, notamment le modèle d'embedding sélectionné, ainsi que la date de création.

En outre, la base de donnée vectorielle PineconeDB est au centre du mécanismes de RAG : son fonctionnement sera expliqué dans la prochaine section.

Cette séparation des rôles simplifie la maintenance et facilite la scalabilité, car on peut accroître les ressources affectées à PineconeDB indépendamment de celles du serveur ou de la base relationnelle.

De plus, cette architecture backend est conçue pour être modulaire, de sorte qu'on puisse remplacer un composant par un autre (S3, NeonDB et PineconeDB par un stockage local, par exemple) si on souhaite un déploiement *full local* pour des raisons de confidentialité plus strictes.

3.3 Stack IA

La stack IA repose sur PineconeDB. Ce dernier héberge ces fameux *embeddings* générés par des modèles de transformation du texte (transformers).

Pour interagir avec ces données, le AI SDK de Next.js est utilisé, offrant des fonctionnalités avancées pour exécuter des requêtes (queries) et du streaming de contenu en temps réel.

Concernant les modèles de génération des embeddings, SafeTrain utilise les suivants :

- **text-embedding-ada-002**
  - Issu de la famille GPT d'OpenAI.
  - Génère des embeddings avec une dimension de 1536.
  - Reconnue pour sa haute précision et sa performance sur des tâches complexes.
- **text-embedding-3-small**
  - Également issu de la famille GPT.
  - Génère des embeddings avec une dimension de 768.
  - Optimisé pour des scénarios nécessitant moins de ressources et des résultats rapides.

Ces modèles offrent une flexibilité dans la gestion des performances et des ressources, permettant à SafeTrain de s'adapter à divers cas d'utilisation selon les besoins.

Enfin, pour les modèles de LLM, la stack supporte les modèles suivants :

- `GPT-4o`
- `GPT-4o-mini`
- `GPT-4-turbo`
- `Grok-2-1212`
- `Grok-beta`
- `Gemini 2.0 Flash Exp`
- `Gemini 1.5 Flash Latest`
- `Gemini 1.5 Flash`

Ces modèles sont sélectionnés pour répondre à des besoins variés, qu'il s'agisse de vitesse, de complexité ou d'efficacité en mémoire, offrant ainsi une infrastructure robuste et flexible pour les applications basées sur l'IA.

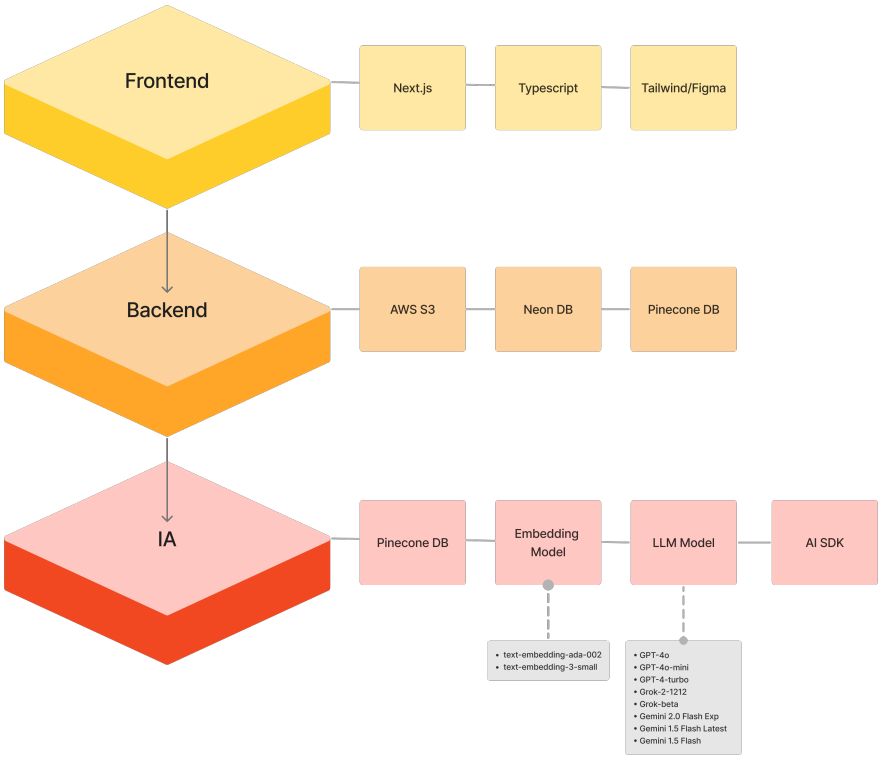


Figure 6: Schéma récapitulatif de l'architecture technique de SafeTrain.

4 Fonctionnalités Développées

4.1 Chat avec un PDF

La fonctionnalité principale de SafeTrain est la possibilité de discuter avec un ou plusieurs documents PDF. Lorsqu'un utilisateur téléverse un fichier, ce document est segmenté et indexé, ce qui autorise par la suite une interrogation instantanée. Par exemple, si un manuel technique de plusieurs centaines de pages est chargé, l'utilisateur peut demander : « Quels sont les prérequis de sécurité pour la maintenance ? » et obtenir une réponse renvoyant aux pages concernées, plutôt que de feuilleter manuellement le texte. Cette capacité d'interroger un document de manière conversationnelle démontre tout l'intérêt d'un RAG, car l'IA puise directement dans la source d'information adaptée au contexte.

4.2 Techniques avancées de RAG

Pour atteindre un niveau de pertinence supérieur dans la génération et la récupération d'informations, SafeTrain intègre plusieurs techniques avancées de Retrieval-Augmented Generation (RAG). Ces techniques permettent d'améliorer la précision et la fiabilité des réponses en combinant des méthodes optimisées pour la gestion des données et leur traitement. Trois approches principales sont mises en œuvre :

- **Late Chunking** : Réorganise les passages extraits en vue de faire remonter les plus perti-nents.
- **Reranking** : Découpe a posteriori de plus gros segments pour une meilleure granularité.
- **Hybrid Search** : Combine la recherche vectorielle (Pinecone) et la recherche lexicale (BM25).

Ces trois techniques seront détaillées dans la suite du document, où leur implémentation et leur intégration au sein de la plateforme SafeTrain seront expliquées étape par étape.

4.3 Création et gestion d'agents personnalisés

SafeTrain propose un mécanisme d'agents paramétrables, permettant à l'utilisateur de définir différentes configurations pour ajuster finement le comportement du modèle. Chaque agent est sauvegardé et peut être rappelé à tout moment, offrant ainsi la possibilité de tester plusieurs réglages ou de s'adapter à divers types de documents, sans avoir à tout reconfigurer à chaque nouvelle session.

4.3.1 Paramètres du modèle LLM

Voici les principaux réglages accessibles pour chaque agent :

- **Température** : Contrôle la créativité et la variabilité des réponses.

  - Domaine de définition : Généralement compris entre 0 et 2 (valeur par défaut proche de 0.7 selon certains modèles).
  - *Valeurs basses* (≈ 0) : Le modèle devient très déterministe, répétant souvent les mêmes formules et mots.
  - *Valeurs hautes* ($\approx 1.5 - 2$) : Les réponses deviennent plus inventives et variées, mais parfois moins cohérentes.

- **Top-k** : Contrôle le nombre de tokens candidats considérés à chaque étape de génération.
  - *Domaine de définition* : Varie en fonction du modèle (souvent de 1 à quelques dizaines).
  - *Valeur basse* (≈ 1) : Le modèle choisit uniquement le token le plus probable à chaque étape, limitant la diversité de la réponse.
  - *Valeur haute* (> 10) : Plus de candidats sont pris en compte, ce qui rend la génération plus variée mais aussi plus incertaine.

- **Top-n** : Similaire à *top-k*, ce paramètre peut limiter davantage le nombre de tokens examinés ou leur probabilité cumulée. Selon la bibliothèque utilisée, *top-n* peut parfois fusionner avec d'autres critères (p. ex. *top-p*). La logique est néanmoins comparable : on fixe un seuil pour restreindre la distribution d'où est prélevé le prochain token.

- **Max-steps** : Définit le nombre maximal d'itérations (ou de tokens) que le modèle peut générer avant d'arrêter.
  - *Domaine de définition* : De quelques dizaines à plusieurs milliers, selon la longueur de contexte désiré.
  - *Valeur basse* (≈ 50) : Les réponses sont courtes, ce qui peut s'avérer pertinent pour des réponses factuelles et directes.
  - *Valeur haute* (> 500) : Autorise des réponses plus longues et détaillées, au risque de produire des digressions.

- **Stopwords** : Liste de mots ou de phrases qui, lorsqu'ils sont générés, entraînent l'arrêt de la génération du texte.
  - *Domaine de définition* : Une liste personnalisée de mots ou de séquences de caractères définis par l'utilisateur.
  - *Utilisation* : Permet de contrôler la fin des réponses générées, évitant ainsi des contenus indésirables ou inappropriés.

  - *Exemple* : Dans un contexte de résumé, on peut définir des stopwords tels que "`FIN`" ou "`Résumé terminé`" pour indiquer la fin du résumé.

- **Prompt** : Texte initial fourni au modèle pour orienter la génération de la réponse.
  - *Domaine de définition* : Chaîne de caractères qui sert de contexte ou d'instruction pour le modèle.
  - *Prompt engineering* : C'est ici que l'on intègre des techniques avancées de conception de prompts pour optimiser les performances du modèle. Ces techniques permettent de guider le modèle de manière plus précise et efficace selon les besoins spécifiques de la tâche.
  - *Exemples de stratégies de prompt engineering* :
    * **Few-shot learning** : Fournir quelques exemples de la tâche souhaitée dans le prompt pour guider le modèle.
    * **Chain-of-thought** : Encourager le modèle à décomposer sa réflexion en plusieurs étapes logiques avant de fournir une réponse finale.
    * **Instruction tuning** : Donner des instructions claires et détaillées sur ce qui est attendu dans la réponse.
    * **Role prompting** : Définir un rôle spécifique pour le modèle (par exemple, "Vous êtes un expert en maintenance technique") afin de contextualiser les réponses.

4.3.2 Sélection des techniques avancées de RAG

Pour chaque agent, l'utilisateur peut activer ou désactiver certaines composantes de la pipeline. Il s'agit des 3 techniques de RAG avancés : *Reranking*, *Late chunking*, *Hybrid search*.

Chaque agent est enregistré et réutilisable dans le temps, offrant ainsi une grande flexibilité pour tester différents réglages ou pour aborder différents types de documents sans devoir tout reconfigurer.

5 La Pipeline de Safetrain

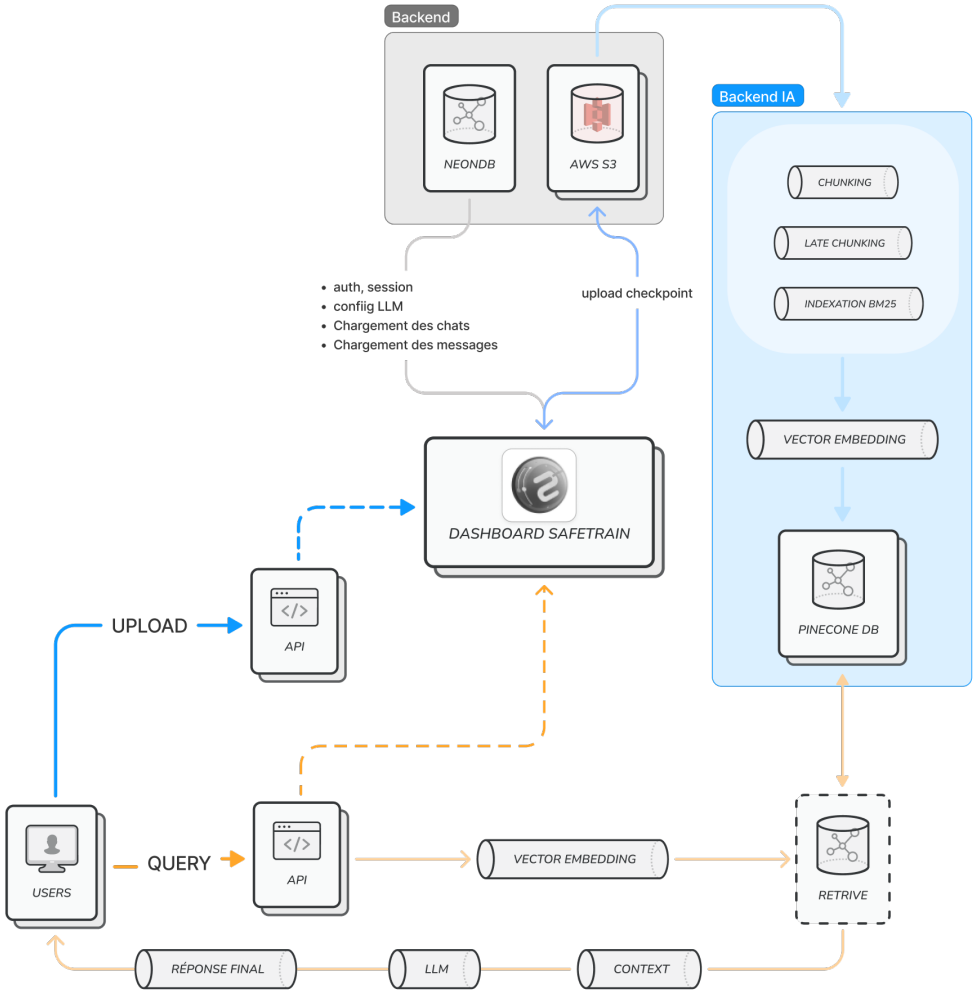


Figure 7: Schéma conceptuel de la pipeline Safetrain IA

5.1 Upload de la data (et preprocessing)

5.1.1 Datasets utilisés

Lors de la conception de la pipeline SafeTrain, toutes sortes de documents ont été utilisés, tels que nos cours de NLP et Big Data, des rapports scientifiques, des rapports

médicaux, des guides, des présentations Powerpoint, et des tableaux complexes. Cependant, Safetrain a une contrainte : il n'est pas possible d'upload plus de 50 documents en simultané sans souscrire à un abonnement Pinecone.

5.1.2 Envoi des fichiers sur Amazon S3.

Les utilisateurs souhaitent traiter des documents (souvent des fichiers PDF). Dans un premier temps, ces fichiers sont chargés et stockés dans un *bucket* S3 via une clé unique permettant de retrouver le fichier ultérieurement.

5.1.3 Création de la vignette PDF.

Pour chaque fichier PDF, le système génère une miniature représentant la première page du document, ce qui facilite par la suite sa visualisation et sa navigation. Cette image est elle aussi conservée dans S3.

5.1.4 Extraction et nettoyage du texte.

Le contenu brut du PDF est analysé pour en extraire du texte. Des opérations de nettoyage sont effectuées : suppression de sauts de ligne superflus et harmonisation des caractères. Cet ensemble d'actions correspond à la phase de *preprocessing*.

5.1.5 Découpage (*chunking*).

Le texte est alors divisé en segments de taille contrôlée (chunks). Deux grandes stratégies coexistent :

- **Standard chunking :** génération de segments de taille moyenne, en introduisant un léger chevauchement pour ne pas perdre de contexte.
- **Late chunking :** création de chunks plus volumineux, qui seront ensuite redécoupés au besoin lors de la requête.

Le découpage permet d'éviter de faire des requêtes trop longues et de maximiser la pertinence au moment de la recherche vectorielle.

5.2 Indexation dans PineconeDB

5.2.1 Extraction des embeddings.

Chaque chunk issu du document est transformé en un vecteur numérique (embedding) grâce à un modèle spécialisé. L'utilisateur a le choix d'utiliser le modèle text-embedding-ada-002 ou text-embedding-3-small de GPT. Cependant, il est impératif qu'il utilise le même modèle d'embedding dans la phase d'uploading du document et

de query pour ne pas créer de conflits. Cette représentation vectorielle capture ainsi la sémantique de l'ensemble des chunks et donc du document.

5.2.2 Création des enregistrements (records).

Le système génère alors des enregistrements, chacun associé à :

- Un identifiant (un hachage créé à partir du contenu),
- Le vecteur calculé par le modèle d'embedding,
- Des métadonnées (*metadata*), qui incluent le numéro de page, le nom du fichier, ou encore le texte d'origine tronqué pour éviter les dépassements de taille.

5.2.3 Insertion dans Pinecone.

Les enregistrements ainsi construits (un vecteur + des métadonnées) sont insérés dans l'index hébergé par Pinecone. Ce dernier se présente comme un service entièrement géré pour le stockage et la recherche de vecteurs de grande dimension, permettant d'effectuer rapidement des requêtes de similarité sur des ensembles volumineux.

Fonctionnement général de Pinecone. Pinecone agit comme une base de données orientée vecteurs, dans laquelle chaque élément (appelé *vector record*) est identifié par un *id* et enrichi par un vecteur numérique. Contrairement à une base de données relationnelle classique, la priorité est ici donnée aux opérations de *similarité vectorielle* : lorsque l'on interroge Pinecone avec un nouveau vecteur (par exemple, un embedding issu d'une question), celui-ci renvoie les enregistrements les plus proches selon une métrique de distance choisie.

Index et namespaces. Dans Pinecone, le *stockage* est organisé sous forme d'un ou plusieurs *index* indépendants. Chaque index est caractérisé par un certain nombre de paramètres : la métrique de similarité, la dimension du vecteur et le mode de capacité. Par ailleurs, il est possible de définir des *namespaces* au sein de l'index : chaque namespace est une subdivision logique permettant de regrouper des ensembles de documents distincts (par exemple, un namespace pour un certain fichier ou pour une configuration spécifique de découpage).

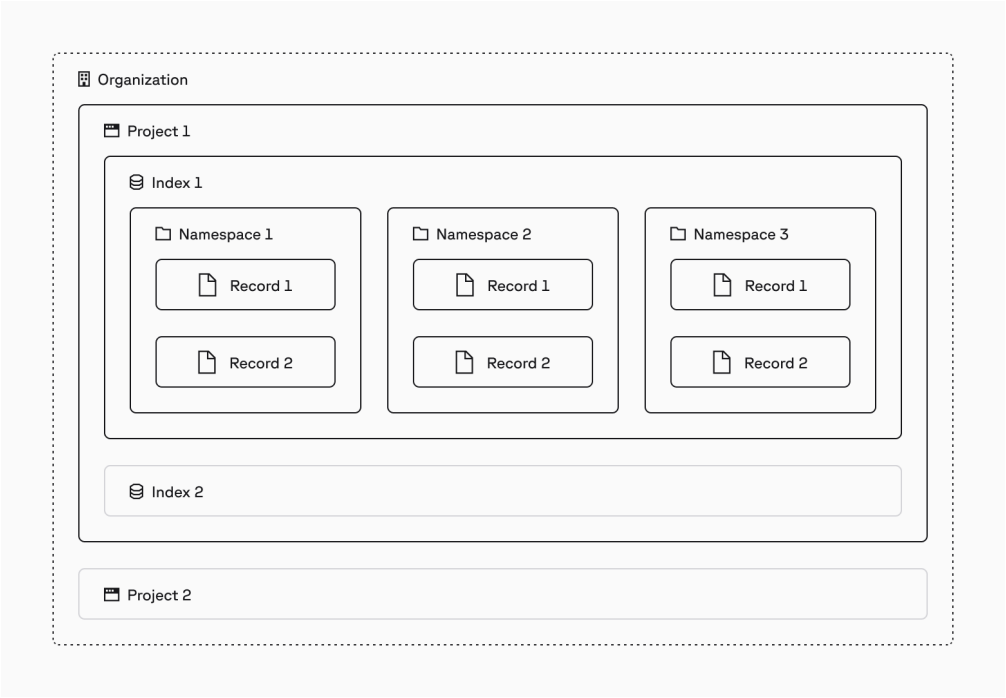


Figure 8: Configuration d'un projet Pinecone

Détails sur la configuration de l'index Pinecone. Dans le projet Safetrain, l'index Pinecone est paramétré de la manière suivante :

- **Métrique** : *cosinus*.

  Cette métrique calcule la proximité entre deux vecteurs en évaluant l'angle qui les sépare. La similarité cosinus est fréquemment utilisée pour mesurer la ressemblance sémantique.

- **Dimension** : 1536.

  Ce choix reflète la taille des vecteurs produits par le modèle d'embedding.

- **Cloud** : *aws*, dans la région `us-east-1` (les autres régions sont payantes).

  Le service Pinecone est donc hébergé sur l'infrastructure AWS pour cette région.

- **Mode de capacité** : *Serverless*.

  Il n'est pas nécessaire de configurer manuellement la taille des ressources ; Pinecone alloue automatiquement les ressources en fonction des demandes.

Au même titre que le choix des dimensions, le choix de la *métrique cosinus* est elle aussi intiallement lié au modèle d'embedding utilisé, tandis que le mode *Serverless* procurent au projet une haute disponibilité sans nécessiter de gestion d'infrastructure.

Par ailleurs, l'usage des namespaces permet de séparer, au sein du même index, plusieurs lots de données (par exemple, en distinguant le *chunking standard* et le *late chunking*). Chaque namespace agit donc comme un espace clos dans lequel les recherches sont

effectuées indépendamment des autres. Dans na version gratuite, Pinecone limite le nombre de namespace maximum à 100.

Processus d'insertion (upsert). Lors de l'insertion, le système exécute une requête d'*upsert* vers l'index Pinecone, en précisant :

1. **l'identifiant du chunk** (généré par une fonction de hachage),
2. **le vecteur** (embedding) de dimension 1536,
3. **les métadonnées** : texte tronqué, numéro de page, nom du fichier, etc.

Une fois insérés, ces enregistrements deviennent interrogeables via l'API de Pinecone, qui se charge de leur distribution et de leur indexation interne, *sans intervention manuelle* supplémentaire.

Cette modularité rend le système plus flexible : il est possible de supprimer un document (et donc son namespace associé) ou de mettre à jour un lot de vecteurs sans affecter le reste des données. Lorsqu'une requête est faite, on *cible* explicitement un namespace de l'index pour n'y retourner que les enregistrements de ce sous-ensembles.

Enfin, lors de la phase d'upload d'un document, Safetrain insère systématiquement 2 namespace dans son index. Un index permettant d'utiliser la stratégie de chunking standard, et un autre permettant l'utilisation de late chunkin. Safetrain peut donc héberger un maximum de 50 documents dans sa pipeline.

5.2.4 Gestion d'un index BM25 (recherche lexicale).

En complément de l'approche sémantique, une indexation textuelle de type BM25 est réalisée et stockée sur S3. Cela permet une *recherche hybride* : d'un côté la similarité vectorielle (embeddings) et de l'autre la recherche lexicale. Comme expliqué précédemment, le système consolide les résultats provenant de ces deux méthodes pour maximiser la qualité.

5.3 Technique de Rag avancés

5.3.1 Late chunking

SafeTrain introduit le *late chunking* en combinant deux voies :

- Un découpage initial dit « standard », où les documents sont déjà indexés en fragments de taille modérée.
- Une fonctionnalité de re-découpage *à la volée*, qui segmente localement les réponses les plus pertinentes en plus petits sous-chunks au moment de l'interrogation.

Au sein du code, on utilise un `RecursiveCharacterTextSplitter` avec des paramètres spécifiques (`chunkSize`, `chunkOverlap`) ajustés au contexte. Le principe fondamental du RecursiveCharacterTextSplitter est de découper le texte de manière récursive en respectant des limites logiques et syntaxiques, telles que les phrases, les paragraphes ou d'autres délimiteurs spécifiques. Lorsque l'utilisateur envoie sa requête, l'algorithme calcule des *embeddings* pour ces plus petits segments, recalcule la similarité et ne retient que les sous-chunks les plus pertinents. De cette façon, SafeTrain préserve la cohérence textuelle, tout en ajustant finement la taille des segments aux besoins de la requête.

5.3.2 Reranking

SafeTrain utilise une fonction spéciale pour cette étape. Après l'appel initial à Pinecone ou BM25, la liste de segments retenus est transmise à cette fonction de ré-ordonnancement. Celle-ci :

- Calcule un nouveau score de pertinence pour chaque segment, parfois en s'appuyant sur un modèle plus spécialisé (ou en combinant score sémantique et score lexical).
- Classe ensuite les segments selon ce nouveau score, en faisant remonter vers le haut du classement les passages jugés comme particulièrement répondant à la requête.

Grâce à ce processus, SafeTrain réduit le risque de renvoyer du contenu non pertinent : même si un segment a initialement été jugé « relativement proche », il peut être relégué en bas du classement (voire exclu) au profit d'autres fragments mieux adaptés à la requête.

5.3.3 Hybrid search

SafeTrain procède à une *hybrid search* en fusionnant les résultats de recherche vectorielle (via Pinecone) et de recherche lexicale (via `wink-bm25-text-search`). Concrètement :

- Il interroge d'abord Pinecone pour récupérer les segments les plus proches en termes de sémantique (`getMatchesFromEmbeddings`).
- Ensuite, il lance une requête BM25 pour extraire les passages particulièrement pertinents en termes de mots-clés (`getBM25Matches`).
- La fonction `applyHybridSearch` assemble les deux résultats pour les regrouper dans une même liste, puis applique éventuellement un *reranking* supplémentaire pour ordonner le tout.

De cette façon, SafeTrain couvre un large spectre de cas d'usage : des requêtes très précises avec des mots-clés définis, comme des requêtes plus ouvertes qui reposent sur la proximité sémantique.

5.4 Interaction avec les LLM (Query)

5.4.1 Réception de la requête.

L'utilisateur formule une question (prompt). Elle est convertie en embeddings grâce au modèle d'embedding spécifié par l'utilisateur.

5.4.2 Comparaison et récupération du contexte pertinent.

Le vecteur issu de la requête est envoyé à Pinecone, qui renvoie les chunks les plus proches dans l'espace vectoriel. Ce vecteur sert donc de point de comparaison pour trouver les fragments de texte les plus pertinents dans PineconeDB. En parallèle, l'index BM25 peut être consulté si la recherche hybride est activée. Lorsque la stratégie *late chunking* est utilisée, les chunks renvoyés par Pinecone sont eux-mêmes redécoupés et rescorrés localement pour en affiner la pertinence.

5.4.3 Reranking.

Les résultats (simples ou hybrides) subissent un tri supplémentaire (*reranking*), qui peut faire intervenir un modèle d'évaluation supplémentaire pour reclasser les passages reçus, assurant que seuls les extraits les plus pertinents seront utilisés pour la phase de génération de réponse.

5.5 Gestion des réponses générées par l'IA

5.5.1 Le Prompting et l'assemblage du contexte

Dans Safetrain, l'envoi final vers le modèle de langage s'appuie sur un *template* de prompt. Le système combine :

1. Un message *systemPrompt*, qui précise le rôle de l'assistant (spécialiste en recherche documentaire) et la manière dont il doit citer les sources ou reconnaître qu'il manque d'information.
2. Un bloc de *contextPrompt*, c'est-à-dire la concaténation des extraits jugés pertinents (avec leurs sources). L'assistant est instruit de ne pas inventer d'information qui ne figurerait pas dans ces extraits.
3. Une *note externe*, qui n'est ajoutée au prompt que si la *variable isRAG* est désactivé (elle est activée par défaut). Cette fameuse note externe autorise l'assistant à s'appuyer également sur sa propre connaissance du monde s'il le juge nécessaire. Le modèle n'est donc plus un Rag à proprement parler.
4. Une *Le prompt de l'agent*, qui est ajouté au prompt final si l'utilisateur Safetrain a fournis à son agent un prompt spécifique. C'est précisément ce prompt qui pourra introduire des techniques de prompt engineering spécifique à la tache que l'on

souhaite assignée à notre agent. Parmis ces techniques, on trouve notamment le Role Prompting, le Few-shot Prompting, l'Instruction-based Prompting, le Chain-of-Thought Prompting, et le Self-Ask Prompting.

5.5.2 Génération du texte final.

Le modèle de langage, muni du contexte et des consignes, retourne une réponse. Cette réponse est alors mise à disposition de l'utilisateur comme texte généré.

5.5.3 Enregistrement des échanges dans NeonDB.

Pour conserver l'historique des conversations et permettre un suivi (analyse, traçabilité, etc.), chaque question et chaque réponse sont sauvegardées dans la base de données NeonDB. Cette logique offre une vue d'ensemble de l'activité de l'IA et de l'utilisation des documents.

5.5.4 Mises à jour et suppressions.

Si un document est supprimé, le système efface également l'objet S3 correspondant. Les éventuels *namespaces* Pinecone associés sont nettoyés, et l'index BM25 sur S3 est retiré. Ainsi, les données et les index restent cohérents, et le système ne sert plus de réponses basées sur un document qui n'existe plus.

En résumé, la pipeline Safetrain met en place un écosystème complet pour la recherche de documents et la génération de réponses expertes :

- **Upload sur S3** et **prétraitement du contenu**,
- **Indexation vectorielle** et **recherche hybride** (Pinecone + BM25),
- **Interaction** avec les LLM s'appuyant sur un *prompt* structuré,
- **Gestion** des résultats et de l'historique dans la base NeonDB.

Chaque requête bénéficie ainsi d'un contexte documentaire adapté, offrant des réponses plus fiables et mieux référencées.

5.6 Les différentes URL

Afin de mieux comprendre l'architecture et les fonctionnalités offertes par Safetrain, voici une présentation des principales sections disponibles dans l'application. Chaque section joue un rôle spécifique dans la gestion des documents, la configuration des agents, et le paramétrage des pipelines RAG. La liste suivante détaille les différentes URL et leurs utilisations correspondantes :

- **safetrain/ :** Page de connexion/inscription. L'utilisateaur peut se connecter via son compte Gmail, Github ou bien GitLab.
- **safetrain/dashboard :** Regroupe l'ensemble des documents upload par l'utilisateur.
- **safetrain/dashboard/chat/chatid :** C'est l'endroit où l'on discute avec un seul document, avec la possibilité de choisir un agent pour configurer les paramètres intrinsèques du LLM et de la pipeline RAG.
- **safetrain/playground :** Permet de discuter avec l'ensemble des documents de l'utilisateur et constitue l'interface de création des agents, en choisissant les configurations qui nous intéressent.
- **safetrain/parametres :**
  * Ici, l'utilisateur choisit le modèle d'embedding, d'upload et de requête de la pipeline RAG.
  * Pour garantir la cohérence et l'efficacité du système RAG tel que Safetrain, il faut utiliser le même modèle d'embedding pour l'ensemble du processus. Il faut donc supprimer tous les documents uploadés avant de changer le modèle utilisé pour les embeddings.
  * Pour le moment, il est conseillé d'utiliser uniquement `text-embedding-ada-002`, car il offre des performances bien supérieures à `text-embedding-3-small`.
- **safetrain/references :** Contient le rapport, la diapo et la démo vidéo de la solution Safetrain.

6 Cas d'Usage

6.1 Retrieval

Le Retrieval est la première étape cruciale qui consiste à extraire, parmi un vaste ensemble de documents (manuels techniques, articles, etc.), les informations les plus pertinentes pour répondre à une requête ou accomplir une tâche spécifique. SafeTrain optimise cette phase grâce à deux approches complémentaires:

1. **Indexation intelligente**: Les documents sont segmentés en unités de texte (phrases ou paragraphes) et indexés dans une base de données spécialisée. Cette structuration facilite l'accès rapide aux informations nécessaires.
2. **Recherche contextuelle**: Lorsqu'un utilisateur soumet une question ou une demande de résumé, l'IA compare la requête aux unités indexées pour sélectionner celles contenant les informations les plus adaptées.

Ce mécanisme offre plusieurs avantages: il permet aux utilisateurs d'accéder rapidement aux extraits essentiels sans avoir à parcourir manuellement des pages entières, assurant ainsi un gain d'efficacité. De plus, les résultats sont filtrés pour correspondre étroitement à la requête initiale, garantissant une pertinence assurée et minimisant les

risques de confusion ou de perte de temps. Enfin, il facilite la veille documentaire en ciblant précisément les passages utiles dans un grand volume de textes, que ce soit pour des recherches scientifiques, légales ou techniques.

Pour résumer un document volumineux par exemple, SafeTrain réalise un Retrieval des passages clés, souvent les paragraphes contenant les idées principales ou les concepts récurrents. Cette étape prépare le terrain pour la phase suivante, le Completion. Lors de ce processus, chaque chunk se voit attribuer un score basé sur sa pertinence par rapport à la requête de l'utilisateur. Ces scores sont ensuite filtrés pour retenir uniquement les chunks les plus pertinents. De cette manière, SafeTrain s'assure que seules les informations les plus significatives et contextuellement appropriées sont sélectionnées.

6.2 Completion

La Completion intervient après le Retrieval et consiste à générer une réponse ou un contenu final à partir des informations collectées. SafeTrain utilise des modèles de langage avancés pour:

1. **Analyser le contexte**: L'IA prend en compte la question ou la demande initiale (par exemple «Rédiger un résumé» ou «Expliquer comment réparer un composant»).
2. **Synthétiser l'information**: Les extraits sélectionnés sont reformulés et organisés de manière cohérente et compréhensible.
3. **Fournir une réponse adaptée**: Que ce soit un résumé concis, une explication technique ou des recommandations pratiques, SafeTrain structure et enrichit le texte pour le rendre clair et utile.

Grâce au Completion, l'IA peut: générer des résumés fluides en évitant la simple juxtaposition de citations pour offrir un texte cohérent et lisible, répondre précisément à des questions techniques en s'appuyant sur des passages vérifiables récupérés lors du Retrieval, et adapter la forme en ajustant le ton ou le style rédactionnel (plus formel ou créatif) tout en régulant la température du modèle pour limiter les erreurs ou les hallucinations. À titre d'exemple, un ingénieur téléverse un manuel de maintenance. SafeTrain réalise d'abord un Retrieval pour identifier la section correspondant au composant défectueux, puis utilise le Completion pour produire une procédure de dépannage claire et adaptée, basée sur les informations exactes et vérifiables du document.

6.3 Cas d'usage

En combinant **Retrieval** et **Completion**, SafeTrain offre un service extrêmement efficace:

– **Extraction rapide de données pertinentes**: À partir de documents volumineux, SafeTrain identifie et extrait les informations cruciales grâce au Retrieval.

- **Génération de réponses finales concises et fiables**: Le Completion transforme ces informations en contenus adaptés au contexte de l'utilisateur.
- **Compréhension et manipulation de données complexes**: SafeTrain peut interpréter des tableaux relationnels, extraire des échantillons d'informations, et traduire des contenus tout en maintenant la cohérence contextuelle.
- **Polyvalence des tâches**: Tant que les demandes restent dans le cadre contextuel défini, SafeTrain peut accomplir une multitude de tâches, allant de la rédaction de résumés à l'explication de procédures techniques.

En somme, l'alliance du **Retrieval** et du **Completion** au sein du **RAG SafeTrain** offre une solution puissante pour extraire et synthétiser efficacement des informations, répondant ainsi aux besoins variés des utilisateurs avec précision et fiabilité.

En annexe, vous trouverez un exemple de tâche de **Retrieval** et de **Completion** illustrés dans les figures 9 et 10. Le dataset utilisé est un rapport technique sur l'implémentation d'un moteur de recherche multimodal basé sur CLIP.

7 Perspectives d'Évolution

7.1 Évaluation de la pipeline RAG sur diverses Benchmarkt

La version gratuite de Namespace impose une limitation technique à l'upload de 100 namespaces, ce qui correspond à un maximum de 50 documents au total. Cette contrainte s'explique par le fait qu'un document PDF génère automatiquement deux namespaces : un pour le chunking standard et un pour le late chunking. En raison de cette restriction, il n'a pas été possible d'évaluer le modèle RAG obtenu sur une diversité de benchmarks. Cette limitation freine l'exploration du potentiel du système sur d'autres tâches, comme l'évaluation de résumés ou la génération de réponses techniques, et restreint les possibilités d'analyse approfondie sur des corpus plus larges ou variés.

7.2 Gestion des Hallucinations

Le système RAG (Retrieval-Augmented Generation) se révèle performant, car il filtre efficacement la plupart des requêtes lorsque l'information précise n'est pas présente dans le document source. Toutefois, il peut survenir des cas d'**hallucination partielle** : des situations où le système extrait certains éléments pertinents et extrapole des informations qui ne sont pas explicitement formulées dans le texte.

7.2.1 Détail de l'exemple

Query : *Quel est le montant recommandé de pièces d'or avant d'acheter le premier objet ?*

Response : *Le montant recommandé de pièces d'or avant d'acheter le premier objet est de 475 pièces d'or. Cela vous permet d'acheter un ou plusieurs objets vous permettant de rester plus longtemps sur votre lane selon les besoins de votre personnage. Cette information provient du document "uplaod/1736guide.pdf".*

Source : Extrait intégral du document utilisé pour générer la réponse :

> « Vous commencez avec 475 pièces d'or et vous en gagnerez tout au long de la partie. Celles-ci vous permettent d'acheter des objets pour améliorer les performances de votre champion.
>
> Lors du début de partie, il est recommandé d'acheter un ou plusieurs objets vous permettant de rester plus longtemps sur votre lane, à savoir des objets augmentant votre base de régénération de vie ou de mana, selon les besoins de votre personnage. »

Dans ce cas précis, le système RAG a correctement identifié la donnée *475 pièces d'or* issue du texte source. Toutefois, il a interprété que ce montant constituait également la recommandation pour le premier achat, ce qui n'est pas explicitement indiqué dans le document. Cette interprétation erronée constitue une hallucination partielle : le système extrapole une recommandation à partir d'une information contextuelle (le montant initial de pièces d'or) sans que celle-ci ne soit directement formulée dans le document.

C'est pourquoi, il faut développer toute une architecture, permettant une meilleure compéhension contextuelle et d'améliorer la précision des réponses générées, afin de lutter contre ce phénomène.

7.3 Développement d'une version full local

Bien que SafeTrain exploite aujourd'hui des services cloud tels que PineconeDB, OpenAI et AWS S3, une version *full local* peut être extrèmement bénéfique. Dans certains secteurs (défense, énergie nucléaire, santé, etc.), la confidentialité des données est critique, et l'on préfère parfois garder la totalité du traitement en interne pour éviter des fuites de données avec des API externes. Un déploiement local nécessiterait :

- Une révision de l'architecture backend en remplacçant AWS S3 et NeonDB par un stockage interne. AWS S3 serait remplacer par MinIO afin de disposer d'un système de stockage objet local conforme aux standards S3. En parallèle, PostgreSQL serait déployé directement en local pour substituer NeonDB.
- L'installation d'une base vectorielle on-premise comme Weaviate pour remplacer PineconeDB.
- L'intégration d'un modèle de langage auto-hébergé, adapté à la puissance de calcul disponible (Ollama ou Mistral par exemple)

Le principal avantage serait de maintenir un contrôle total sur le flux de données, garantissant l'absence de fuite d'informations sensibles. La contrepartie en est un investissement en temps et en ressources pour dimensionner et gérer la maintenance technique.

7.4 Ciblage de problématiques complexes spécifiques

SafeTrain peut évoluer vers des tâches plus spécialisées que le simple QA (Questions & Answers). Un RAG orienté « compliance » pourrait, par exemple, vérifier la conformité de documents réglementaires en cherchant si des clauses obligatoires sont présentes ou absentes. Les pistes d'amélioration incluent :

- Des règles de recherche linguistique avancées (détection de paraphrases, d'articles de loi précis).
- Un mode d'évaluation interne comparant les réponses de l'IA à celles d'experts du domaine.
- Des workflows dédiés, où l'IA guiderait l'utilisateur à travers un processus étape par étape.

Par ailleurs, L'objectif initial de SafeTrain reposait sur l'intégration d'un mécanisme innovant de génération de QCM à partir du contenu de fichiers PDF. Ce système permettait à l'utilisateur de solliciter l'IA afin de formuler des questions ciblées sur des sections précises du document, en proposant pour chacune plusieurs options de réponse dont certaines pouvaient être correctes. Ainsi, les apprenants bénéficiaient d'une interface interactive leur permettant de répondre directement en cochant leurs choix, de recevoir une note accompagnée d'un feedback automatisé sur leurs erreurs, et même d'approfondir les notions mal assimilées grâce à des renvois précis vers les passages pertinents du PDF.

- Répondre directement depuis l'interface, en cochant les réponses jugées pertinentes.
- Recevoir une note et un feedback automatisé sur leurs erreurs.
- En analysant les résultats des QCM, l'IA peut identifier avec précision les domaines de difficulté de chaque apprenant et proposer des séries de questions sur mesure pour renforcer immédiatement leurs points faibles

Ce dispositif trouve tout particulièrement sa pertinence dans le cadre de la formation continue, notamment dans des secteurs où la mémorisation de protocoles rigoureux est cruciale, comme chez les étudiants ou les professionnels devant suivre des procédures strictes (par exemple, les opérateurs dans les centrales nucléaires ou les personnels évoluant dans le domaine médical). Dans ces domaines, les documents de référence sont souvent extrêmement volumineux et complexes. L'implémentation d'un RAG dans ce contexte prend donc tout son sens.

En outre, l'enjeu actuel dans le monde de l'IA ne se limite pas seulement au développement de modèles puissants, mais réside également dans la création d'une interface UX/UI véritablement intuitive. Bien que nous disposions d'outils de traitement du langage naturel capables de générer des réponses pertinentes à partir d'un simple prompt, il manque encore des initiatives pour transformer cette interaction en une expérience très intuitive. En effet, l'objectif serait d'intégrer la puissance de ces LLM dans un workflow, une interface utilisateur conçu sur mesure pour répondre à une problématique précise. C'est précisément cette raison qui nous a poussé à concevoir Safetrain comme une interface web. Malheureusement, le temps à manquer pour le développement de cette feature, mais il est possible de voir à quoi ça aurait pu ressembler dans la maquette Figma (voir ressource).

7.5 Évolution vers un système d'agents autonomes

À plus long terme, SafeTrain pourrait se muer en un système d'agents doués d'autonomie, capables :

- D'appeler diverses API pour exécuter des actions concrètes (recherche d'informations complémentaires, etc.).
- De réaliser des tâches de programmation ou de scripting.
- De planifier leurs propres sous-tâches si une requête complexe le justifie.

Cette vision rejoint l'idée d'« IA outillée » où le modèle de langage, plutôt que de se borner à répondre par du texte, peut agir sur l'environnement et déclencher des opérations au-delà du simple champ conversationnel.

8 Notebook Complémentaire : RAG local sur le dataset FQuAD

Afin de répondre à certaines problématiques non couvertes par l'implémentation initiale de SafeTrain, un notebook Jupyter dédié a été développé. Celui-ci met en œuvre des techniques avancées de RAG, tout en permettant de déployer le système *local*. Enfin, il permet de tester la robustesse du modèle sur un jeu de données conséquent (*benchmark*) ce qui n'était pas possible à faire sur Safetrain. Ce benchmarkt a été réalisé sur le **FQuAD** (French Question Answering Dataset), version française du célèbre SQuAD, afin de disposer d'un ensemble de questions-réponses standardisées et d'évaluer objectivement les performances du système. Enfin, ce benchmarkt a permis de comparer plusieurs méthodes de recherche (BM25, embeddings denses via *Sentence Transformers*, Hybrid Search), ainsi que différentes stratégies de découpage (*early chunking*, *late chunking*) pour optimiser la précision des réponses.

8.1 Description de la Pipeline

8.1.1 Prétraitement et Indexation :

- Découpage des contextes du dataset (FQuAD) en *chunks* de tailles adaptées.
- Création d'index pour la recherche lexicale (*BM25Okapi*) et la recherche vectorielle (*FAISS* + embeddings).

8.1.2 Méthodes de RAG Exploreés :

- **Early Chunking**
- **Late Chunking**
- **Hybrid Search**

8.1.3 Génération de Réponses :

- Utilisation du modèle **Mistral-7B-Instruct-v0.1** pour générer les réponses en français.
- Ajustement de paramètres tels que la *température* pour influer sur la créativité ou la précision des réponses.

8.2 Analyse des performances du RAG

8.2.1 Résultat Benchmarkt

- Métriques **Exact Match (EM)**, **F1 Score**, **BERTScore**, **ROUGE** et **BLEU**.
- Comparaison des différentes approches RAG (simple, early chunking, late chunking, hybrid) pour déterminer la plus adaptée selon le type de question.

Table 1: Résumé des performances pour différentes méthodes et températures (première étude).

Méthode	F1 (Moy.)	F1 (Max)	BERTScore (Moy.)	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
No Processing + Temp=0.1	0.36	0.44	0.70	0.34	0.26	0.34	0.03
No Processing + Temp=0.9	0.14	0.24	0.61	0.18	0.14	0.18	0.02
Early Chunking + Temp=0.1	0.36	0.44	0.69	0.34	0.27	0.34	0.03
Early Chunking + Temp=0.9	0.20	0.40	0.64	0.22	0.19	0.22	0.03
Hybrid Search + Temp=0.1	0.36	0.44	0.70	0.34	0.26	0.34	0.03
Hybrid Search + Temp=0.9	0.12	0.16	0.62	0.13	0.09	0.13	0.01
Late Chunking + Temp=0.1	0.36	0.44	0.69	0.34	0.26	0.34	0.03
Late Chunking + Temp=0.9	0.27	0.40	0.68	0.26	0.19	0.26	0.04

Table 2: Résumé des performances pour différentes méthodes et températures (deuxième étude).

Méthode	F1 (Moy.)	F1 (Max)	BERTScore (Moy.)	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
No Processing + Temp=0.1	0.36	0.44	0.70	0.34	0.26	0.34	0.03
No Processing + Temp=0.2	0.36	0.44	0.70	0.34	0.26	0.34	0.03
No Processing + Temp=0.3	0.35	0.44	0.70	0.34	0.26	0.34	0.03
No Processing + Temp=0.4	0.26	0.44	0.67	0.25	0.21	0.25	0.03
No Processing + Temp=0.5	0.31	0.43	0.69	0.30	0.27	0.30	0.03
No Processing + Temp=0.6	0.36	0.46	0.69	0.37	0.29	0.37	0.04
No Processing + Temp=0.7	0.36	0.60	0.69	0.38	0.35	0.38	0.05
No Processing + Temp=0.8	0.25	0.44	0.68	0.27	0.22	0.27	0.03
No Processing + Temp=0.9	0.14	0.24	0.61	0.18	0.14	0.18	0.02
No Processing + Temp=1.0	0.13	0.24	0.63	0.18	0.14	0.18	0.02
Early Chunking + Temp=0.1	0.36	0.44	0.69	0.34	0.27	0.34	0.03
Early Chunking + Temp=0.9	0.20	0.40	0.64	0.22	0.19	0.22	0.03
Hybrid Search + Temp=0.1	0.36	0.44	0.70	0.34	0.26	0.34	0.03
Hybrid Search + Temp=0.9	0.12	0.16	0.62	0.13	0.09	0.13	0.01
Late Chunking + Temp=0.1	0.36	0.44	0.69	0.34	0.26	0.34	0.03
Late Chunking + Temp=0.9	0.27	0.40	0.68	0.26	0.19	0.26	0.04

8.3 Analyse des résultats

8.3.1 Impact des stratégies de RAG avancés

Les différentes stratégies de RAG n'ont pas montré de différence significative sur la plupart des métriques. Les résultats suggèrent que :

- **No Processing (baseline)** offre déjà des performances correctes, avec un F1 moyen de 0.36 pour les températures basses (0.1–0.3).
- **Early Chunking** et **Late Chunking** produisent des scores similaires, mais le Late Chunking semble légèrement mieux résister à des températures élevées (>0.7).
- **Hybrid Search** (BM25 + embeddings denses) n'apporte pas de bénéfices marquants, voire affiche une performance inférieure à haute température (F1 Moy = 0.12 à Temp=0.9).

Ces faibles résultats peuvent s'expliquer par la structure relativement courte et uniforme des données FQuAD, qui ne bénéficient pas pleinement des techniques de segmentation.

8.3.2 Impact de la Température

La température s'est révélée être un facteur clé dans la performance du système :

- **Températures basses (0.1–0.3)** :
  - Offrent les résultats les plus stables et précis, avec un F1 moyen constant (0.36) et des BERTScores autour de 0.70.
  - Génèrent des réponses plus littérales, proches des gold answers, ce qui est bénéfique pour un benchmark de question-réponse comme FQuAD.
- **Températures modérées (0.4–0.7)** :

    - Montrent un potentiel d'amélioration pour des cas spécifiques. Par exemple, à Temp=0.7, la combinaison No Processing atteint un F1 Max de 0.60, le score maximal observé dans l'étude.
    - Cela suggère qu'une température modérée peut être bénéfique pour des questions nécessitant une légère reformulation.
- **Températures élevées (0.8–1.0)** :
    - Dégradent fortement les performances. Le modèle génère des réponses plus «créatives», souvent incohérentes ou éloignées des réponses attendues, ce qui se traduit par des chutes notables des scores F1, ROUGE et BLEU.

8.3.3 Analyse des Métriques

- **F1 Score** : Reste la mesure centrale pour évaluer la précision textuelle. Les températures basses/modérées (0.1–0.7) offrent des résultats cohérents autour de 0.36 (Moyen) et 0.60 (Max).
- **BERTScore** : Évalue la similarité sémantique et reste stable (0.68–0.70) pour les températures basses à modérées.
- **ROUGE (1, 2, L)** : Indique une couverture lexicale modeste (25 à 38
- **BLEU** : Les scores BLEU sont faibles (0.01 à 0.05), reflétant la difficulté pour un modèle génératif de produire des réponses parfaitement alignées sur les gold answers.

8.4 Synthèse générale

Le système RAG, malgré des résultats modestes en F1 (0.36 en moyenne), montre une stabilité notable pour des températures bien calibrées, tandis que le modèle Mistral-7B-Instruct parvient à générer des réponses textuellement et sémantiquement proches, comme l'indique son BERTScore élevé (0.70). Cependant, les méthodes de chunking et de recherche hybride n'ont pas apporté d'amélioration significative sur ce benchmark, probablement en raison de la nature des données FQuAD, caractérisées par des contextes courts et homogènes, et les températures élevées (>0.8) introduisent une variabilité excessive qui nuit à la précision des réponses. Par ailleurs, les scores F1 obtenus (0.36) restent bien inférieurs à ceux des modèles extractifs spécialisés sur FQuAD, qui dépassent souvent 0.80, ce qui est compréhensible puisque le système génère des réponses complètes, une tâche plus complexe que l'extraction stricte.

Ainsi, le système RAG local montre des performances globalement modestes sur le benchmark FQuAD, bien qu'elles soient encore éloignées de l'état de l'art pour des tâches de QA. La température s'est avérée être le facteur clé : des températures basses à modérées offrent des résultats bien meilleurs que des températures élevées, qui introduisent trop de variabilité. En outre, Les méthodes de chunking et de recherche n'ont

pas apporté de gains significatifs dans ce contexte spécifique, mais elles pourraient être plus pertinentes pour des corpus plus volumineux ou hétérogènes.

De plus, avec des ajustements (fine-tuning, prompting, combinaison extractif-génératif), ce système pourrait probablement se hisser à des niveaux de performance plus compétitifs, tout en exploitant la puissance générative du modèle Mistral-7B pour des scénarios nécessitant des réponses reformulées ou enrichies.

9 Conclusion

Le projet SafeTrain illustre avec succès l'intégration des avancées récentes en intelligence artificielle appliquées à la gestion et à l'exploration documentaire. En combinant la puissance des modèles de langage de grande taille avec des techniques avancées de Retrieval-Augmented Generation telles que le late chunking, le reranking et l'hybrid search, nous avons pu proposer une solution innovante permettant d'extraire et d'exploiter précisément l'information issue de documents PDF. Cette approche hybride renforce la pertinence et la fiabilité des réponses générées par le chatbot, tout en contournant les limitations inhérentes aux LLM classiques, notamment en matière de connaissances figées et de risques d'hallucination.

De plus, son interface web moderne abaisse la barrière d'entrée : un utilisateur lambda peut téléverser ses documents en quelques clics et exploiter directement la puissance d'un RAG. Ceci permet notamment d'ouvrir l'accès à ces technologies de pointe à un large public, sans exiger de compétences techniques approfondies. Par ailleurs, la personnalisation via les agents permet de modifier les paramètres intrinsèques de l'IA, de la pipeline RAG et d'intégrer de multiples stratégies de prompt engineering. Finalement, la scalabilité est également assurée par l'architecture cloud (Neon DB, S3, PineconeDB).

Néanmoins, SafeTrain présente certaines limites, comme celles associées aux capacités de stockage limitées des namespaces dans PineconeDB, qui empêchent l'évaluation de la pipeline sur un éventail de benchmarks. Bien qu'elle permette à n'importe qui d'accéder à ses services via une connexion internet, l'architecture actuelle de SafeTrain empêche également de proposer une solution de déploiement en local sans devoir changer la stack technique.

En outre, pour anticiper les futures évolutions de SafeTrain, un notebook Colab Jupyter a permis de répondre à ces problématiques en fournissant une solution via l'utilisation du modèle Mistral-7B-Instruct-v0.1 en local. Cette pipeline RAG basée sur Langchain a été évaluée sur le dataset français FQuAD à l'aide d'un benchmark constitué d'un ensemble de questions-réponses standardisées. Or, bien que les performances soient modestes par rapport aux modèles LLM à la pointe de la technologie, ce dernier nous a permis de comparer les performances de différentes approches de techniques avancées

de RAG ainsi que l'impact du paramètre température sur la pipeline finale. Les résultats ainsi obtenus démontrent que l'optimisation de la pipeline RAG reste une tâche complexe, nécessitant un grand effort d'optimisation.

Pour finir, les explorations menées dans le cadre de ce projet ouvrent la voie à de futures évolutions pour SafeTrain, telles que celles implémentées dans le notebook, qui renforcent sa capacité à éviter les états d'hallucination grâce à des architectures spécifiques, et d'autres qui n'ont pas été évoquées dans ce rapport, telles que l'adaptation de la solution à d'autres formats de documents, ou encore l'amélioration de sa résilience face à un grand nombre de requêtes et la sécurisation des données via des protocoles dédiés.

10 Annexes

10.1 Screenshots de l'interface utilisateur

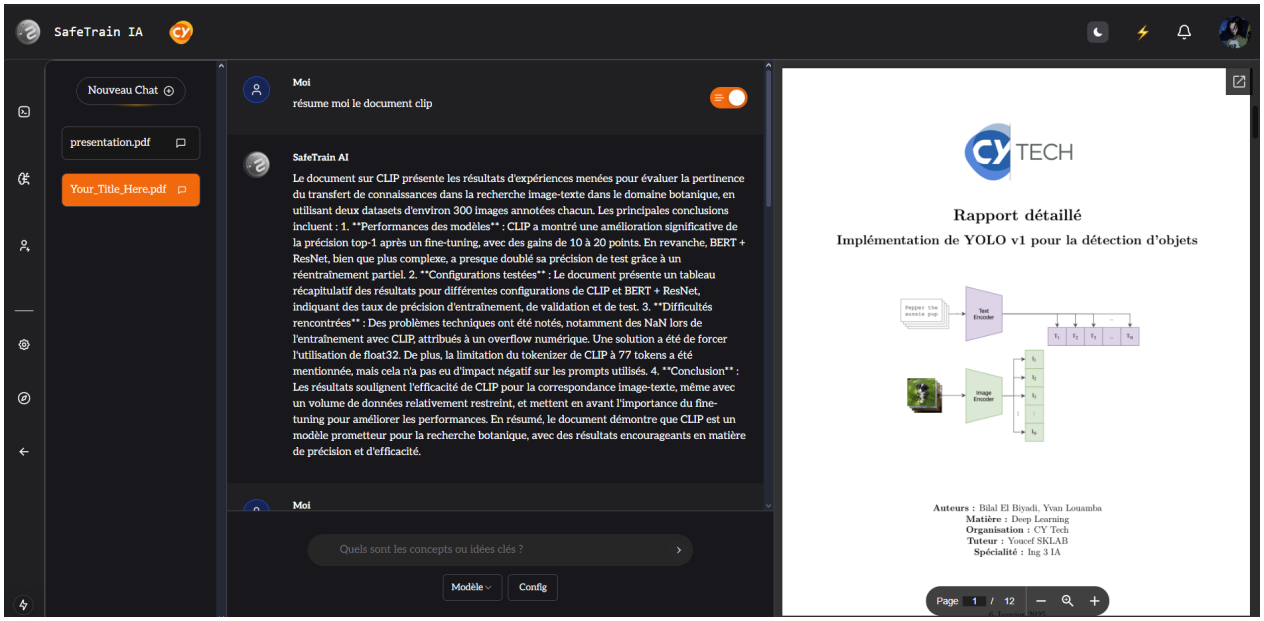


Figure 9: Page Chat

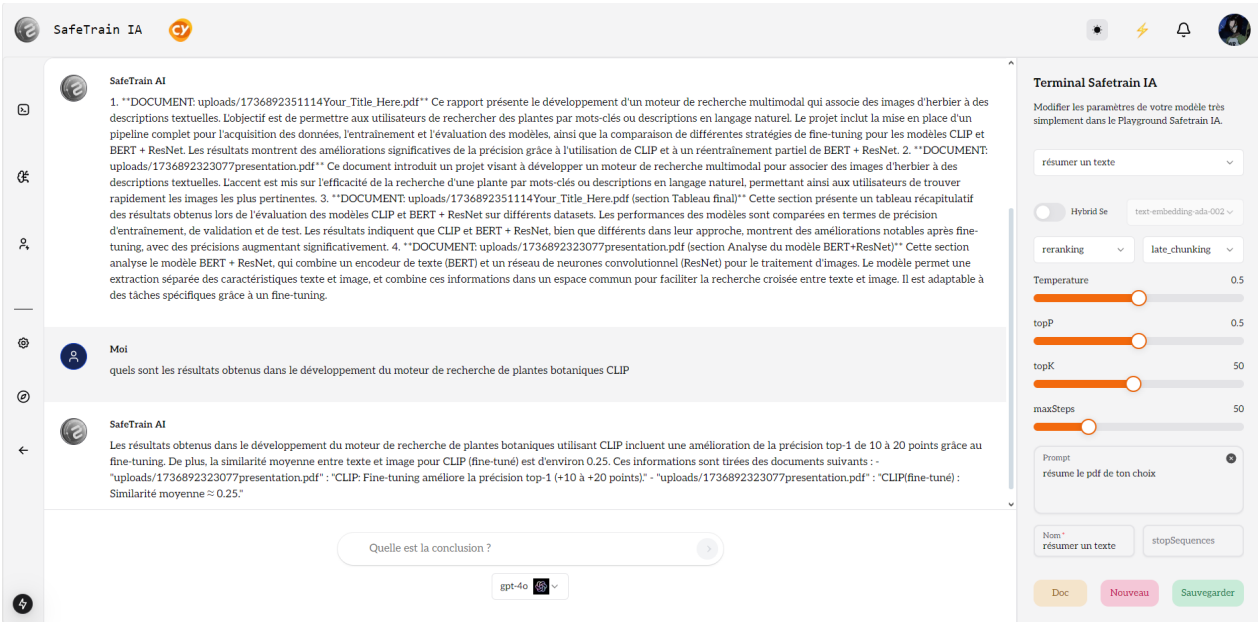


Figure 10: Page Playground

10.2 Liens utiles

- maquette Figma : https://www.figma.com/proto/7CS5whDGJ1piqZGqw00bCB/Safetrain?node-id=0-1fuid=1430500497485538796
- Les documentations officielles de Next.js, Typescript, Tailwind CSS, Neon DB, AWS S3, PineconeDB, Langchain, OpenAI, Grok et Gemini
- Le leaderboard Huggingface
- https://lilianweng.github.io/posts/2023-03-15-prompt-engineering/
- https://www.promptingguide.ai/fr
- https://www.pinecone.io/learn/series/rag/rerankers/
- https://www.pinecone.io/learn/series/rag/rerankers/
- https://github.com/jina-ai/late-chunking