Optimisation des processus de gestion des Connaissances par l'intelligence artificielle

Développement d'un agent conversationnel intelligent basé sur le RAG

Mouhamadou SECK

Sous la direction de Pedro Lucas Freitas, Project leader Software Solutions, Global R&D - Software Factory chez ArcelorMittal Méditerranée

Tuteur académique : Grégorie Maillard, Responsable du master 2 Data Science, Aix-Marseille Univérsité

UNIVERSITE AIX-MARSEILLE

Faculté des Sciences

Département des Mathématiques Appliquées et Statistiques (MAS)

Mémoire de master 2 Mathématiques Appliquées et Statistique

Parcours : Data Science

Année universitaire 2024-2025

Remerciements

Je souhaite exprimer ma profonde gratitude à l'ensemble des équipes d'ArcelorMittal – R&D Software Factory, dirigées par Monsieur Gérard Pasquet, pour leur accueil et leur disponibilité durant mon stage de fin d'études. Je remercie tout particulièrement Aymeric, Nicolas, Sarah, Noël, Mouhamed, Fabrice et l'ensemble des collaborateurs, dont le soutien et les échanges quotidiens ont rendu cette expérience aussi enrichissante sur le plan humain que sur le plan scientifique.

J'adresse également mes remerciements à Frank Dale et Stéphane Jimenez pour leur implication, leurs conseils pertinents et leur disponibilité, qui m'ont permis de mieux comprendre les besoins du projet et de progresser sur des aspects techniques essentiels tels que les stratégies de segmentation, les modèles d'embeddings et l'optimisation du pipeline.

Une mention spéciale revient à mon tuteur industriel, Pedro Lucas Freitas, Project Leader Software Solutions, pour son encadrement rigoureux, sa confiance et ses conseils avisés tout au long de ce travail.

Enfin, je remercie l'ensemble des enseignants du master Data Science, en particulier Gregory Maillard, Pierre Pudlo, Kai Schnaider, Frédéric Richard, JM. Freyermuth,Arthur Marmin, Hadrien Lorenz, Hichem Kadri, Emmanuel Caron, pour la qualité de leur enseignement et leur accompagnement bienveillant, qui ont grandement contribué à la réussite de ce parcours et à ma recherche de stage.

Je tiens à adresser une mention spéciale à l'ensemble de la promotion M2DS 2024/2025. Votre esprit d'entraide, votre bienveillance et votre solidarité ont contribué à rendre cette année non seulement formatrice, mais aussi profondément humaine. Comme l'écrit Albert Schweitzer : *« Le succès n'est pas la clé du bonheur. Le bonheur est la clé du succès. Si vous aimez ce que vous faites, vous réussirez »* [Schweitzer]. Cet état d'esprit collectif a été, pour moi, une véritable source d'inspiration et de motivation.

Résumé

Ce rapport rend compte du projet réalisé dans le cadre de mon stage de fin d'études de Master 2 en Data Science au sein du département R&D Software Factory d'ArcelorMittal Fos-sur-Mer. L'objectif était de concevoir et d'évaluer un agent conversationnel intelligent (Chatbot) reposant sur le paradigme du Retrieval-Augmented Generation (RAG), afin de renforcer la gestion des tickets, d'exploiter plus efficacement la documentation technique et de favoriser le partage de connaissances entre collaborateurs.

La démarche suivie s'est appuyée sur l'expérimentation de plusieurs méthodes de segmentation documentaire, d'indexation et d'ingénierie du contexte, associées à des outils open source tels que LangChain, Ollama, ChromaDB et Streamlit. Cette approche a conduit à la mise en place d'un prototype fonctionnel, capable d'interroger un corpus hétérogène et de fournir des réponses pertinentes, fiables, sourcés, contextualisées et traçables.

Les résultats montrent qu'un pipeline RAG, correctement paramétré, améliore significativement l'accès à l'information et le support interne, bien que des défis subsistent concernant la gestion de formats complexes, la robustesse face aux erreurs et la scalabilité du système.

Ce travail illustre le potentiel de la combinaison LLM + RAG en milieu industriel et ouvre des perspectives prometteuses pour un futur déploiement à plus grande échelle, notamment par l'optimisation des performances, l'intégration multi-formats et l'amélioration continue des mécanismes d'évaluation.

Mots-clés : Chatbot, RAG, NLP, LangChain, Ollama, ChromaDB, IA appliquée, industrie sidérurgique

Table of Contents

Sigles et abréviations utilisées

IA : Artificial Intelligence / Intelligence Artificielle

API : Application Programming Interface

ChromaDB : Base de données Chroma

GSMPM: Global Steelmaking Process Modeling

LLM: Large Language Model (grand modèle de langage)

NLP : Natural Language Processing (traitement automatique du langage naturel)

RAG : Retrieval-Augmented Generation

R&D : Recherche et Développement

UI : User Interface (interface utilisateur)

UX : User Experience (expérience utilisateur)

Glossaire

Terme	Définition
Agent conversationnel	Logiciel capable de dialoguer avec un utilisateur en langage naturel, également appelé chatbot.
Base vectorielle	Base de données spécialisée dans le stockage et la recherche de vecteurs, utilisée pour la recherche sémantique.
Chunking	Technique de segmentation des documents en morceaux (chunks) plus petits afin de faciliter l'indexation et la recherche.
Embedding (Représentation vectorielle)	Conversion d'un texte en vecteur numérique permettant de mesurer la similarité sémantique entre différents passages.
Hallucination	Réponse générée par un LLM mais non fidèle aux documents sources ou à la réalité, c'est-à-dire une information inventée.
LangChain	Framework open source permettant de créer et d'orchestrer des applications basées sur les LLM et le paradigme RAG.
Ollama	Outil open source permettant d'exécuter localement des modèles de langage sans dépendre de services cloud externes.
Prompt	Instruction ou requête donnée à un modèle de langage afin d'orienter la génération de texte.
Streamlit	Framework Python facilitant le développement rapide d'interfaces web interactives pour des projets de data science et d'intelligence artificielle.

Introduction

Dans un contexte industriel marqué par une transformation numérique accélérée, la gestion et la valorisation de la documentation technique constituent un défi central. ArcelorMittal, leader mondial de la sidérurgie, produit une vaste quantité de documents – guides d'utilisation, procédures internes, rapports techniques, présentations – essentiels au quotidien des équipes. Pourtant, leur diversité de formats et leur dispersion rendent l'accès à l'information complexe, ralentissant la résolution des problèmes et limitant le partage des connaissances.

Les systèmes de recherche traditionnels, basés sur des mots-clés, montrent rapidement leurs limites face à cette complexité : incapacité à restituer le contexte, difficulté à gérer synonymes et acronymes, et surtout manque de réponses exploitables directement par les ingénieurs. En parallèle, l'émergence des grands modèles de langage (LLM) a ouvert de nouvelles perspectives pour l'accès à l'information. Mais ces modèles souffrent de deux faiblesses majeures : la production d'hallucinations et une connaissance figée, limitée à leur phase d'entraînement.

Le paradigme du Retrieval-Augmented Generation (RAG) a été conçu pour dépasser ces limites. En combinant recherche documentaire et génération de texte, il permet de produire des réponses contextualisées, fiables et alignées sur les sources internes. L'architecture RAG repose sur trois étapes : récupération des passages pertinents, intégration de ces extraits dans le prompt, puis génération d'une réponse enrichie. Cette approche hybride valorise les bases documentaires des entreprises tout en renforçant la fiabilité des résultats.

C'est dans ce cadre que s'inscrit le projet mené au sein du département Digital Transformation – R&D Software Factory d'ArcelorMittal. L'objectif était de concevoir et de prototyper un agent conversationnel intelligent exploitant la documentation technique de l'équipe GSMPM (Global Steelmaking Process Modeling). Le projet a porté sur la préparation et l'unification du corpus documentaire, la mise en place d'un pipeline RAG complet (extraction, segmentation, vectorisation, indexation), le développement d'une interface interactive avec Streamlit et l'évaluation du prototype à l'aide de métriques quantitatives et qualitatives.

Au-delà de l'aspect technique, ce travail illustre le potentiel du couple LLM + RAG pour répondre à des enjeux industriels concrets : réduire le temps de recherche d'informations, fiabiliser les réponses et améliorer la gestion des tickets. Ce mémoire retrace cette démarche, du contexte théorique à la construction et à l'évaluation du prototype, en soulignant les apports, les limites et les perspectives d'évolution vers un déploiement opérationnel à grande échelle.

Chapitre 1

Présentation de l'entreprise et cadre du stage

Ce chapitre présente l'environnement institutionnel et technique du projet, en détaillant le rôle d'ArcelorMittal, du site de Fos-sur-Mer, de la R&D Software Factory et du programme GSMPM, afin de situer les enjeux qui ont motivé la mise en place d'un chatbot RAG.

1.1 Présentation de l'entreprise ArcelorMittal

ArcelorMittal est aujourd'hui le numéro un mondial de la sidérurgie et de l'exploitation minière. Présente dans plus de 60 pays et disposant de sites industriels dans une vingtaine d'entre eux, l'entreprise fournit des solutions en acier à une large gamme de secteurs : automobile, construction, énergie, emballage et industrie lourde.

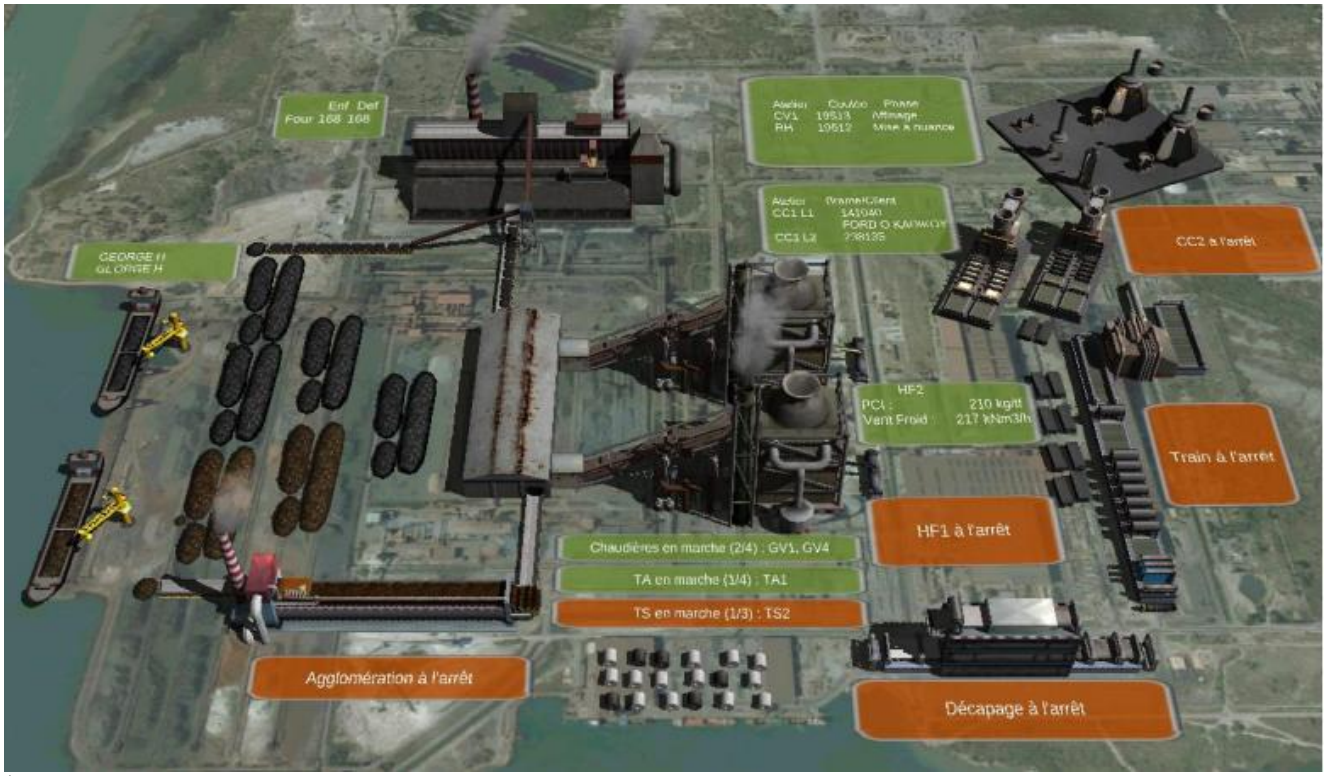
Sa réputation repose sur trois piliers : une capacité de production parmi les plus élevées au monde, une stratégie d'innovation continue, et un engagement fort en faveur du développement durable. Dans cette logique, ArcelorMittal investit massivement dans la décarbonation de ses procédés, en intégrant davantage de ferraille recyclée, en réduisant ses émissions de CO_2 et en adoptant les principes de l'économie circulaire.

1.1.1 Site Fos sur-Mer

Mon stage de fin d'études s'est déroulé sur le site ArcelorMittal de Fos-sur-Mer, l'une des plus grandes usines sidérurgiques françaises. Installé sur 1 600 hectares et employant environ 2 500 personnes, ce site stratégique produit chaque année plus de 4 millions de tonnes d'acier plat destinés à des secteurs clés comme l'automobile, l'emballage et le bâtiment.

Intégrant l'ensemble du cycle de production – de la fonte brute au laminage et aux traitements de surface – Fos-sur-Mer propose plus de 120 nuances d'acier adaptées à des besoins variés. Classé SEVESO seuil haut, il place la sécurité et la formation des équipes au cœur de son fonctionnement.

En parallèle, le site investit dans l'innovation et la transition énergétique. L'installation en 2024 d'un four poche de dernière génération a notamment permis d'augmenter la part de ferraille recyclée et de réduire l'empreinte carbone de près de 10 %, illustrant la volonté d'ArcelorMittal de concilier performance industrielle et durabilité.



[1]

Figure 1 : vue aérienne du site ArceloMittal de Fos-sur-Mer

[1] Synopsis. ArcelorMittal-Fos-Sur-Mer

Processus de fabrication de L'acier

ArcelorMittal Fos-sur-Mer, un site industriel majeur, fabrique des **nuances d'acier** de haute technologie en suivant un processus rigoureux. La production annuelle de plus de 4 millions de tonnes sert les marchés européens et méditerranéens dans des secteurs clés comme l'automobile et la construction. Le processus se divise en plusieurs étapes.

Du minerai de fer à la fonte

Tout commence par l'arrivée du minerai de fer par bateau. Ce minerai est transformé en un **aggloméré** sur une ligne de production dédiée. Simultanément, la cokerie transforme le charbon brut en **coke**, qui est presque du carbone pur. L'aggloméré et le coke sont ensuite chargés dans un **haut fourneau** pour la production de **fonte liquide**. Dans le fourneau, la combustion du coke sépare l'oxygène du fer, créant un alliage de fer et de carbone qui est ensuite acheminé vers l'aciérie.

Affinage et coulée continue

À l'**aciérie**, la fonte liquide est versée dans des **convertisseurs** où un lit de ferraille a déjà été placé. De l'oxygène pur est insufflé à 1600 °C pour brûler les impuretés comme le carbone. L'acier brut est ensuite transféré à la station d'affinage pour être enrichi en divers composants chimiques comme le nickel ou le chrome, permettant d'obtenir la **nuance d'acier** souhaitée.

Le métal en fusion est ensuite coulé en continu dans un moule appelé **lingotière**, où il commence à se solidifier au contact des parois refroidies. Le produit solidifié est découpé en longs morceaux appelés **brames**.

Laminage à chaud et décapage

Les brames sont chauffées et passent par le **train à bande**, un laminoir puissant qui réduit leur épaisseur et les transforme en bobines. Ce laminage à chaud améliore les propriétés mécaniques de l'acier et élimine les défauts. Certaines bobines subissent un **décapage**, un traitement chimique pour nettoyer leur surface des impuretés et de la rouille. Finalement, les bobines d'acier sont prêtes à être expédiées pour des usages variés.

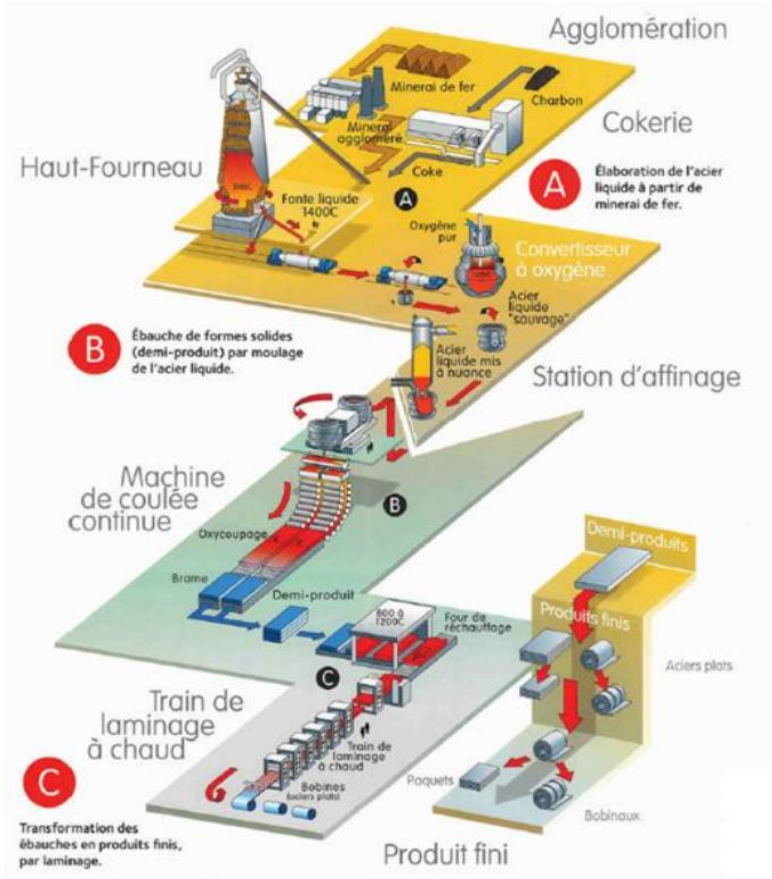


Figure 2 : Processus d'affinage et de coulé continue

1.1.2 R&D Software Factory

Au sein d'ArcelorMittal, un leader mondial de la sidérurgie, la **Recherche et Développement (R&D)** est essentielle à l'innovation continue. Ses équipes internationales se concentrent sur l'amélioration des procédés de production, le développement de nouveaux aciers, et la réduction de l'empreinte environnementale.

L'entité **Global Research and Development – Software Factory** incarne cette stratégie. Son rôle est de combler le fossé entre la recherche théorique et les applications industrielles. Elle prend les modèles sophistiqués développés par la R&D, comme les simulations thermiques ou les modèles cinétiques, et les transforme en solutions logicielles concrètes. Ces solutions sont ensuite déployées dans les usines du groupe, assurant que les innovations scientifiques se traduisent en gains d'efficacité, de qualité et de durabilité sur le terrain.

1.1.3 Équipe GSMPM

Mon stage a été réalisé au sein de l'équipe **GSMPM (Global Steelmaking Process Models)**, une composante clé de la R&D Software Factory. Lancé en 2009, le programme GSMPM visait à créer une plateforme générique et "faite maison" pour la modélisation des processus de fabrication de l'acier.

L'objectif principal du programme était de centraliser, gérer et standardiser l'utilisation de modèles complexes de production d'acier pour toutes les usines du groupe, et ce, à partir d'une interface unique. Le système est une **architecture client-serveur** qui gère l'ensemble du cycle de vie des modèles, de la collecte de données brutes sur les sites de production (via une communication bidirectionnelle avec les systèmes de l'usine) jusqu'au déploiement d'outils de surveillance et de réglage à distance pour les ingénieurs.

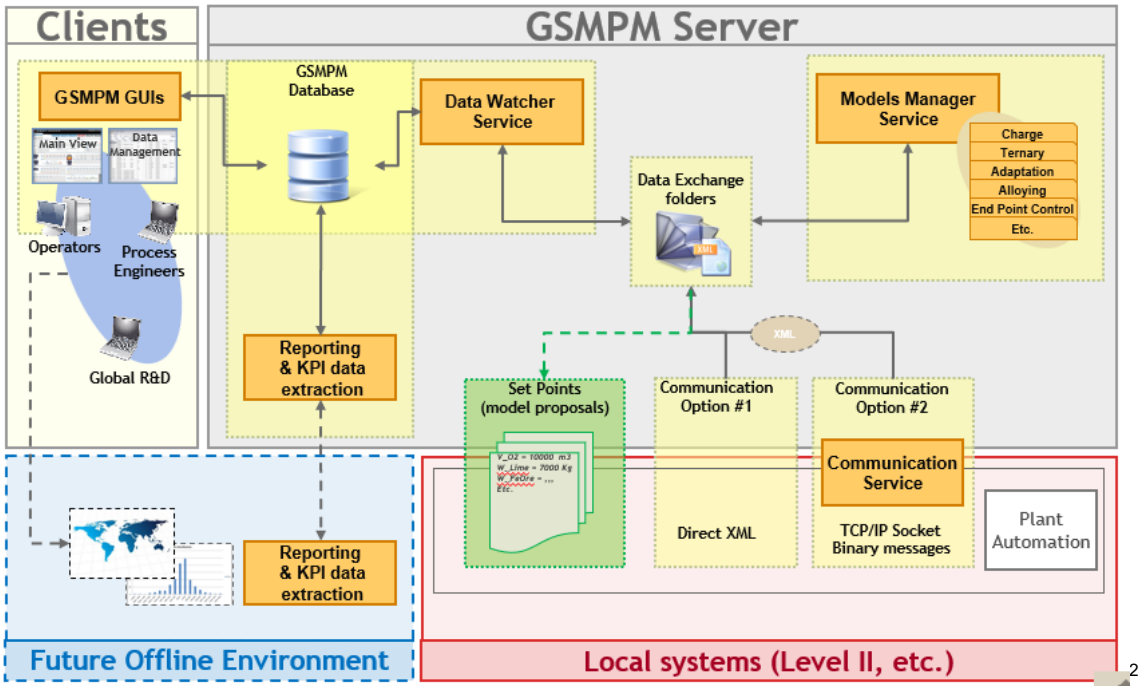


Figure 3 : GSMPM Standard Architecture

[2] Global Steel Making Process Models (GSMPM)
BOF charge calculation model
Getting started with the system
April 2014

Contexte du stage

Mon stage s'est déroulé au sein d'ArcelorMittal, un leader mondial de la sidérurgie. J'ai intégré le département R&D Software Factory, une entité clé qui transforme les modèles de recherche théorique en solutions logicielles opérationnelles pour l'ensemble du groupe. L'équipe GSMPM (Global Steelmaking Process Models), dans laquelle j'ai travaillé, est spécifiquement chargée de la modélisation des processus de fabrication de l'acier.

Dans ce cadre, la R&D fait face à un défi majeur : l'optimisation des processus de support et de partage des connaissances. Face au volume croissant des solutions déployées, l'équipe gère un grand nombre de requêtes et de documents techniques hétérogènes. Leur gestion manuelle est chronophage, ce qui freine la réactivité de l'équipe et peut ralentir les opérations de support.

Objectifs et missions

Pour répondre à cette problématique, l'objectif de mon stage a été de concevoir un agent conversationnel intelligent basé sur l'IA générative. Cet agent avait une double ambition : d'une part, aider les utilisateurs finaux à trouver des réponses aux questions les plus fréquentes pour réduire le volume de tickets de support. D'autre part, il devait faciliter le partage de la connaissance interne en permettant aux ingénieurs de l'équipe R&D d'accéder plus facilement à l'information technique.

Pour atteindre cet objectif, mes missions principales ont été :

- L'analyse du besoin et l'étude des composants existants.
- L'implémentation d'un agent conversationnel.
- L'intégration de cet outil dans les processus de support utilisateur et de partage de la connaissance interne.

Ce projet m'a permis de mettre en pratique des concepts avancés de l'IA pour créer une solution concrète qui optimise les processus informatiques du département R&D, en lien direct avec les enjeux de l'entreprise.

Chapitre 2

Contexte théorique et état de l'art

Ce chapitre présente les concepts fondamentaux nécessaires à la compréhension du projet. Il revient d'abord sur l'évolution des modèles de langage et du paradigme de l'intelligence artificielle générative. Il explore ensuite les mécanismes de représentation vectorielle et d'indexation sémantique, avant d'exposer le fonctionnement du **Retrieval-Augmented Generation (RAG)**. Enfin, il aborde les approches d'évaluation, indispensables pour mesurer la pertinence et la fiabilité d'un tel système dans un contexte industriel.

2.1. Intelligence artificielle générative et modèles de langage

2.1.1. Historique des IA-Gen

L'intelligence artificielle appliquée au traitement du langage naturel a connu une progression fulgurante, jalonnée par plusieurs ruptures technologiques. Les premières recherches, entre les années 1950 et 1990, reposaient sur des systèmes à base de règles, conçus pour des tâches très précises comme la traduction automatique. Bien que novatrices pour leur époque, ces approches restaient rigides et limitées, car elles ne pouvaient s'adapter à la diversité et à la complexité du langage humain. Dans les années 1990, une nouvelle génération de modèles a vu le jour avec l'introduction des méthodes statistiques. Ces dernières offraient une plus grande souplesse, mais leur efficacité restait contrainte par la puissance de calcul encore modeste.

Au début des années 2000, l'essor du *machine learning* et l'accès à des volumes de données sans précédent ont marqué un tournant décisif. Les modèles sont devenus plus robustes, capables d'exploiter des corrélations plus fines entre les mots et les phrases. En 2012, l'arrivée du *deep learning* a véritablement bouleversé le domaine, grâce aux réseaux neuronaux profonds capables d'apprendre des représentations complexes du langage. Quelques années plus tard, en 2018, Google introduisait BERT, un modèle basé sur l'architecture des *Transformers* qui a permis une compréhension contextuelle beaucoup plus fine. Puis, en 2020, OpenAI a franchi une étape supplémentaire avec GPT-3 et ses 175 milliards de paramètres, établissant une nouvelle référence en matière de génération de texte. Deux ans plus tard, ChatGPT a popularisé ces avancées auprès d'un large public, démocratisant ainsi l'accès à l'IA générative. Enfin, depuis 2023, les modèles *open source* tels que LLaMA, Falcon ou Mistral se sont multipliés et améliorés, intégrant des innovations comme le *Mixture of Experts* (MoE) ou le *Chain of Thought* (CoT). Ces évolutions récentes montrent une tendance à combiner puissance, modularité et spécialisation, ouvrant la voie à des architectures hybrides comme le RAG.

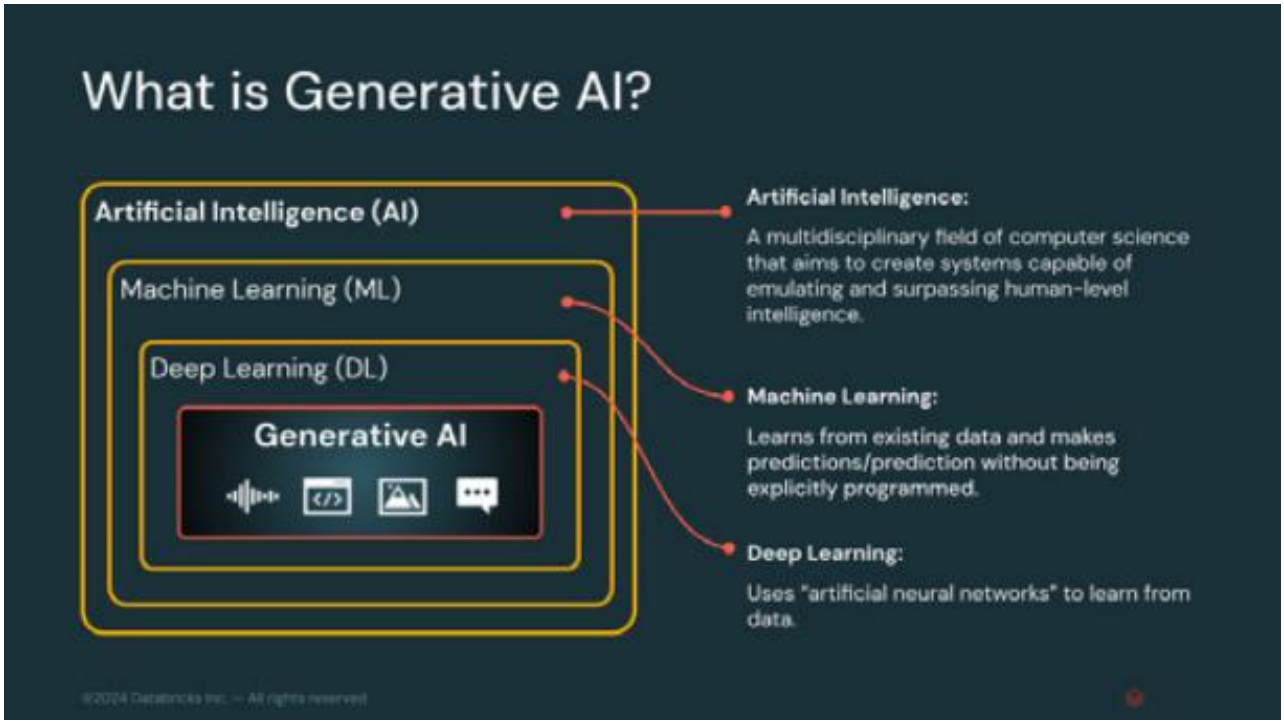


Figure 4 : Illustration de l'IA générative

[3]

[3] Databricks, Apache Spark™: A Compact Guide to Large Language Models

2.1.2 Les grands modèles de langage

Un *Large Language Model* (LLM) est un modèle d'intelligence artificielle entraîné sur de vastes corpus textuels. Son objectif principal est de prédire le mot le plus probable à la suite d'une séquence donnée, ce qui lui permet de générer du texte fluide et pertinent.

Image 5 : évolution LLM

2.1.2.1 Définition et fonctionnement

Le fonctionnent des LLMs reposent sur un principe de prédiction du token suivant, le token étant l'unité de base du texte (un mot, une syllabe ou un caractère). L'efficacité de ces modèles est directement liée à leur fenêtre de contexte, c'est-à-dire le nombre maximal de tokens qu'ils peuvent analyser simultanément pour générer une réponse.

Malgré leur puissance, les LLMs présentent des limitations inhérentes. Leur connaissance est figée au moment de leur entraînement, les rendant incapables de fournir des informations récentes ou spécifiques à un contexte non inclus dans leurs données d'origine.

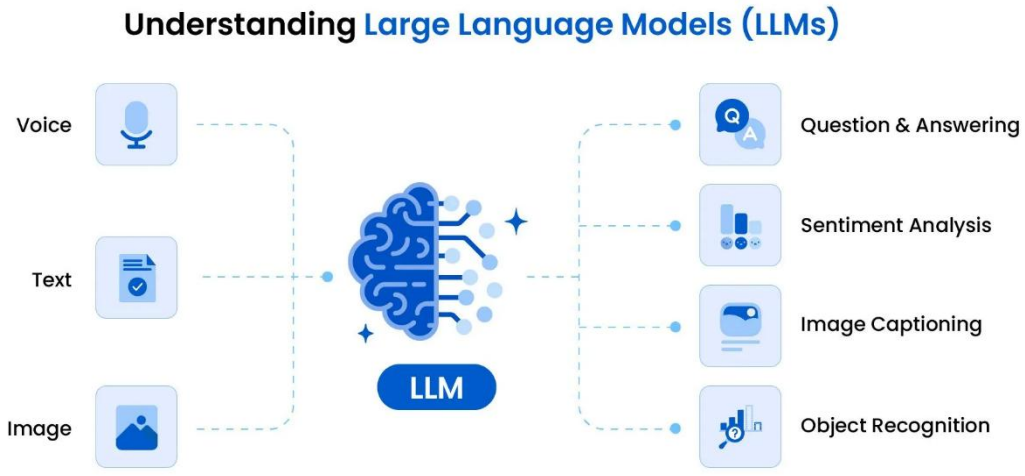


Figure 6 : Comprendre les LLMs

2.1.2.2 Modèle Transformer et le Mécanisme d'Attention

Les modèles LLMs utilisés dans ce projet reposent sur l'architecture Transformer, un modèle qui a révolutionné le traitement du langage naturel (NLP) depuis son introduction par Vaswani et al. en 2017. Le Transformer a permis de dépasser les limitations des architectures séquentielles précédentes, en introduisant un mécanisme capable de traiter l'intégralité d'une phrase de manière parallèle, offrant ainsi une compréhension contextuelle bien plus riche.

Mécanisme d'Attention

Le concept central du Transformer est le mécanisme d'attention multi-têtes, qui permet au modèle de pondérer l'importance de chaque mot d'une phrase en fonction des autres. Pour chaque mot, ou token, le modèle calcule un score d'attention indiquant l'importance relative des autres mots par rapport à celui en question

L'attention est calculée via la formule suivante :

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Où :

- **Q** (*queries*), **K** (*keys*) et **V** (*values*) sont des matrices obtenues à partir des *embeddings* d'entrée.
- **dk** est la dimension des vecteurs clés.
- La normalisation par **dk** est une technique qui stabilise les gradients pendant l'entraînement en évitant des valeurs trop grandes dans la fonction softmax.

L'**attention multi-têtes (Multi-Head Attention - MHA)**

L'attention multi-têtes étend ce mécanisme en effectuant des calculs en parallèle sur plusieurs "têtes" d'attention. Chaque tête peut ainsi se concentrer sur différents aspects ou relations entre les mots, ce qui enrichit la représentation sémantique. Les résultats de ces têtes sont ensuite combinés pour produire la sortie finale. Pour que le modèle comprenne l'ordre des mots, des embeddings de position sont ajoutés.

Architecture Globale du Transformer

Le modèle Transformer est généralement composé d'un encodeur et d'un décodeur. L'encodeur traite la séquence d'entrée pour produire une représentation sémantique riche, tandis que le décodeur utilise cette représentation pour générer la séquence de sortie. . Chaque composant est constitué de couches de self-attention et de réseaux de neurones *feed-forward*. Pour que le modèle comprenne l'ordre des mots, des embeddings de position sont ajoutés aux embeddings des mots.

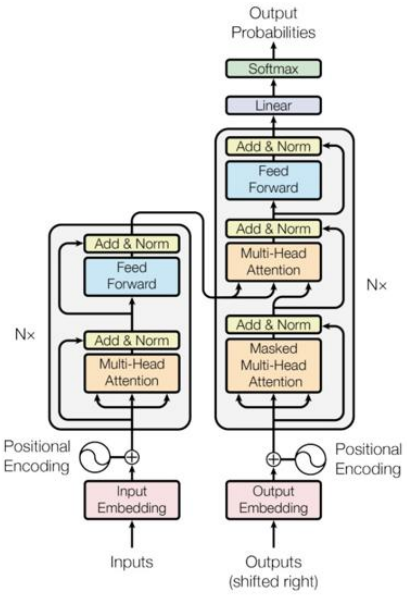


Figure 7 : Architechtute de Valwani et al.,(2017)

En résumé :

Le modèle Transformer est l'architecture fondatrice des LLM modernes. Il se distingue par l'utilisation de mécanismes d'attention, qui permettent au modèle de pondérer l'importance des mots dans une phrase, sans recourir à des architectures récurrentes ou convolutinelles. L'architecture se compose de deux modules principaux : **l'encodeur** et le **décodeur**, chacun étant un empilement de plusieurs couches identiques. L'encodeur traite la séquence d'entrée pour produire une représentation sémantique riche, tandis que le décodeur utilise cette représentation pour générer la séquence de sortie.

2.1.2.3 Paramètres de génération

La génération textuelle des LLMs est influencée par plusieurs paramètres, notamment la Température (qui contrôle le degré de créativité) et Top_k (qui restreint le nombre de tokens à considérer pour la prédiction). Ces paramètres sont essentiels pour équilibrer la créativité et la fidélité.

2.1.2.4 Modèles propriétaires vs modèles open source

Deux approches coexistent aujourd'hui dans l'usage des LLM. Les modèles propriétaires, proposés par des entreprises comme OpenAI, Anthropic ou Google, se distinguent par leurs performances souvent supérieures et leur simplicité d'utilisation via des API. Toutefois, cette dépendance à un fournisseur externe soulève plusieurs défis : coûts élevés, absence de transparence sur les données d'entraînement et risques liés à la confidentialité des informations transmises.

Face à cela, les modèles *open source*, développés et diffusés sur des plateformes comme Hugging Face, constituent une alternative intéressante. Ils peuvent être hébergés localement, garantissant une maîtrise totale des données et une meilleure confidentialité. De plus, ils offrent la possibilité d'être adaptés ou entraînés sur des corpus internes, ce qui les rend particulièrement pertinents pour des usages spécialisés.

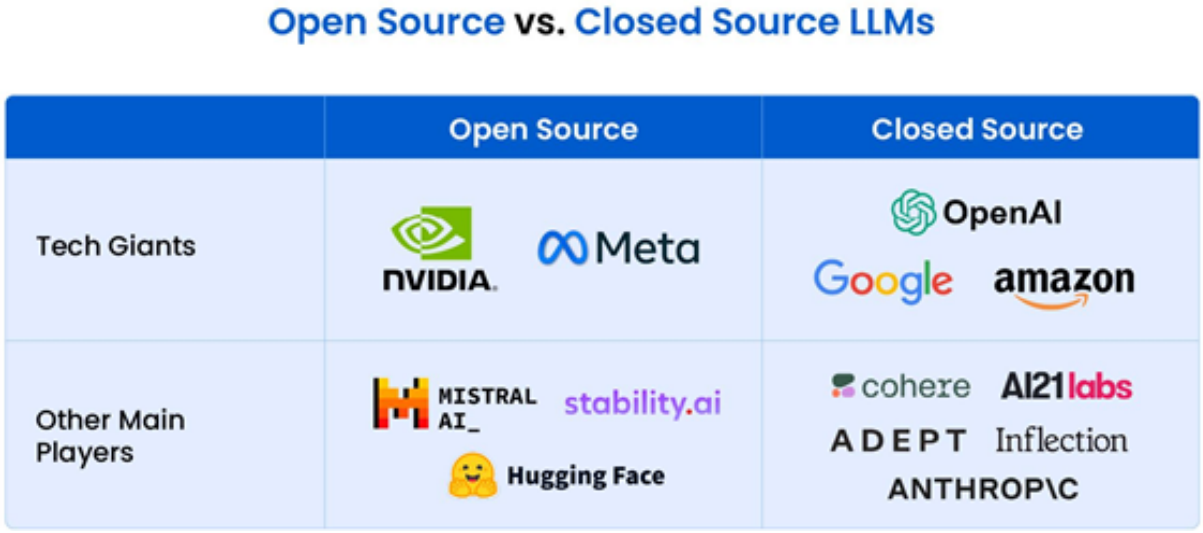


Figure 8 : Les modéles d'open sources vs les modéles payantes

2.3 Le paradigme RAG et motivation du projet

Pour pallier les limites des grands modèles de langage (LLM), notamment leur tendance à halluciner et leur connaissance figée, une nouvelle approche a été introduite par Lewis et al. en 2020 : le Retrieval-Augmented Generation (RAG). Ce paradigme combine la puissance de génération d'un LLM avec la précision de bases de connaissances externes.

Le RAG suit un processus en trois étapes distinctes pour produire des réponses fiables et factuelles :

1. **Récupération (*Retrieval*)** : Le système recherche et extrait les fragments d'informations les plus pertinents à partir d'une base de connaissances documentaire, grâce à des techniques de recherche sémantique.
2. **Augmentation (*Augmentation*)** : Les informations récupérées sont utilisées pour enrichir la requête initiale de l'utilisateur, créant ainsi un *prompt* plus détaillé et contextuel.
3. **Génération (*Generation*)** : Le LLM utilise ce *prompt* augmenté pour générer une réponse finale. Le modèle s'appuie alors sur les faits extraits des documents sources, réduisant ainsi le risque d'hallucination.

Cette approche est particulièrement pertinente dans le cadre de mon projet GSMPM. Le RAG permet de transformer une base documentaire hétérogène (composée de fichiers PDF, DOCX et PPTX) en un assistant conversationnel fiable. Il garantit que les réponses fournies aux ingénieurs sont non seulement pertinentes, mais aussi traçables, ce qui est crucial pour le support technique.

Définition : Hallucination

Hallucination : réponse générée par un LLM mais incorrecte ou non fondée sur les documents sources. Le RAG limite ce phénomène en injectant un contexte factuel dans le prompt.

Bien que le RAG soit une solution pragmatique, la compréhension de l'origine des hallucinations est un sujet de recherche en constante évolution. Le papier de recherche "Why Language Models Hallucinate" (Kalai et al., 2025) publié par OpenAI apporte un éclairage crucial sur cette problématique. Les auteurs y affirment que les hallucinations ne sont pas un phénomène mystérieux, mais peuvent être considérées comme des erreurs qui surviennent inévitablement dans les pipelines d'entraînement des modèles.

L'étude propose une nouvelle approche pour évaluer et pénaliser les hallucinations, suggérant que le problème ne peut être résolu uniquement par des solutions techniques. Il nécessite également une évaluation rigoureuse qui inclut le jugement humain pour déterminer la plausibilité d'une réponse. Le papier établit que les hallucinations peuvent être causées par des erreurs de classification lors du processus de génération et que la "plausibilité" est un facteur clé. Cette recherche renforce l'importance de concevoir des systèmes, comme le RAG, qui ne se contentent pas de générer du texte, mais qui garantissent également l'exactitude des informations en se basant sur des sources vérifiables.

Why Language Models Hallucinate

Adam Tauman Kalai* OpenAI
Ofir Nachum OpenAI
Santosh S. Vempala† Georgia Tech
Edwin Zhang OpenAI

September 4, 2025

Abstract

Like students facing hard exam questions, large language models sometimes guess when uncertain, producing plausible yet incorrect statements instead of admitting uncertainty. Such "hallucinations" persist even in state-of-the-art systems and undermine trust. We argue that language models hallucinate because the training and evaluation procedures reward guessing over acknowledging uncertainty, and we analyze the statistical causes of hallucinations in the modern training pipeline. Hallucinations need not be mysterious—they originate simply as errors in binary classification. If incorrect statements cannot be distinguished from facts, then hallucinations in pretrained language models will arise through natural statistical pressures. We then argue that hallucinations persist due to the way most evaluations are graded—language models are optimized to be good test-takers, and guessing when uncertain improves test performance. This "epidemic" of penalizing uncertain responses can only be addressed through a socio-technical mitigation: modifying the scoring of existing benchmarks that are misaligned but dominate leaderboards, rather than introducing additional hallucination evaluations. This change may steer the field toward more trustworthy AI systems.

1 Introduction

Language models are known to produce overconfident, plausible falsehoods, which diminish their utility. This error mode is known as "hallucination," though it differs fundamentally from the human perceptual experience. Despite significant progress, hallucinations continue to plague the field, and are still present in the latest models (OpenAI, 2025a). Consider the prompt:

What is Adam Tauman Kalai's birthday? If you know, just respond with DD-MM.

On three separate attempts, a state-of-the-art open-source language model[1] output three incorrect dates: "03-07", "15-06", and "01-01", even though a response was requested only if known. The correct date is in Autumn. Table 1 provides an example of more elaborate hallucinations.

Hallucinations are an important special case of *errors* produced by language models, which we analyze more generally using computational learning theory (e.g., Kearns and Vazirani, 1994). We consider general sets of *errors* $\mathcal{E}$, an arbitrary subset of plausible strings $\mathcal{X} = \mathcal{E} \cup \mathcal{V}$, with the other plausible strings $\mathcal{V}$ being called *valid*. We then analyze the statistical nature of these errors, and

*Email: adam@kal.ai
†Supported in part by NSF award CCF-2106444 and a Simons Investigator award. Email: vempala@gatech.edu
[1]The language model was DeepSeek-V3 (600 B parameters), accessed via the DeepSeek app on 11 May 2025.

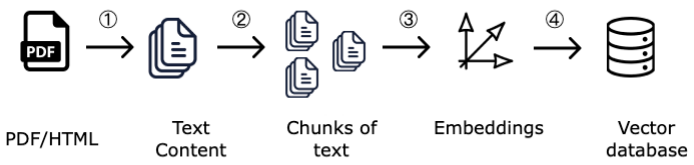
4

[4] https://cdn.openai.com/ why-language-models-hallucinate

Chapitre 3

Préparation et ingestion des documents

La préparation et l'ingestion des documents constituent la première étape opérationnelle du pipeline RAG. L'objectif de cette phase était de transformer le corpus brut des données en une base de connaissances exploitable par un agent conversationnel. Le travail a porté sur trois dimensions complémentaires : la collecte et la normalisation des données, la segmentation en unités textuelles cohérentes et enfin leur vectorisation suivie de l'indexation dans une base spécialisée.



3.1. Collecte et traitement des données

3.1.1. Contexte et objectifs

Le corpus documentaire mis à disposition couvrait une large variété de formats et de thématiques : rapports techniques en PDF, procédures internes en DOCX et présentations en PPTX. Ces fichiers, d'un grand volume, rassemblaient des manuels d'utilisation et des documents de support liés aux processus industriels, ect... Leur hétérogénéité représentait un défi majeur, car l'extraction de l'information devait à la fois centraliser les contenus et préserver les éléments critiques tels que les images, les tableaux, les schémas et les formules.

L'objectif principal était donc double : d'une part, convertir ces fichiers disparates en un corpus homogène et unifié ; d'autre part, conserver la richesse sémantique des documents, afin de ne pas perdre d'informations essentielles lors des traitements ultérieurs.

3.1.2 Défis liés à l'hétérogénéité des formats

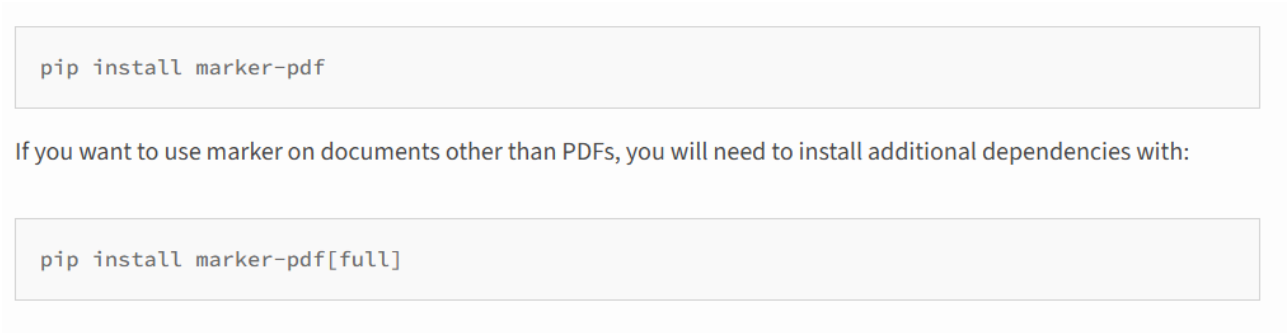
L'extraction de texte à partir de fichiers industriels ne se limite pas à une simple opération technique. Les fichiers PDF, par exemple, sont conçus pour l'affichage et non pour la structuration du contenu. Ils posent ainsi des problèmes fréquents de segmentation de mots, de gestion des colonnes ou d'interprétation des tableaux. Les fichiers Word sont plus faciles à manipuler mais contiennent parfois des schémas ou des zones non textuelles qui doivent être identifiées et traitées à part. Enfin, les présentations PowerPoint posent une difficulté particulière : le contenu est fragmenté en diapositives courtes et souvent condensé, ce qui nécessite une recomposition pour reconstituer une narration fluide.

3.1.3 Normalisation et choix du format cible

Pour simplifier les étapes ultérieures de traitement, il a été décidé de convertir l'ensemble du corpus en un format unique. Le choix s'est porté sur le Markdown (.md), car ce format combine légèreté, universalité et flexibilité. Il permet de conserver une structure hiérarchique claire (titres, sous-titres, paragraphes), de représenter les listes ou les équations et surtout de baliser explicitement les éléments non textuels comme les images ou les tableaux. Contrairement à un format brut comme le TXT, il rend possible une exploitation ultérieure beaucoup plus riche dans le cadre d'un pipeline RAG.

3.1.4 Conversion avec Marker

L'outil retenu pour cette conversion a été Marker, une librairie open source conçue pour extraire du contenu de documents complexes et les transformer en Markdown. Ce choix a été motivé par sa compatibilité avec différents formats (PDF, DOCX, PPTX, etc..) et sa capacité à préserver la structure des documents. Par exemple, les titres sont automatiquement reconnus et hiérarchisés, les tableaux sont transcrits dans une syntaxe lisible et les équations sont balisées pour rester exploitables et enfin les images sont conservé à l'aide d'un OCR nommé Soraya.



[5]

Figures 9 : Installation de marker

Un cas concret illustre bien cette valeur ajoutée : dans un rapport technique décrivant les consignes de sécurité, les tableaux PDF ont été extraits sans perte de structure. Là où une extraction classique aurait produit une simple suite de

[5] https://pypi.org/project/marker-pdf

mots sans organisation, Marker a permis de restituer les colonnes et les lignes en Markdown, garantissant ainsi la lisibilité et la réutilisabilité du contenu.

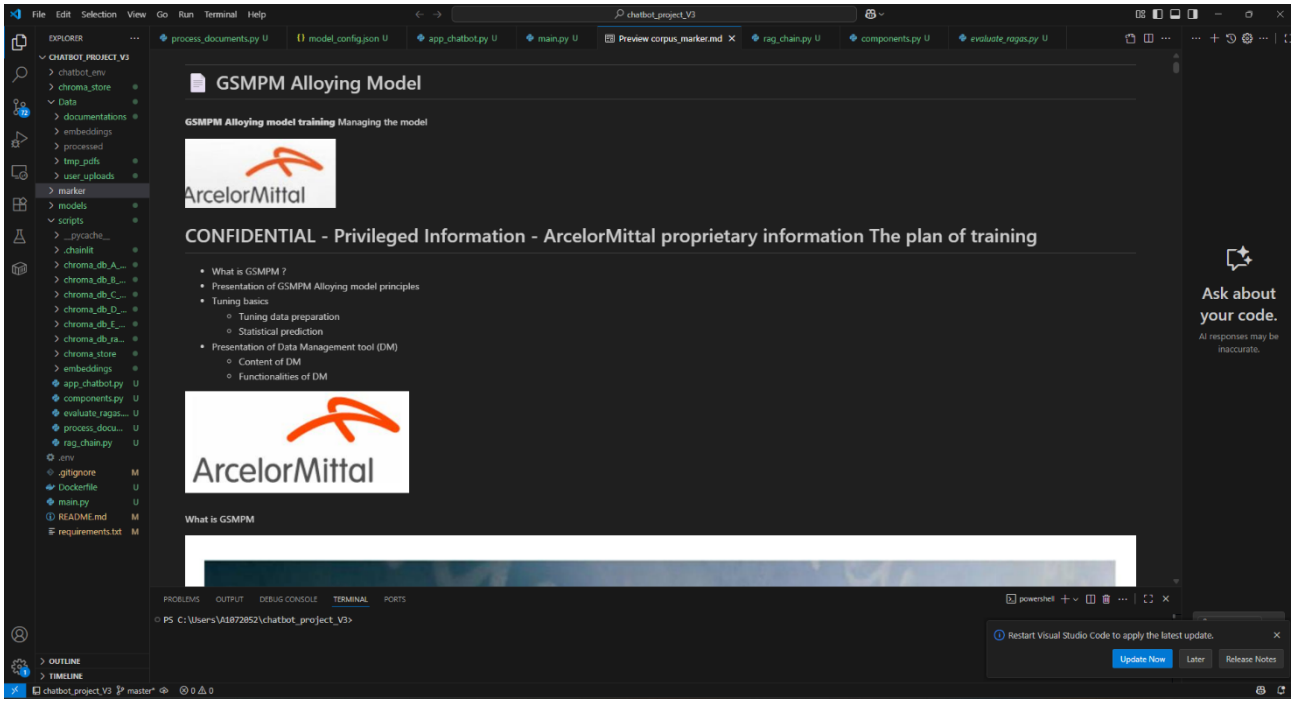


Figure 10 : overview guide-file-serre-file.md

3.1.5 Résultats de la phase de préparation

À la suite de cette phase préparatoire, le corpus initial, composé de documents hétérogènes, a été uniformisé pour aboutir à une collection cohérente de fichiers Markdown. Cette transformation a considérablement simplifié les étapes ultérieures du projet, en garantissant que le contenu crucial de chaque document (titres, images, tableaux) soit conservé et rendu pleinement exploitable par les modèles de langage.

3.2 La segmentation des documents (Chunking)

Une fois le corpus de documents uniformisé, l'étape suivante, cruciale pour la performance de notre système, a consisté à segmenter le contenu en unités de texte plus petites, appelées chunks. Cette démarche est une nécessité technique car les grands modèles de langage (LLM) comme Mistral ou GPT ont une capacité de traitement simultanée de l'information limitée par leur fenêtre de contexte. De nombreux documents du corpus GSMPM, tels que les manuels techniques, dépassaient largement cette limite, rendant leur traitement direct impossible.

L'objectif de cette segmentation était double : il fallait à la fois garantir une granularité adaptée au LLM tout en préservant la cohérence sémantique du contenu. Pour atteindre ce compromis, j'ai optimisé deux paramètres clés:

- Le chunk_size : la taille maximale d'un fragment, fixée à environ 205 tokens.
- Le chunk_overlap : le chevauchement entre les fragments, établi à 40 tokens.

Cette optimisation, obtenue par expérimentation, a permis d'assurer que des informations essentielles ne soient pas perdues aux frontières des fragments.

```
[54]
    for i, chunks_recursive in enumerate(chunks_recursive[:3]):
        print(f"\n--- Chunk {i+1} --- ({len(chunks_recursive)} caractères)\n{chunks_recursive[:1000]}...\n")
```

```
...
--- Chunk 1 --- (161 caractères)
#### **Guide File :** *GUIDE LES PERSONNES VERS LES ISSUES de SECOURS* **GIP 31.31**

### **Prise de fonction**

- Savoir où est le détecteur de gaz (salle café)...


--- Chunk 2 --- (140 caractères)
- Avoir suivi la formation « Risque Gaz , utilisation du détecteur gaz » et la formation « Guide-file, Serre-file »

### **Alerte incendie**...


--- Chunk 3 --- (128 caractères)
- Revêt un gilet jaune à sa disposition (salle café)
- Parcourt sa zone , demande et guide l'évacuation IMMEDIATE des personnels...
```

Figure 11 : visualisations des chunks avec la strattégie de recursive text splitter

3.2.1 Stratégies de segmentation explorées

Pour choisir la meilleure méthode de découpage, j'ai évalué plusieurs approches, chacune avec ses avantages et ses inconvénients :

- **Recursive Character Text Splitter** : Cette méthode découpe le texte en se basant sur une hiérarchie de séparateurs (paragraphes, phrases, etc.). Elle est robuste et respecte la structure logique, mais peut parfois générer des fragments peu naturels.
- **Token-based Splitter** : Cette approche est plus brute et segmente directement en fonction du nombre de tokens. Bien qu'efficace pour respecter la fenêtre de contexte des LLM, elle peut couper une phrase au milieu, ce qui dégrade la cohérence.
- **Semantic Chunking** : Cette méthode utilise un modèle de vectorisation pour regrouper les phrases par similarité sémantique. Elle est très performante pour la cohérence thématique, mais se révèle coûteuse en ressources de calcul.
- **Title-based Chunking** : Cette stratégie exploite la structure des documents en utilisant les titres et les sous-titres comme points de découpage. Elle est très pertinente pour les documents bien structurés, mais inefficace pour les fichiers mal formatés.
- **Agentic Chunking** : Une approche plus récente où un LLM décide de manière dynamique des frontières des fragments. Bien que très flexible, cette méthode est encore considérée comme instable.

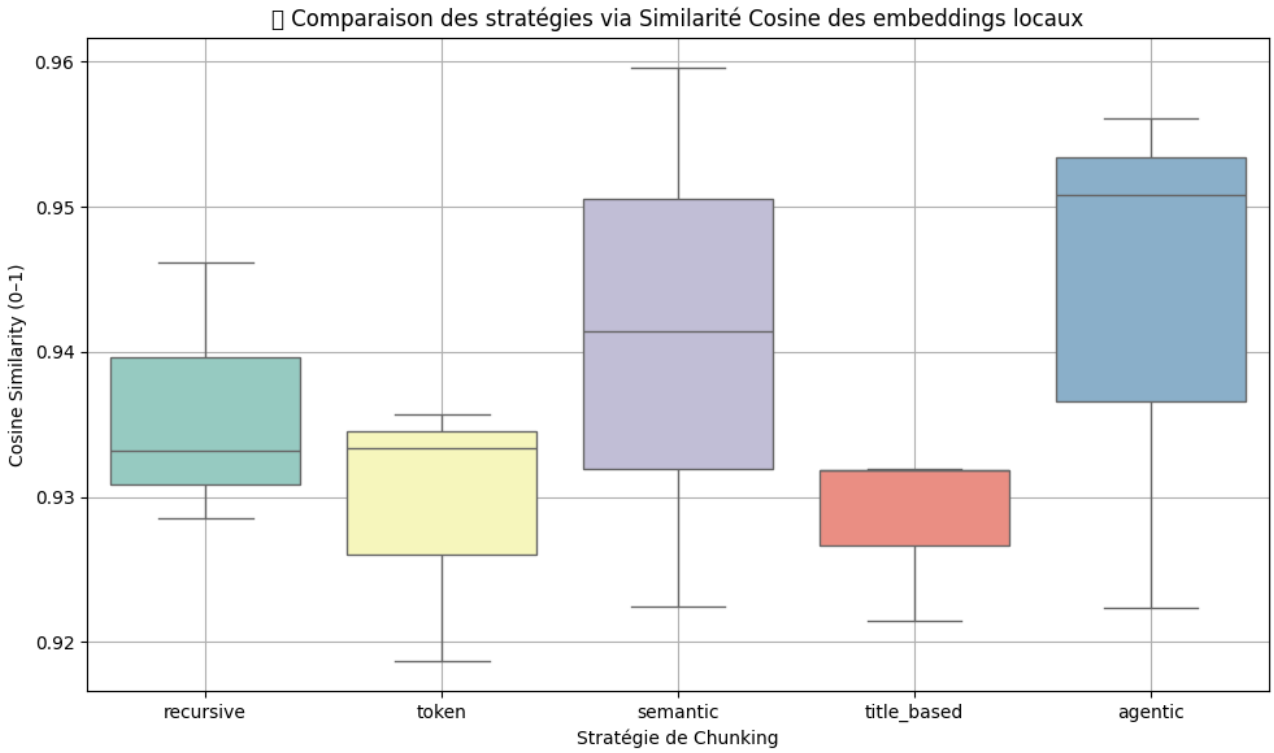


Figure 12 : Boxplot des différents stratégies de chunking testés

Après cette analyse comparative, j'ai choisi d'adopter le Recursive Character Text Splitter comme méthode principale. Il représentait le meilleur compromis entre efficacité et simplicité de mise en œuvre, s'adaptant bien à l'hétérogénéité de notre corpus.

Pour pallier les limites de cette approche, j'ai également exploré l'intégration d'un mécanisme de **reranking** des fragments récupérés. Cette technique, qui vise à classer les résultats de recherche par pertinence avant de les soumettre au LLM, est une piste d'amélioration prometteuse.

3.3 La vectorisation et l'indexation

Après avoir segmenté le corpus documentaire, l'étape suivante consistait à rendre ce texte exploitable par un modèle de recherche sémantique. J'ai alors entrepris la vectorisation du contenu, un processus clé dans le développement d'un système RAG.

3.3.1. Les embeddings : du texte au vecteur

La vectorisation, ou *embedding*, a pour objectif de transformer chaque fragment de texte (*chunk*) en un vecteur numérique dans un espace multidimensionnel. Ce processus est essentiel car il permet d'encoder le sens des mots. Dans cet espace, la proximité géométrique des vecteurs reflète la similarité sémantique des textes. Par exemple, même si les phrases « évacuer le personnel vers la sortie sud » et « guider les employés vers le point de rassemblement sud » ne partagent pas le même vocabulaire, leurs vecteurs se retrouveront très proches, car elles décrivent des actions similaires.

Pour ce projet, le choix du modèle d'embedding a été stratégique. J'ai sélectionné le modèle **nomic-embed-text**, exécuté localement via Ollama. Cette décision a été motivée par plusieurs raisons : la qualité sémantique du modèle, son exécution en local qui garantit la confidentialité des données d'ArcelorMittal, et sa compatibilité native avec les outils de notre pipeline. Chaque *chunk* a ainsi été converti en un vecteur de 768 dimensions, prêt pour l'étape suivante.

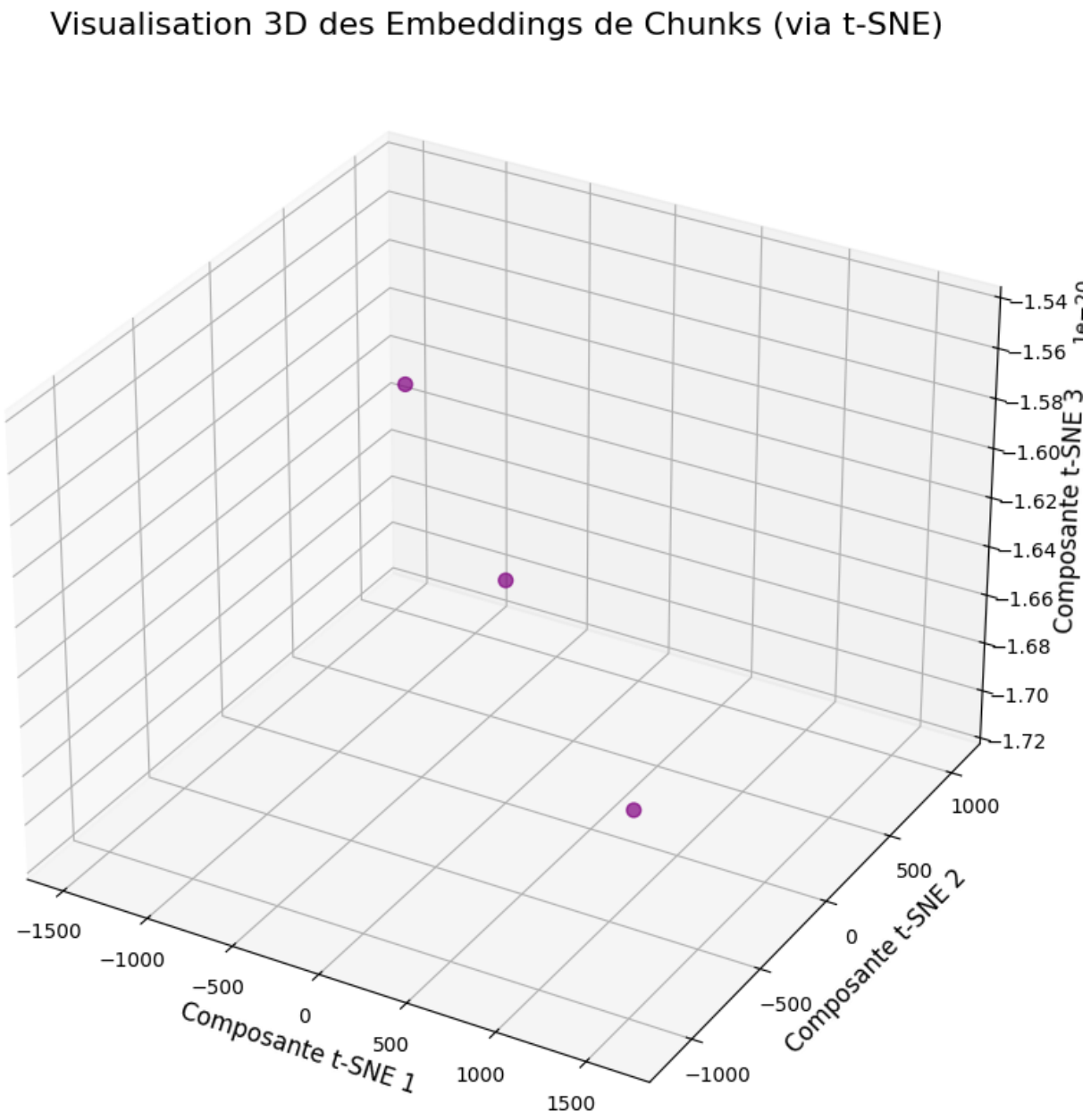


Figure 13 : visualistion 3D embeddings des chunks

3.3.2. L'indexation vectorielle

Après la vectorisation des fragments de texte, ceux-ci doivent être organisés dans une structure qui permette de les retrouver efficacement. C'est le rôle de l'indexation vectorielle. Contrairement à une recherche classique par mots-clés, elle s'appuie sur la similarité sémantique : chaque fragment est représenté par un vecteur numérique, et la recherche consiste à trouver les vecteurs les plus proches de celui de la requête de l'utilisateur.

La proximité entre deux vecteurs peut être mesurée de différentes manières, les plus utilisées étant :

- **la similarité cosinus**, qui compare l'orientation des vecteurs (adaptée aux textes),
- **la distance euclidienne**, qui mesure leur écart géométrique,
- **le produit scalaire**, qui évalue leur corrélation.

Comme ces vecteurs ont souvent plusieurs centaines de dimensions, une recherche exhaustive serait trop coûteuse. Pour accélérer le processus, des algorithmes spécialisés sont utilisés:

- **FAISS**, développé par Meta, qui gère efficacement des millions de vecteurs avec GPU,
- **HNSW**, basé sur un graphe de voisinage hiérarchique, connu pour sa rapidité et sa précision,
- **IVF et PQ**, qui réduisent l'espace de recherche en regroupant ou en compressant les vecteurs.

Dans le cadre de ce projet, l'indexation a été réalisée avec **ChromaDB**, une base vectorielle adaptée à des corpus de taille moyenne et facilement intégrable dans un pipeline RAG. Lors de chaque requête, le vecteur de la question est comparé à ceux des fragments indexés. Les *k* fragments les plus proches (avec $k = 5$ dans nos tests) sont alors sélectionnés et transmis au modèle génératif. Ce paramétrage permet d'assurer un bon équilibre entre rappel (ne pas manquer d'informations utiles) et précision (éviter d'ajouter trop de bruit).

3.3.3. Quelques exemple de bases vectorielles

Définition : Base vectorielle

Base vectorielle : base de données conçue pour stocker et interroger des vecteurs numériques. Elle est utilisée dans le cadre du RAG pour retrouver rapidement les passages les plus pertinents d'un corpus documentaire.

Plusieurs solutions open source et propriétaires existent pour stocker et interroger efficacement les embeddings.

— **FAISS (Facebook AI Similarity Search)**

- ✓ Développée par Meta.
- ✓ Points forts : rapidité, efficacité en mémoire, adaptée aux très grands volumes de données.
- ✓ Limite : interface bas-niveau, nécessite plus d'expertise technique pour être intégrée dans un projet de prototypage rapide.

— **Pinecone**

- ✓ Solution SaaS (hébergée dans le cloud).
- ✓ Points forts : gestion automatique de la scalabilité, haute disponibilité.
- ✓ Limite : dépendance au cloud, ce qui pose des questions de confidentialité dans un contexte industriel sensible.

— **Weaviate**

- ✓ Base vectorielle open source, extensible via des modules (par ex. recherche hybride texte + vecteur).
- ✓ Points forts : riche en fonctionnalités avancées, supporte la recherche multimodale.
- ✓ Limite : plus lourde à mettre en place dans un environnement local.

— **ChromaDB**

- ✓ Base vectorielle open source, légère et facile à intégrer.
- ✓ Points forts : simplicité, rapidité de déploiement, intégration native avec LangChain et Ollama.
- ✓ Limite : moins adaptée que FAISS ou Pinecone pour des corpus massifs (> 100 millions de vecteurs)

3.3.4. ChromaDB

Dans le cadre de ce stage, le choix s'est porté sur ChromaDB, pour plusieurs raisons :

1. **Simplicité et rapidité de mise en œuvre** : ChromaDB s'installe et s'utilise en quelques lignes de code, ce qui a permis un prototypage rapide.
2. **Compatibilité avec LangChain** : la base s'intègre directement dans le pipeline RAG développé, sans nécessiter d'adaptations lourdes.
3. **Exécution locale** : contrairement à Pinecone, ChromaDB peut être déployée entièrement en local, ce qui garantit la confidentialité des documents internes.
4. **Performance suffisante** : pour un corpus de taille moyenne, ChromaDB offre une rapidité de recherche adéquate.

Concrètement, lors d'une requête utilisateur, le pipeline RAG interroge ChromaDB avec un paramètre top_k = 5, ce qui permet de récupérer les 5 chunks les plus proches sémantiquement. Ces fragments sont ensuite fournis au LLM, qui les utilise pour générer une réponse contextualisée et fiable.

Chapitre 4

Construction du pipeline RAG

La construction d'un pipeline Retrieval-Augmented Generation (RAG) représente le cœur de ce projet. Elle consiste à articuler différentes briques technologiques pour transformer une requête utilisateur en une réponse fiable, claire et contextualisée à partir des documents GSMPM. L'architecture développée au cours de ce stage suit une logique modulaire, ce qui permet d'expérimenter, d'ajuster et d'optimiser chaque composant indépendamment.

Le pipeline mis en œuvre comprend plusieurs étapes successives : la récupération des passages pertinents dans la base vectorielle, leur réorganisation (reranking), l'intégration des métadonnées, la construction du prompt, la génération de la réponse par le modèle de langage et enfin l'interaction via une interface utilisateur. Cette organisation vise à garantir à la fois robustesse (limitation des hallucinations), précision (sélection des informations les plus pertinentes) et utilisabilité (ergonomie adaptée aux besoins métiers).

4.1 Orchestration avec LangChain

L'orchestration des différentes étapes du pipeline a été réalisée avec LangChain, un framework spécifiquement conçu pour construire des applications basées sur des modèles de langage. Son rôle a été d'assurer une articulation fluide entre la recherche des documents, la préparation du prompt et la génération de la réponse.

Le processus se déroule en trois phases principales :

- Retrieval : la requête utilisateur est convertie en vecteur, puis comparée aux vecteurs présents dans ChromaDB afin de récupérer les fragments les plus proches sémantiquement.
- Augmentation : les passages retrouvés sont insérés dans le prompt, en y intégrant leurs métadonnées pour renforcer la traçabilité.
- Generation : le modèle de langage utilise ce prompt enrichi pour produire une réponse claire et factuelle, ancrée dans la documentation.

L'utilisation de LangChain a offert plusieurs avantages : un niveau d'abstraction élevé, qui évite de gérer manuellement les appels aux modèles ; la possibilité de tracer chaque étape pour faciliter les ajustements ; et la souplesse nécessaire pour remplacer ou tester différents composants (embeddings, vectorstore, modèle génératif).

Dans ce projet, l'orchestration a donc permis d'expérimenter diverses configurations tout en conservant une architecture claire et évolutive.

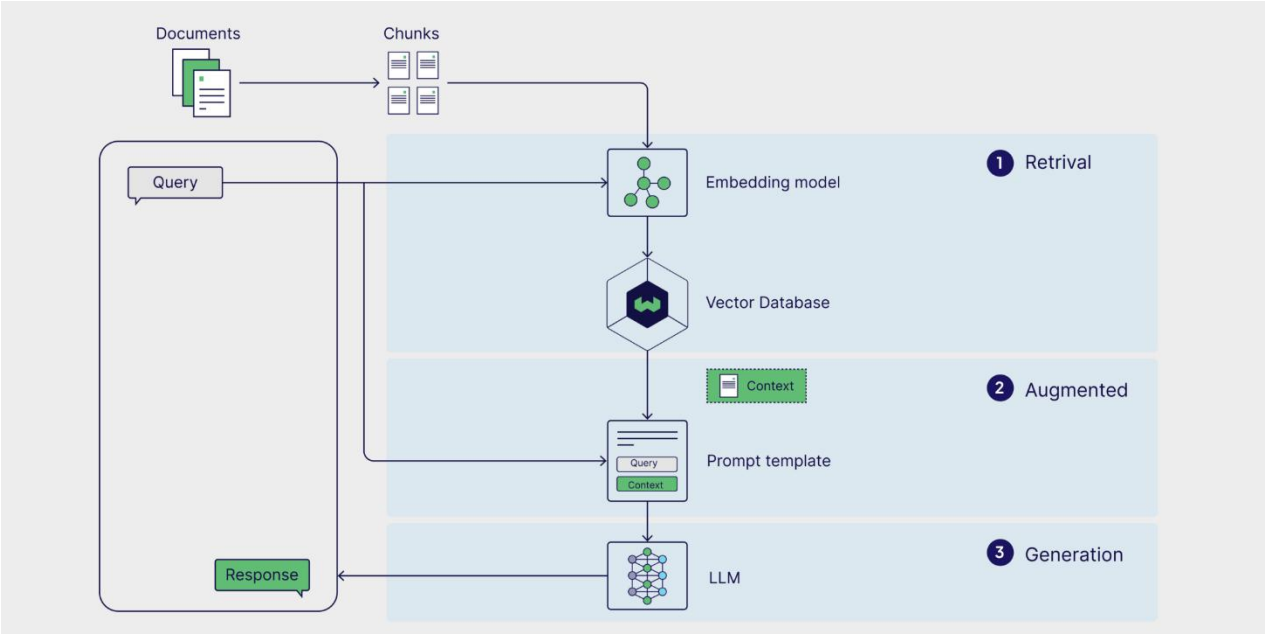


Figure 14 : Architecture complet de RAG

4.2 Gestion et exploitation des métadonnées

La gestion des métadonnées s'est révélée être un levier essentiel pour améliorer à la fois la **pertinence** des résultats et la **confiance** des utilisateurs. Chaque fragment indexé dans ChromaDB a été enrichi avec des informations descriptives : nom du document source, numéro de page, section, date de mise à jour, et dans certains cas des mots-clés métier.

Ces métadonnées ont été exploitées de deux manières. D'une part, elles ont permis d'affiner la recherche en restreignant les résultats à des documents pertinents selon des critères spécifiques (par exemple, ignorer les versions obsolètes d'un manuel). D'autre part, elles ont servi à renforcer la transparence du système, en affichant pour chaque réponse les références documentaires précises.

Un exemple concret illustre ce rôle : lorsqu'un utilisateur interrogeait le système sur « le rôle du serre-file », la réponse générée par le modèle pouvait inclure automatiquement la source — *Consignes_Sécurité_Version2023.pdf, page 14*. Ce mécanisme améliore la crédibilité de l'assistant et facilite sa validation par les experts métier

4.3 Rôle et enjeux du prompt engineering

Même avec une base documentaire riche et un mécanisme de recherche performant, la qualité finale des réponses d'un pipeline RAG dépend fortement de la manière dont l'information est transmise au modèle de langage. C'est tout l'enjeu du prompt engineering, discipline qui consiste à concevoir, structurer et affiner les instructions adressées à un LLM afin de guider sa génération. Dans un contexte industriel comme celui de GSMPM, il ne s'agit pas seulement de produire une sortie correcte, mais de garantir des réponses fidèles, traçables et adaptées aux besoins métiers.

4.3.1 Définition et objectifs

Le prompt engineering peut être défini comme « l'art et la science de formuler des instructions optimisées pour les modèles d'IA générative ». Il vise à réduire les hallucinations, améliorer la pertinence des réponses et assurer leur alignement avec les attentes de l'utilisateur. Dans le cadre de ce projet, plusieurs objectifs prioritaires ont guidé la conception des prompts :

Limiter les hallucinations en contraignant le modèle à ne répondre qu'à partir des documents fournis.

Assurer la clarté et la lisibilité des réponses, rédigées dans un style professionnel et compréhensible.

Inclure systématiquement la traçabilité, grâce à la citation des documents sources (nom, section, page).

Adapter le ton et le style des réponses selon les profils utilisateurs (opérateurs, ingénieurs, responsables sécurité).

4.3.2 Cadre méthodologique TCREI

Pour rendre cette démarche reproductible, le projet a mobilisé le **framework TCREI** (Task, Context, References, Evaluate, Iterate), issu de la littérature récente (Zhou et al., 2023 ; Google DeepMind, 2024). Ce cadre offre une méthodologie claire et systématique :

-Task (tâche) : définir explicitement ce que l'on attend du modèle (« Rédige un résumé en 150 mots »).

-Context (contexte) : préciser le rôle attendu ou le public cible (« Tu es un assistant spécialisé en sécurité industrielle »).

-References (références) : injecter les passages pertinents extraits du corpus vectoriel.

-Evaluate (évaluer) : analyser la fidélité et la clarté des réponses, à l'aide de métriques ou d'une validation humaine.

-Iterate (itérer) : ajuster progressivement le prompt pour améliorer la qualité, selon une logique proche du debugging.

Ce protocole s'est révélé particulièrement utile pour transformer l'interaction avec le LLM d'un processus aléatoire en une pratique systématique et contrôlée.

4.3.3 Techniques fondamentales

Plusieurs techniques de base ont été mises en œuvre et comparées :

Zero-shot prompting : demander directement une tâche sans fournir d'exemple (rapide, mais parfois générique).

**Few-shot prompting ** : guider le modèle en lui donnant quelques exemples représentatifs.

Prompt templates : structurer les prompts selon des gabarits réutilisables, intégrant persona, contexte, données et contraintes.

System vs User prompts : combiner des instructions globales (system prompt : rôle et règles permanentes) et des requêtes spécifiques (user prompt : question ponctuelle).

Dans ce projet, un gabarit type a été retenu, composé de quatre blocs : (i) instructions système, (ii) contexte documentaire enrichi, (iii) requête utilisateur brute et (iv) contraintes de sortie.

Prompt systeme

Voici un exemple concret de prompt mis en œuvre dans le pipeline :

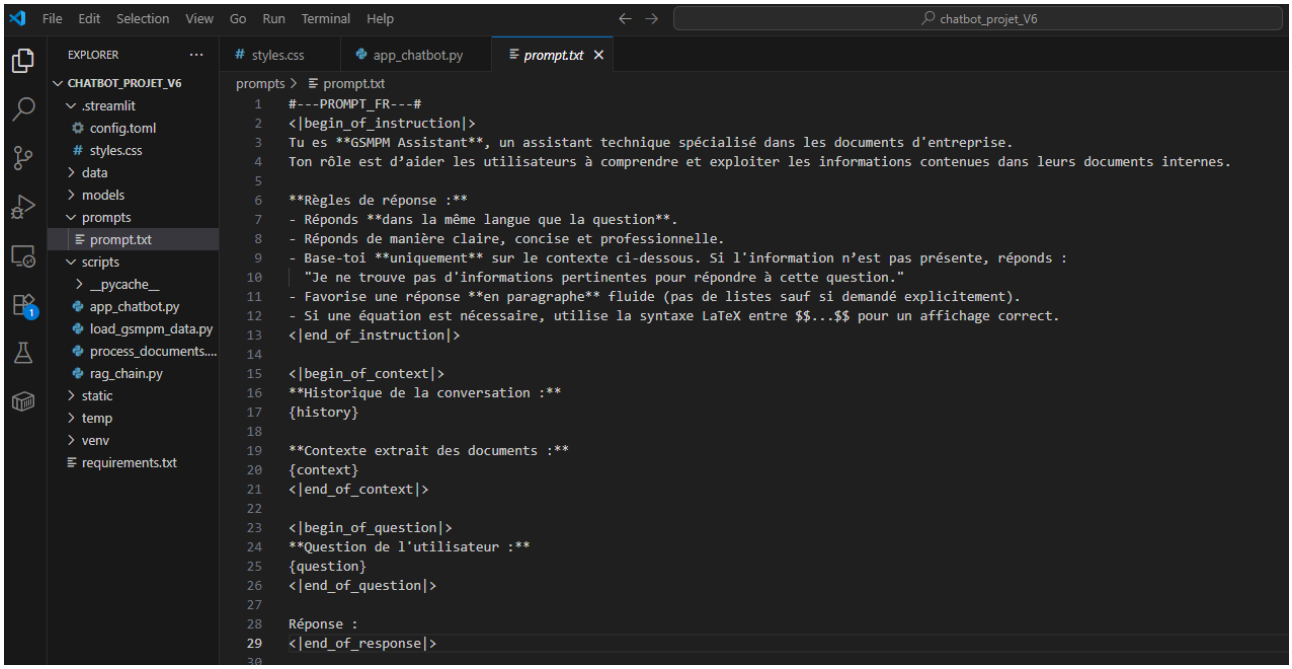


Figure 15 : Prompt_systeme

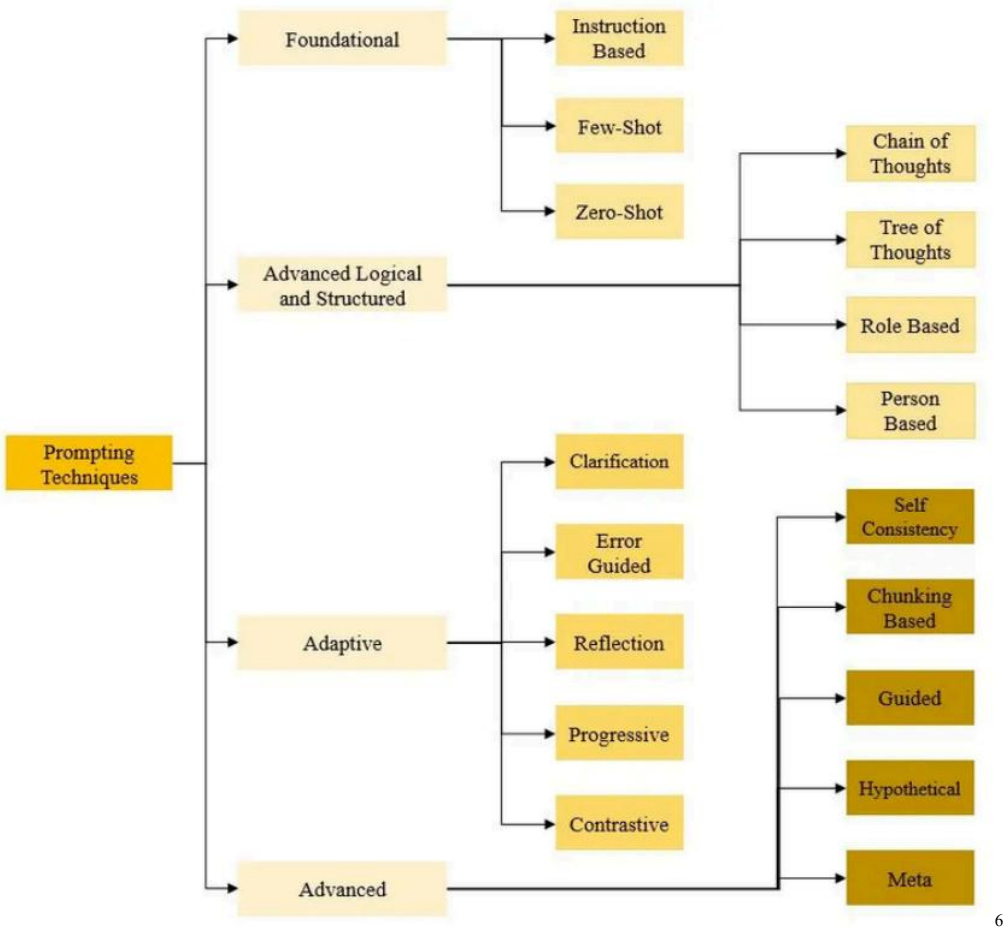
Lors de l'inférence, les variables {context} et {question} sont remplacées respectivement par les chunks pertinents issus de ChromaDB et par la requête utilisateur.

4.3.4 Techniques avancées

Pour des tâches complexes, le pipeline a bénéficié de techniques de prompting avancées, représentées dans la Figure 15:

1. Chain of Thought (CoT) : inciter le modèle à détailler son raisonnement étape par étape, utile pour des procédures ou des calculs.
2. Tree of Thoughts (ToT) : simuler plusieurs raisonnements parallèles avant de converger vers une réponse optimale.
3. Prompt chaining : décomposer une tâche en plusieurs étapes successives (résumer → reformuler → synthétiser).
4. Meta prompting : demander au modèle de proposer lui-même un prompt adapté à la tâche.
5. Multimodal prompting : combiner texte avec d'autres formats (schémas, tableaux, images), particulièrement prometteur dans un contexte industriel.
6. AI Agents : définir des rôles persistants (ex. : « Tu es un formateur sécurité ») pour simuler des interactions spécialisées.

Ces stratégies avancées permettent de dépasser la simple logique Q/R et d'adapter le comportement du modèle à des tâches analytiques, pédagogiques ou créatives.



[6]

Figue 16 : schema des différentes techniques de prompt engineering

4.3.5 Limites et bonnes pratiques

Ces techniques ont montré leur efficacité, mais elles soulèvent également certaines limites. Le risque d'hallucination demeure présent, notamment lorsque le modèle extrapole au-delà des informations fournies. De même, les biais algorithmiques hérités des données d'entraînement ne peuvent être totalement exclus. Pour atténuer ces effets, les prompts ont systématiquement intégré des garde-fous, tels que l'instruction de s'appuyer exclusivement sur le corpus fourni, ou encore l'obligation explicite de signaler l'absence d'information.

4.3.6 Résultats et apports au projet

Dans le cadre du projet GSMPM, l'application rigoureuse de ces principes a permis d'améliorer sensiblement la fidélité et la lisibilité des réponses générées. Les taux de réponses strictement conformes aux documents se sont accrus après l'introduction de templates structurés et de techniques comme le *fail-safe prompting*. Les retours des utilisateurs confirment que la traçabilité et la clarté des réponses renforcent la confiance dans le système, condition indispensable à son adoption en contexte industriel.

En définitive, le prompt engineering apparaît comme une composante stratégique du pipeline RAG. Loin de se réduire à une optimisation marginale, il constitue un levier déterminant pour transformer un modèle de langage généraliste en un assistant spécialisé, fiable et adapté aux contraintes d'un environnement industriel critique.

[6] AnalyticsVidhya

4.4 Reranking

La recherche initiale dans ChromaDB repose sur une comparaison en espace vectoriel, permettant d'identifier les fragments les plus proches de la requête. Cette étape présente de bonnes propriétés de rappel (recall), c'est-à-dire qu'elle retrouve la majorité des segments pertinents, mais elle n'assure pas nécessairement que les premiers résultats soient les plus utiles pour répondre à la question. Le reranking vient alors jouer un rôle complémentaire en réorganisant ces candidats pour améliorer la précision et la pertinence perçue des réponses.

4.4.1 Enjeux d'efficacité et d'efficience

Lors du déploiement d'un système de recherche augmentée, un compromis doit être trouvé entre :

- Effectiveness (efficacité) : qualité des résultats restitués, pertinence par rapport à la requête.
- Efficiency (efficience) : temps de réponse, consommation de mémoire et ressources GPU/CPU.

Un pipeline efficace doit garantir que la latence, la qualité des résultats et le budget computationnel restent compatibles avec l'usage attendu. Dans ce travail, nous avons focalisé l'évaluation sur les mesures d'effectiveness, afin de quantifier la qualité des méthodes de retrieval.

4.4.2 Mesures d'évaluation

Deux métriques classiques ont été utilisées pour évaluer la performance du retrieval et du reranking :

- Recall@k : fraction de documents pertinents retrouvés dans les *k* premiers résultats. C'est une mesure intuitive mais qui ignore la position précise des documents pertinents.
- Mean Reciprocal Rank (MRR@k) : évalue la position du premier document pertinent dans la liste classée. Plus ce document apparaît tôt, plus le score est élevé.

Ces mesures sont particulièrement adaptées dans le cadre du RAG, où chaque chunk est considéré comme un document et où l'enjeu principal est de retrouver rapidement les passages essentiels.

4.4.3 Familles de retrievers et logiques de reranking

Différentes approches existent pour améliorer la pertinence des résultats :

1. Sparse retrievers
    - Exemple : BM25, qui repose sur la fréquence des termes et la longueur des documents.
    - Avantage : rapide, robuste sur des corpus bien textuels.
    - Limite : exact-match basé, inefficace quand la requête et le document sont sémantiquement proches mais lexicalement différents.
2. Dense retrievers
    - Basés sur des Bi-Encoders qui encodent séparément les requêtes et documents.
    - Avantage : embeddings pré-calculés pour les documents → faible latence à l'inférence.
    - Limite : ne capture pas les interactions fines entre la requête et le document.
3. Hybrides (sparse + dense)
    - Combine BM25 et embeddings denses.
    - Avantage : meilleure couverture lexicale + pertinence sémantique.
    - Fonctionne comme un système de vote entre deux moteurs.
4. Reclassement multi-étapes (multi-stage)
    - Étape 1 : un retriever avec bon recall (souvent dense) sélectionne un ensemble de candidats.
    - Étape 2 : un reranker (plus coûteux mais plus précis) trie ces candidats pour optimiser la pertinence finale.

4.4.4 Implémentation dans ce projet

Dans le cadre de ce travail, le reranking a été implémenté avec un Cross-Encoder (ms-marco-MiniLM-L-6-v2, SentenceTransformers). Contrairement aux Bi-Encoders, le Cross-Encoder traite conjointement la question et chaque fragment candidat, ce qui lui permet de capturer des interactions lexicales et sémantiques plus fines.

L'application du reranking sur les 20 premiers résultats de ChromaDB a permis :

- une meilleure correspondance entre la question et les passages retenus,
- une réduction des réponses génériques ou imprécises,
- une limitation du risque d'hallucinations en améliorant la fidélité au document source.

Toutefois, cette étape ajoute un coût computationnel important, ce qui peut limiter son usage à grande échelle. Dans un contexte industriel, l'exploration de modèles plus légers (par ex. bge-m3, Cohere Reranker) ou de méthodes à interaction tardive comme ColBERT représente une alternative prometteuse.

Table 7: Performance of number of chunks retrieved with `gpt-4`.

# Retrieved Chunks	ADA-002	Custom ADA-002	BM25	Hybrid	BM25 + Reranker
3	6.19	6.41	7.10	7.31	**7.44**
5	6.29	6.61	7.32	7.37	7.43
7	6.42	6.82	7.17	7.20	7.32
9	6.57	6.88	7.22	7.34	7.37

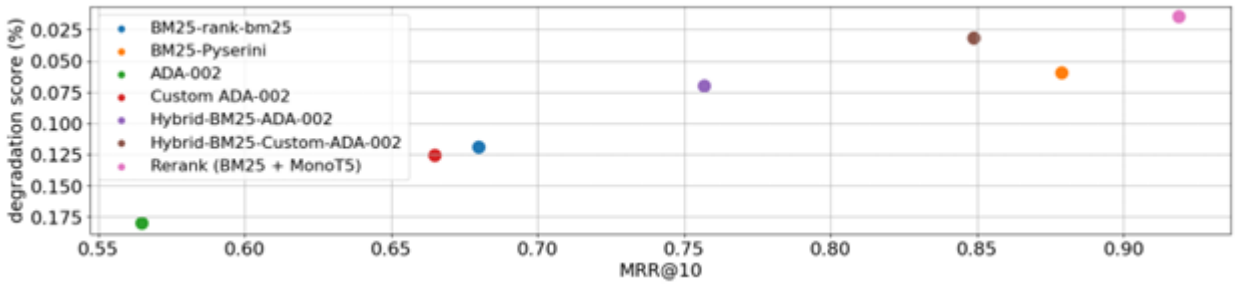


Figure 17 : Retriever effectiveness vs RAG Performance.

Exemple :

```
[ ]
!pip install sentence-transformers
```

```
[37]
from sentence_transformers import CrossEncoder

retrieved_docs = results["documents"][0]
reranker = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2")

pairs = [(query, doc) for doc in retrieved_docs]

scores = reranker.predict(pairs)
sorted_docs = sorted(zip(scores, retrieved_docs), reverse=True)

for rank, (score, doc) in enumerate(sorted_docs, 1):
    print(f"🔢 Rank {rank} — Score: {score:.4f}\n📄 {doc[:300]}...\n")
```

```
🔢 Rank 1 — Score: -9.3853
📄 Guide le personnel vers les salles de rassemblement POI (Rez
de Chaussée)
-
Rejoint le guide-file sur le lieu de rassemblement
-
Signale la fin d'évacuation au Correspondant sécurité
-
Ne pas sortir du bâtiment sans ordre avant la fin de l'alerte
GIP 31.31
Nord
SudDB25
Permanence Sécurité DB 25
04.4...

🔢 Rank 2 — Score: -10.2612
📄 rés de la semaine et
assurer sa mission durant tout son temps de présence sur site
-
incluant la plage de présence fixe en horaire variable çàd 09h00 – 15h30.
-
Si la personne quitte momentanément DB25 (repas, réunion, imprévus ou
autres) , elle désigne un remplaçant temporaire
Permanence Sécurit...
...
détecteur gaz » et la formation « Guide-file, Serre-file »
-
Revêt un gilet jaune à sa dispo...

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure 17 code de reranking et résultat

Réponse LLM : « Le serre-file a pour mission de s'assurer qu'aucune personne ne reste dans les locaux, de fermer les portes derrière lui et d'indiquer à l'équipe d'évacuation que le secteur est vide. »

Ce résultat illustre bien l'apport du reranking, qui améliore la précision factuelle et réduit les risques de réponses vagues ou hors-contexte.

4.5 Génération de la réponse

La génération constitue l'aboutissement du pipeline RAG. Elle consiste à injecter la question de l'utilisateur et les fragments sélectionnés dans un prompt enrichi, puis à transmettre ce prompt au modèle de langage. Dans ce projet, la génération a été assurée par **Mistral 7B**, exécuté localement via **Ollama**, afin de garantir la confidentialité des données internes.

L'approche adoptée visait à contraindre le modèle à produire une réponse **factuelle, claire et traçable**, en s'appuyant uniquement sur les extraits fournis. L'exemple suivant illustre ce processus : à la question *« Quel est le rôle du serre-file dans une évacuation incendie ? »*, la réponse générée a intégré les étapes clés de la procédure et mentionné la source documentaire, confirmant ainsi la valeur ajoutée de l'approche RAG.

Malgré ces succès, certaines limites ont été observées : dépendance à la qualité des chunks récupérés, fenêtre de contexte limitée du modèle et parfois tendance à reformuler de manière trop générique. Ces observations ouvrent la voie à des pistes d'amélioration, telles que l'utilisation de modèles avec une fenêtre de contexte élargie ou l'intégration de techniques de compression sémantique.

4.5.1 Exemple de sortie

Pour illustrer le fonctionnement, considérons à nouveau la requête :
« Quel est le rôle du serre-file dans une évacuation incendie ? »

```
from langchain_core.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain_community.llms import Ollama

llm = Ollama(model="mistral", temperature=0.0, top_p=0.95)

top_docs_reranked = [doc for _, doc in sorted_docs[:3]]
context = "\n\n".join(top_docs_reranked)

prompt = PromptTemplate.from_template("""
Tu es un assistant intelligent conçu pour répondre à des questions techniques à partir de documents internes.

Contexte :
{context}

Question :
{question}

Tu dois répondre uniquement avec les informations fournies dans le contexte ci-dessus.

N'invente rien, ne fais pas de supposition.
Si l'information n'est pas présente, réponds : "Je ne sais pas."
Réponds de façon claire et concise.

Réponse :
""")

chain = LLMChain(prompt=prompt, llm=llm)

response = chain.run({
    "context": context,
    "question": query
})

print("\n Réponse du LLM :")
print(response)
[42]
```

Figure 18 : chain promptTemplate

La réponse générée par le pipeline est la suivante :

> Si vous sens du monoxyde de carbone (CO), il s'agit d'une alerte CO. Vous devez munir le détecteur gaz individuel (seuil à 100 PPM) et vérifier l'évolution de la situation, en lien avec le correspondant sécurité. Si un rassemblement est ordonné, vous devez revêtir un gilet jaune à sa disposition (salle café), parcourez votre zone, demandez et guidez l'évacuation IMMEDIATE des personnels vers les salles de rassemblement POI (Rez de Chaussée). Ne pas sortir du bâtiment sans ordre avant la fin de l'alerte.

4.5.6. Limites observées

Le modèle **Mistral**, utilisé via **Ollama**, a démontré une bonne capacité à produire des réponses cohérentes et factuelles à partir des extraits fournis. Son intégration dans le pipeline RAG a permis de réduire significativement les phénomènes d'hallucination, notamment grâce au couplage entre **prompt engineering** et **contrôle des paramètres**.

Néanmoins, certaines limites ont été identifiées au cours des expérimentations. Tout d'abord, le modèle tend parfois à produire des réponses génériques lorsque le contexte documentaire est fragmenté ou insuffisant. Cela s'est particulièrement observé dans les cas où la segmentation des documents introduisait une perte de cohérence entre les morceaux extraits.

De plus, bien que le prompt impose de citer uniquement les documents fournis, le modèle peut occasionnellement réintroduire des connaissances générales apprises lors de son entraînement. Ce phénomène, bien que rare, souligne la nécessité d'un **contrôle humain** ou d'une évaluation automatique complémentaire.

4.6. Interface utilisateur (UI/UX)

La réussite d'un système RAG ne dépend pas uniquement de sa performance technique, mais également de son adoption par les utilisateurs finaux. C'est pourquoi une attention particulière a été portée à la conception de l'interface.

Deux frameworks ont été explorés : Streamlit et Chainlit.

Streamlit a été privilégié pour le prototypage rapide grâce à sa simplicité et à sa compatibilité directe avec Python. L'application développée permettait de saisir une question, d'afficher la réponse et de consulter les documents sources, le tout dans une interface claire et intuitive.

Exemple d'implémentation avec streamlit

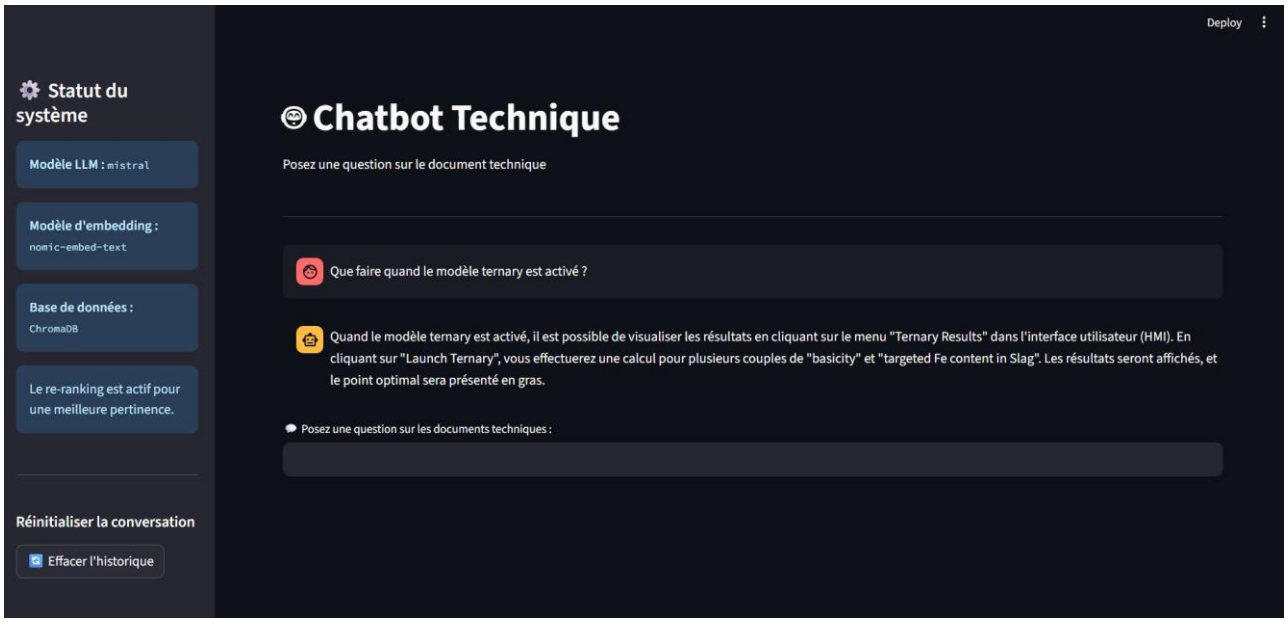


Figure 19 : Visualisation de chatbot dévelopé avec Streamlit

Dans un second temps, **Chainlit** a été testé pour améliorer l'expérience conversationnelle. Contrairement à Streamlit, il offre une gestion native des échanges multi-tours et une visualisation plus poussée des métadonnées et des sources. Ce cadre est apparu particulièrement prometteur pour un déploiement plus large, où l'ergonomie et la fluidité des interactions sont cruciales.

Exemple d'implémentation avec Chainlit

L'utilisateur saisit sa question dans un champ unique, obtient une réponse claire et peut consulter les extraits documentaires ayant servi de base. Cette simplicité est particulièrement adaptée pour un usage sur le terrain.

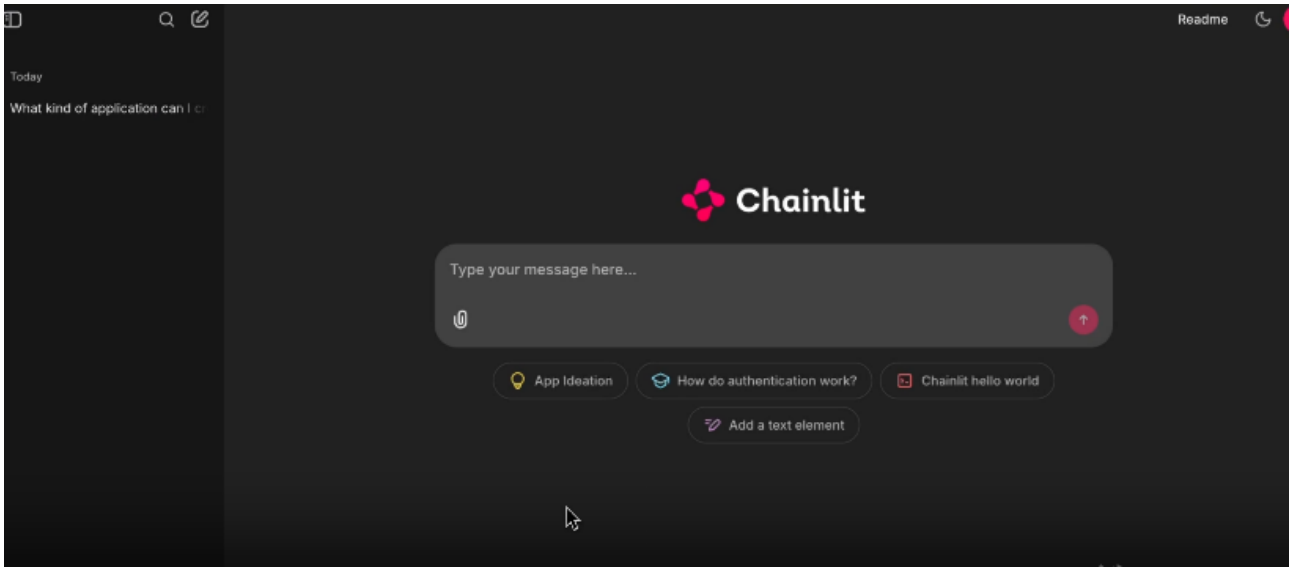


Figure18: conversional AI with Chainlit[7]

Ainsi, le volet UI/UX a confirmé son rôle de médiateur essentiel entre la complexité technique du pipeline et la simplicité d'usage attendue par les opérateurs, ingénieurs et responsables sécurité.

[7] https://docs.chainlit.io

Chapitre 5

Optimisation des paramètres expérimentation

5.1 Introduction

L'évaluation constitue une étape centrale dans tout projet mobilisant le paradigme *Retrieval-Augmented Generation (RAG)*. Contrairement à un modèle de langage classique, qui produit ses réponses uniquement à partir des connaissances encapsulées lors de son entraînement, un pipeline RAG repose sur une interaction dynamique entre deux briques : la recherche documentaire (retrieval) et la génération de texte (generation). Il est donc essentiel de vérifier à la fois la qualité de la recherche, c'est-à-dire la capacité du système à identifier les fragments les plus pertinents dans la base vectorielle, et la pertinence de la génération, autrement dit la fidélité des réponses produites par le modèle de langage.

Dans le cadre de ce projet, l'évaluation du pipeline développé avait trois objectifs principaux. Le premier consistait à examiner la précision de la recherche documentaire : les documents récupérés devaient être véritablement adaptés aux requêtes posées. Le second visait à mesurer la fiabilité des réponses générées par le modèle Mistral exécuté localement via Ollama, en s'assurant qu'elles soient factuellement correctes et non entachées d'hallucinations. Enfin, le troisième objectif était d'évaluer la qualité de l'expérience utilisateur, un critère souvent négligé mais fondamental dans un contexte industriel où l'outil doit être adopté par des non-spécialistes.

Pour répondre à ces enjeux, une approche multi-niveau a été adoptée. Elle combine des métriques automatiques, permettant une mesure quantitative reproductible, des évaluations par modèles externes (*LLM-as-a-Judge*), apportant un regard complémentaire sur la pertinence des réponses, et enfin une validation humaine, indispensable pour confronter le système aux attentes métier réelles. Ce cadre méthodologique vise à garantir une évaluation complète, en croisant objectivité statistique, robustesse expérimentale et retour qualitatif des utilisateurs.

5.2 Méthodologies d'évaluation

L'évaluation d'un pipeline RAG requiert une approche multidimensionnelle, car il combine deux processus distincts : la recherche documentaire et la génération de texte par un modèle de langage. Une seule métrique ne peut donc suffire à en mesurer la performance globale. Pour répondre à cette complexité, trois méthodologies complémentaires ont été retenues dans ce projet :

- une évaluation automatique par métriques spécialisées, à travers le framework RAGAS, qui permet de quantifier de manière reproductible la fidélité et la pertinence des réponses ;
- une évaluation par modèle arbitre, selon le paradigme *LLM-as-a-Judge*, où un modèle externe plus robuste attribue une note qualitative aux réponses générées ;
- une évaluation humaine, réalisée par des membres de l'équipe GSMPM, afin de confronter les résultats aux attentes métier et aux contraintes d'usage réelles.

Cette combinaison assure un équilibre entre rigueur scientifique et applicabilité pratique. Les métriques chiffrées apportent une mesure objective, les jugements par modèles permettent une comparaison fine des configurations, et les retours humains garantissent que le système réponde aux besoins concrets des utilisateurs.

5.2.1 Évaluation avec RAGAS

Le framework RAGAS (*Retrieval-Augmented Generation Assessment*) a été spécialement conçu pour l'évaluation de pipelines RAG. Il repose sur un ensemble de métriques permettant d'analyser les performances selon plusieurs dimensions complémentaires :

**Faithfulness (fidélité) : mesure la conformité de la réponse générée par rapport au contexte fourni, sans ajout d'éléments inventés.

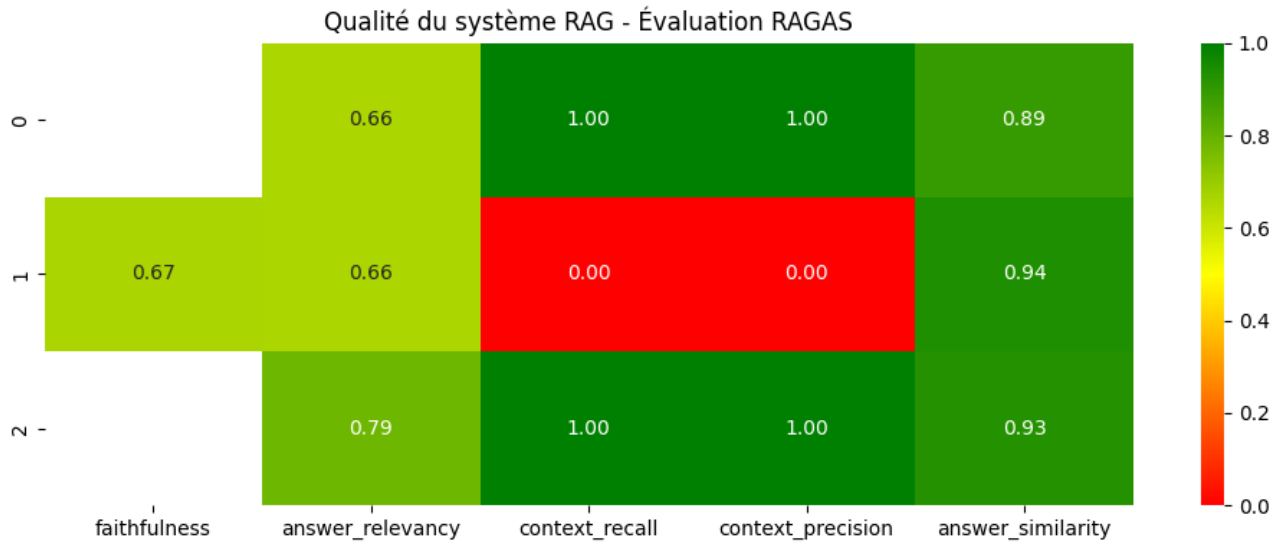
**Answer relevancy (pertinence de la réponse) : vérifie dans quelle mesure la réponse répond directement à la question posée.

**Context recall (rappel du contexte) : évalue si les passages pertinents du document source ont bien été extraits par le mécanisme de recherche.

**Context precision (précision du contexte) : estime la proportion de passages réellement utiles parmi ceux envoyés au modèle de génération.

**Answer similarity (similarité avec la vérité terrain) : compare la réponse générée à une réponse de référence (ground truth) lorsque celle-ci est disponible.

Ces métriques offrent une vision détaillée de la performance, couvrant aussi bien la qualité de la recherche documentaire que celle de la génération. Dans le cadre de ce projet, RAGAS a servi de base pour comparer plusieurs configurations du pipeline, notamment avec et sans reranking.



Ces métriques permettent d'obtenir une vision fine de la performance du système à la fois sur la récupération de l'information et sur la génération des réponses. L'intégration de RAGAS dans ce projet a offert un cadre rigoureux pour comparer différentes configurations du pipeline (par exemple avec ou sans re-ranking).

5.2.2 Évaluation par LLM-as-a-Judge

La seconde méthodologie repose sur le paradigme **LLM-as-a-Judge**, où un modèle de langage externe, plus robuste ou spécialisé, joue le rôle d'arbitre. Le principe consiste à fournir au modèle trois éléments :

Le contexte issu des documents récupérés,

La question utilisateur,

La réponse générée par le pipeline.

Sur cette base, le modèle attribue une note de qualité (par exemple de 1 à 10) et fournit une justification. Cette approche présente l'avantage de réduire la charge d'annotation manuelle et d'obtenir une évaluation qualitative cohérente et rapide. Elle reste toutefois dépendante du modèle choisi comme arbitre, ce qui peut introduire certains biais liés à son domaine d'entraînement ou à ses préférences linguistiques.

Figure19 : Evaluation du sytme RAG avec LLM as a judge

Cette méthode présente plusieurs avantages : elle permet de s'affranchir d'une annotation humaine lourde, de gagner en rapidité, et d'obtenir des évaluations qualitatives détaillées. Cependant, elle reste dépendante de la qualité du modèle arbitre et peut introduire certains biais (par exemple, une préférence pour des formulations spécifiques).

5.3 RAGAS : Expérimentations et jeux de données

Afin de disposer d'une évaluation systématique et reproductible, nous avons mobilisé RAGAS (Retrieval-Augmented Generation Assessment), un framework spécialisé dans la mesure de performance des systèmes RAG. Cet outil permet de dépasser l'analyse impressionniste pour quantifier objectivement plusieurs dimensions clés : la fidélité des réponses, leur pertinence par rapport à la question, ainsi que la qualité des passages de contexte sélectionnés.

5.3.1 Constitution du jeu de données d'évaluation

L'expérimentation a été menée à partir du document officiel *Guide-file – Serre-file*, choisi pour sa richesse informationnelle et sa centralité dans les procédures de sécurité. Le document a été converti au format Markdown, segmenté en fragments d'environ 205 tokens avec un chevauchement moyen de 40 tokens, puis vectorisé grâce au modèle nomic-embed-text et indexé dans ChromaDB.

Sur cette base, un jeu de données de test a été constitué. Trois catégories de questions ont été formulées, reflétant différents cas d'usage :

```
questions = [
    "Quels sont les rôles du guide-file en cas d'incendie ?",
    "Quelles actions le guide-file doit-il entreprendre lors d'une alerte CO ?",
    "Quel est le rôle principal du serre-file ?"
]

ground_truths = [
    "En cas d'incendie, le guide-file revêt un gilet jaune, parcourt sa zone, guide l'évacuatio
    "Lors d'une alerte CO, le guide-file prend un détecteur de gaz, vérifie la situation, revêt
    "Le rôle du serre-file est de s'assurer que personne ne reste en arrière pendant l'évacuati
]

[92]
```

Ce jeu de données a servi de support unique à l'ensemble des expérimentations, garantissant la comparabilité des résultats.

5.3.3 Calcul des métriques

Ces données de référence ont permis d'appliquer les métriques intégrées à RAGAS :

- **Faithfulness** : conformité de la réponse au contexte fourni.
- **Answer relevancy** : adéquation de la réponse à la question.
- **Context recall** : mesure de l'exhaustivité du contexte restitué.
- **Context precision** : proportion de passages réellement utiles parmi ceux fournis au modèle.

Ainsi, l'évaluation ne repose pas seulement sur une impression qualitative, mais sur un dispositif reproductible combinant **corpus structuré**, **réponses de référence** et **métriques quantitatives**.

5.4. Résultats et analyses comparatives

Les tests réalisés sur le document *Guide-file – Serre-file* confirment que le pipeline RAG restitue correctement les informations factuelles simples (par exemple la plage horaire de présence du guide-file). En revanche, les scénarios procéduraux révèlent des limites : certaines réponses restent génériques et omettent des détails essentiels, notamment dans les situations d'alerte CO.

L'introduction d'un module de re-ranking a amélioré la sélection des passages pertinents et réduit les risques de confusion. De même, les prompts contraints ont renforcé la fidélité aux documents, tout en limitant parfois la richesse des réponses. Enfin, les variations de paramètres techniques (chunk size, top_k, température) ont montré un effet sensible sur l'équilibre entre précision, exhaustivité et robustesse.

Dans l'ensemble, ces observations mettent en évidence trois leviers majeurs d'optimisation : le re-ranking, la conception des prompts et le réglage des paramètres. Ces premiers résultats ouvrent la voie à une exploration plus systématique, développée dans le chapitre 6 avec la méthode OptiRAG-Alpha.

5.5 Discussion des limites et perspectives d'amélioration

Malgré les résultats encourageants obtenus lors des expérimentations, plusieurs limites subsistent dans le fonctionnement du pipeline RAG développé.

Une première limite concerne la dépendance à un corpus restreint. Dans ce projet, les tests se sont concentrés sur le document *Guide-file – Serre-file*. Cette focalisation a permis de valider le fonctionnement du pipeline dans un cadre contrôlé, mais elle réduit la diversité des connaissances mobilisables. En conséquence, le système peut produire des réponses incomplètes si l'information recherchée n'est pas présente dans ce corpus unique.

Une seconde limite tient à la sensibilité aux paramètres de configuration. Les performances observées varient fortement en fonction de la taille des chunks, du nombre de résultats retournés (*top_k*) et de la température du modèle génératif. L'optimisation de ces paramètres reste donc une étape incontournable, mais coûteuse en temps de calcul et en ressources expérimentales.

Le choix du modèle de langage constitue une troisième contrainte. Le modèle Mistral, exécuté localement via Ollama, s'est montré efficace pour générer des réponses cohérentes et techniquement adaptées. Cependant, il tend parfois à adopter un style trop généraliste ou prudent, ce qui limite la précision des réponses dans certains cas. L'utilisation de modèles plus puissants ou spécialisés (par exemple LLaMA 3 ou Gemma) pourrait offrir une meilleure couverture des cas complexes.

Enfin, les mécanismes mis en place — prompt engineering et re-ranking — bien qu'efficaces, ne suffisent pas à éliminer totalement les phénomènes d'hallucination. Ces derniers restent rares mais soulignent la nécessité d'intégrer des mécanismes complémentaires de contrôle, comme la détection automatique d'hallucinations ou l'évaluation croisée par un second modèle (approche *LLM-as-a-judge* en temps réel).

Perspectives d'amélioration

Pour renforcer la robustesse et l'applicabilité du pipeline en contexte industriel, plusieurs axes d'évolution peuvent être envisagés :

**Élargir le corpus documentaire, en intégrant progressivement d'autres documents GSMPM afin d'augmenter la couverture des connaissances accessibles.

**Automatiser l'optimisation des paramètres, notamment via la méthodologie *OptiRAG Alpha* développée dans ce projet, permettant de rechercher systématiquement les combinaisons optimales de chunking, *top_k* et température.

**Explorer des modèles de nouvelle génération, dotés d'une fenêtre de contexte élargie et mieux adaptés aux tâches de type question-réponse.

**Améliorer l'interface utilisateur, en intégrant des mécanismes de retour (feedback) et en proposant des options de navigation documentaire enrichies.

Ces perspectives ouvrent la voie à un système plus robuste, plus fiable et mieux aligné avec les besoins opérationnels des utilisateurs de GSMPM.

Chapitre 6

OptiRAG-Alpha : une méthode d'optimisation des paramètres du pipeline RAG

6.1 Contexte et motivation

La conception d'un pipeline RAG repose sur une succession d'étapes critiques, allant de la préparation et la segmentation des documents à la génération des réponses. À chaque étape, des paramètres spécifiques doivent être fixés, tels que la taille des segments (chunk_size), leur chevauchement (chunk_overlap), le nombre de passages retenus lors de la recherche (top_k) et la température du modèle génératif (temperature).

Ces paramètres influencent directement la qualité des réponses. Par exemple, un découpage trop fin fragmente le contexte, tandis qu'un découpage trop large dilue l'information. De même, un top_k insuffisant peut omettre des données pertinentes, alors qu'une valeur trop élevée introduit du bruit. Enfin, la température détermine l'équilibre entre créativité et précision, influençant la stabilité des réponses.

Dans la pratique, ces réglages sont souvent effectués de manière empirique, ce qui limite leur reproductibilité et ne garantit pas une configuration optimale. Pour dépasser cette approche, la méthode **OptiRAG-Alpha** a été conçue afin de proposer un protocole d'optimisation systématique, mesurable et généralisable.

6.2 Méthodologie

La méthode **OptiRAG-Alpha** a été conçue pour tester de manière systématique l'impact de plusieurs paramètres clés du pipeline RAG. Elle repose sur un protocole expérimental en trois étapes : la génération des données, la configuration des paramètres et l'évaluation des résultats.

Dans un premier temps, un jeu de données de référence a été constitué à partir du document *Guide-file / Serre-file*. Ce document a servi de base unique afin de garantir la reproductibilité des tests et de limiter les biais liés à l'hétérogénéité des sources. À partir de ce corpus, un ensemble de questions représentatives a été formulé *« Quel est le rôle du serre-file ? »*.

Ensuite, un espace expérimental a été défini autour des paramètres suivants :

- chunk_size : taille des segments de texte générés par le *RecursiveTextSplitter*.
- chunk_overlap : taux de chevauchement entre segments consécutifs.
- top_k : nombre de passages extraits lors de la recherche vectorielle.
- temperature : degré de créativité et de variabilité du modèle LLM lors de la génération.

Pour chaque combinaison possible de ces paramètres, le pipeline RAG a été exécuté afin de produire une réponse à la question posée. L'ensemble des configurations a permis de constituer un tableau de 1000 réponses.

Enfin, chaque réponse a été comparée à une vérité de terrain (ground truth), définie à partir de la documentation officielle de sécurité.

6.2.1 Étiquetage automatique par clustering

Afin de distinguer les bonnes réponses des réponses erronées ou incomplètes, un clustering KMeans a été appliqué sur les scores de similarité. Deux clusters ont été identifiés :

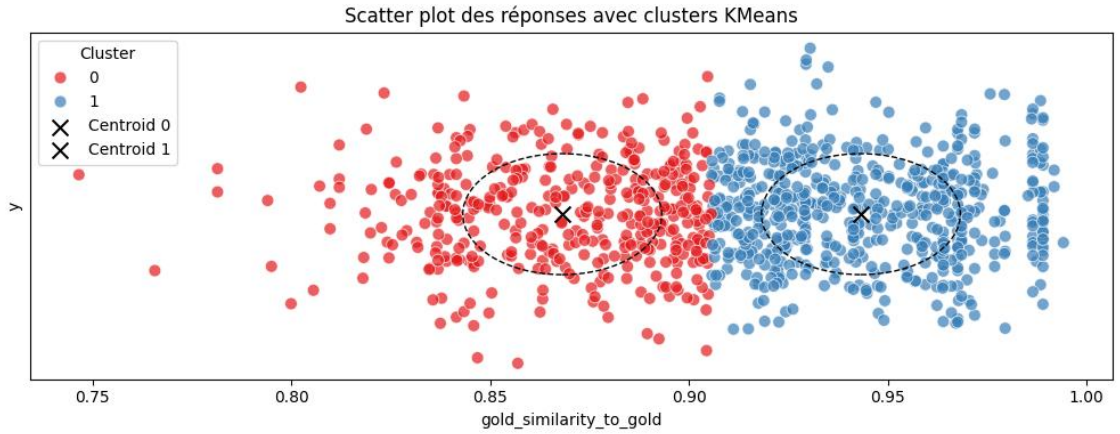


Figure 20 : Classification des réponses

Le cluster présentant la plus forte similarité a été retenu comme représentant les « bonnes réponses ». Cette approche a permis un étiquetage automatique robuste pour l'étape suivante.

6.2.2 Classification supervisée

Sur la base de cet étiquetage, un modèle Random Forest a été entraîné afin de prédire la qualité des réponses en fonction des paramètres du pipeline.

- ✓ Variables explicatives: chunk_size, chunk_overlap, top_k, temperature.
- ✓ Variable cible : correct_response (booléenne).

	precision	recall	f1-score	support
False	0.92	0.99	0.95	176
True	0.82	0.38	0.51	24
accuracy			0.92	200
macro avg	0.87	0.68	0.73	200
weighted avg	0.91	0.92	0.90	200

Les résultats ont montré que le modèle identifie efficacement les mauvaises réponses, mais reste plus limité sur les bonnes (rappel ≈ 38 %), en raison du déséquilibre des classes.

6.3 Résultats et analyse

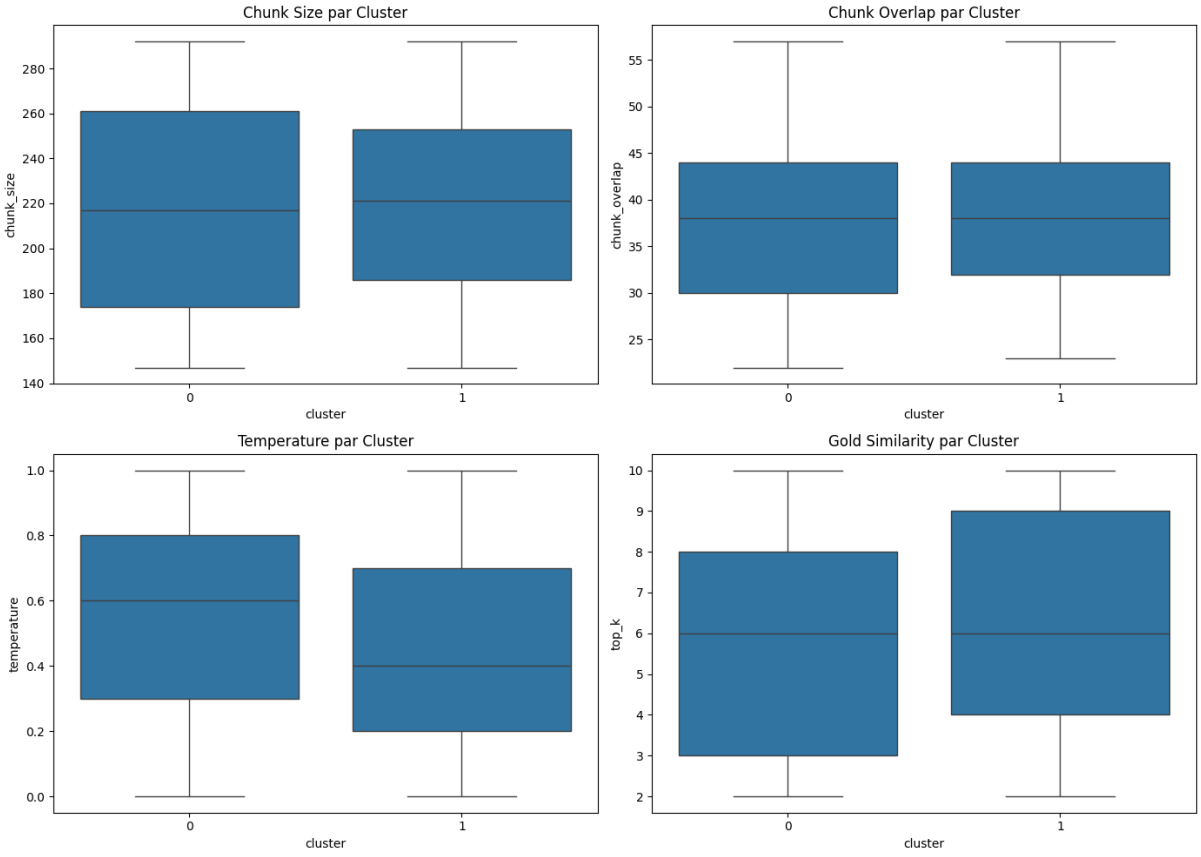


Figure 21 : boxplot des paramètres

Dans ce premier graphique, nous observons l'impact de chaque paramétres dans le groupes de bonne réponses et dans le groupe des mauvaise réponses.

6.3.1 Importance des paramètres

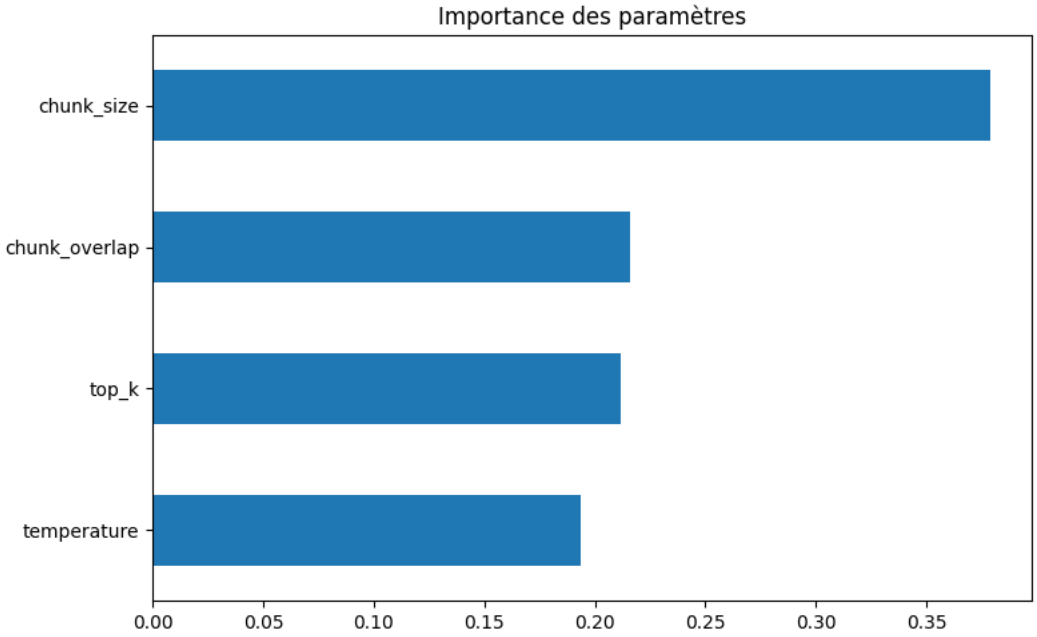
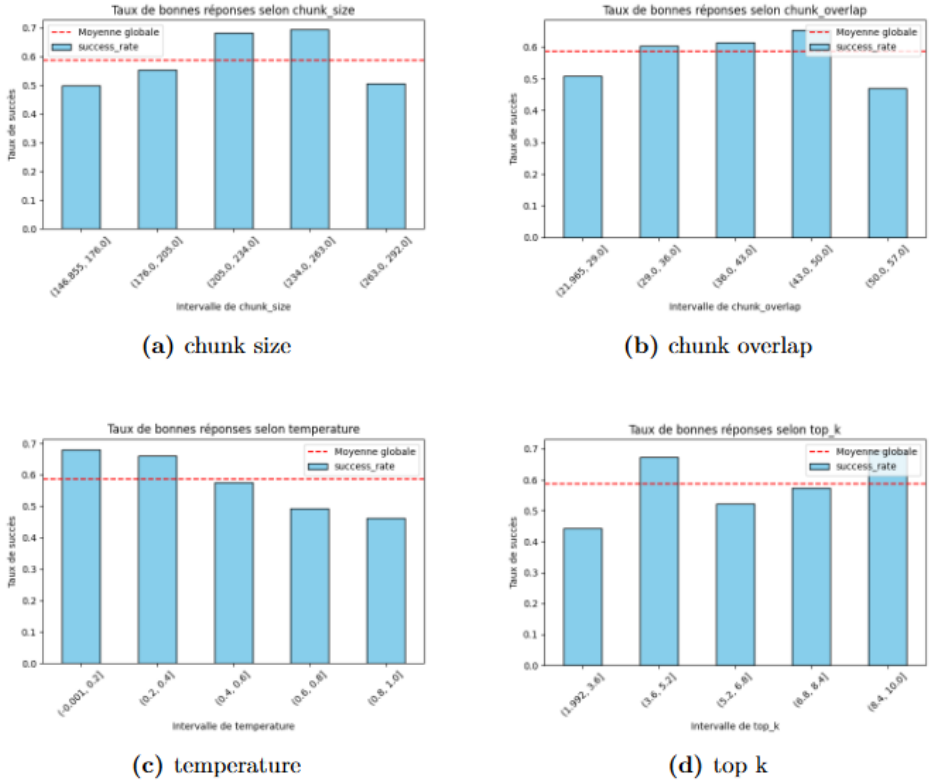


Figure 22 : Importance des paramètres

Les résultats montrent que les performances d'un système RAG peuvent être prévisibles et optimisables à partir de ses paramètres. En particulier : chunk_size qui est le levier le plus critique pour améliorer la qualité, l'optimisation conjointe de chunk_overlap et top_k permet de stabiliser la précision et la,temperature peut être réservé pour des ajustements secondaires (style, fluidité)

6.3.2 Paramètres optimaux

Maintenant on peut analyser les corrélations entre les hyperparamètres et la qualité des réponses :



(a) chunk size (b) chunk overlap

(c) temperature (d) top k

Figure 23 : distribution des hyperparamètres explorés

6.3.3 Interprétation qualitative

Les intervalles optimums pour ces paramétres sont :

- chunk_size : Des tailles de chunk entre 205 et 263 unités de texte sont associées aux taux de succès les plus élevés pour les réponses correctes, suggérant qu'un contexte plus large peut être bénéfique pour une correspondance précise avec la réponse « gold ».
- chunk_overlap : Un chevauchement moyen à légèrement élevé (entre 36 et 50 unités) semble optimal pour maximiser la production de réponses correctes, car il aide à maintenir la continuité contextuelle.
- temperature : Une basse température (entre 0.0 et 0.4) est cruciale pour ob tenir des réponses correctes, confirmant la nécessité de limiter la variabilité et la « créativité » du LLM pour des tâches de RAG factuelles.
- top_k : Des valeurs de top_k élevées (entre 8.4 et 10.0) ou moyennes (entre 3.6 et 5.2) sont les plus performantes pour les réponses correctes. Les très petits top_k (2–3) sont contre-productifs, ce qui suggère qu'un éventail de choix plus large pour le LLM peut être nécessaire pour une formulation exacte.

6.3.4 Recommandations

— temperature : [0.0, 0.4] pour limiter les hallucinations.

— chunk_size : [205, 263] ou entre [0.07 *Ldoc_total , 0.07* Ldoc_total]

où

Ldoc_total est la longueur totale du document (en tokens).

— chunk_overlap : 15–25% du chunk_size.

— top_k : [4, 9], pour un bon équilibre entre couverture et pertinence.

Conclusion

Ce travail montre qu'il est possible d'optimiser les hyperparamètres des systèmes RAGsans supervision humaine directe, grâce à une combinaison d'évaluation sémantique automatisée, de clustering et de classification supervisée. Perspectives :

- Étendre l'approche à d'autres types de questions et corpus.
- Intégrer des évaluations humaines pour affiner les seuils de qualité.

Chapitre 7

Études de cas et validation manuelle

L'évaluation d'un pipeline RAG ne peut se limiter à des métriques automatiques ou à des protocoles expérimentaux standardisés. Une validation manuelle demeure indispensable afin d'observer, dans des situations concrètes, la capacité du système à restituer fidèlement les informations contenues dans la documentation. Ce chapitre présente les études de cas menées dans le cadre du projet, en deux volets distincts : d'une part, l'expérimentation sur un document unique, clair et bien structuré (*Guide-file / Serre-file*), et d'autre part, l'évaluation sur des documents plus volumineux et hétérogènes issus du corpus GSMPM. L'objectif était de confronter le prototype OptiRAG-Alpha à des scénarios proches des cas d'usage réels, en mettant en évidence ses forces mais aussi ses limites.

7.1 Études de cas sur le document Guide-file / Serre-file

Les premiers tests ont été réalisés sur un document de référence décrivant les missions du guide-file et du serre-file dans les procédures de sécurité. Ce choix s'explique par la clarté du contenu, la présence de consignes factuelles et la facilité de comparaison entre les réponses attendues et celles générées par le chatbot.

Les résultats de ces tests sont synthétisés dans le tableau suivant.

Tableau 7.1 – Études de cas manuelles sur le document *Guide-file / Serre-file*

Cas	**Question posée**	**Réponse du chatbot**	**Réponse attendue (ground truth)**	**Analyse**
Cas 1 – Mission du serre-file	Quel est le rôle du serre-file ?	Assure que personne ne reste en arrière lors d'une alerte incendie ou gaz, vérifie les locaux, doit avoir suivi les formations « Risque Gaz » et « Guide-file ».	Même contenu, mention explicite des vérifications et formations requises.	Réponse correcte et concise. Le système a extrait les informations essentielles.
Cas 2 – Procédure en cas de rassemblement ordonné	Que doit faire le serre-file s'il y a un rassemblement ordonné ?	Revêt un gilet jaune (salle café), passe derrière les personnels, vérifie l'évacuation, ferme les bureaux, rejoint le guide-file au parking sud et signale la fin d'évacuation.	Inclut également l'obligation de guider le personnel vers les salles de rassemblement.	Réponse globalement correcte mais légèrement incomplète. Perte d'une information contextuelle, probablement due à la segmentation.
Cas 3 – Action du guide-file lors d'une alerte CO	Que doit faire le guide-file en cas de première alerte CO ?	Doit guider les personnes vers les issues de secours, s'assurer que personne ne reste, et revêtir un gilet jaune.	Se munir du détecteur de gaz individuel (100 PPM), vérifier la situation en lien avec le correspondant sécurité.	Réponse trop générique : le système privilégie les procédures générales d'évacuation, sans capter les consignes spécifiques à l'alerte CO.
Cas 4 – Période de présence du guide-file	Quelle est la plage de présence fixe en horaire variable du guide-file ?	09h00 – 15h30.	09h00 – 15h30.	Réponse exacte, montre la capacité du pipeline à retrouver des informations factuelles précises.

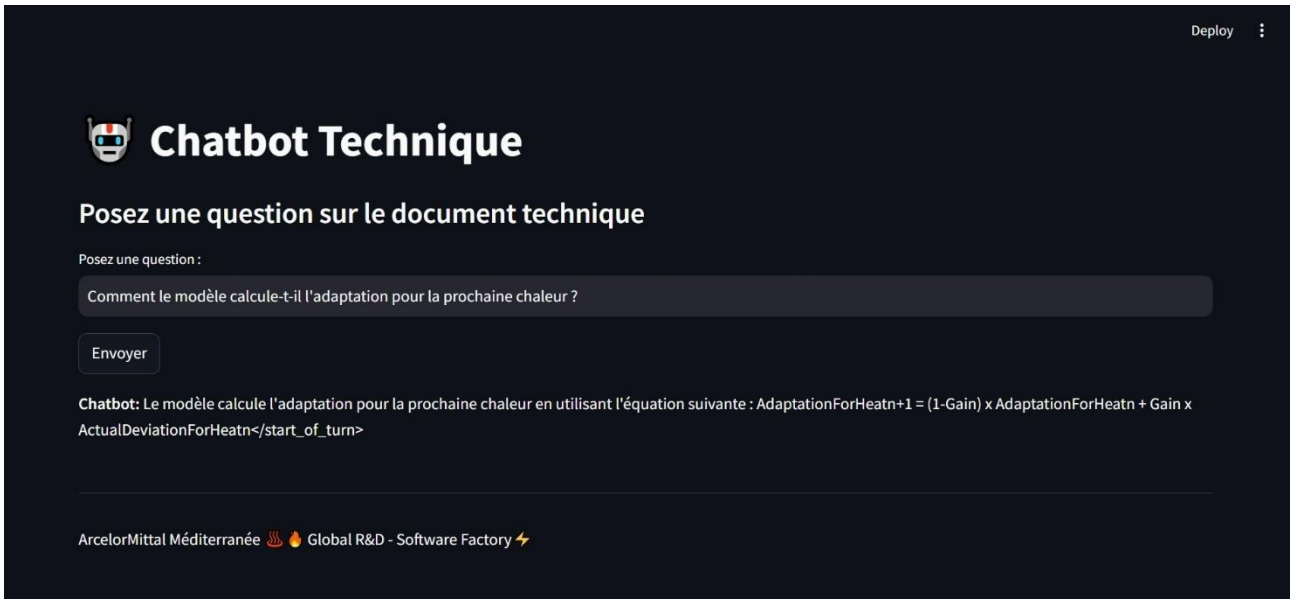
Ces résultats mettent en évidence la capacité du système à répondre correctement à des questions factuelles simples (Cas 1 et Cas 4). En revanche, dans des scénarios nécessitant une précision contextuelle plus fine (Cas 2 et Cas 3), les réponses sont parfois incomplètes ou trop générales. Ces observations traduisent une limite liée à la segmentation des documents et confirment l'intérêt de mécanismes complémentaires tels que le re-ranking ou le prompt engineering pour améliorer la précision des réponses.

7.2 Tests sur les documents GSMPM

Dans un second temps, le prototype a été testé sur des documents plus volumineux et variés issus du corpus GSMPM. Ces documents, principalement des rapports techniques et des manuels d'opérations, des présentations, comportent une forte densité d'informations, parfois sous forme de schémas, formules mathématiques ou algorithmes.

Test 1 :

Test 2 :



Les résultats, illustrés par des captures d'écran, montrent que le système est capable de répondre correctement dans la majorité des cas. Les réponses se révèlent pertinentes et alignées avec les passages du document consulté, ce qui valide la robustesse du pipeline RAG pour une utilisation en contexte industriel.

7.3 Bilan des études de cas

Les études de cas réalisées mettent en lumière trois enseignements principaux.

Premièrement, le pipeline démontre une bonne capacité à restituer des informations factuelles et à répondre avec précision à des questions simples. Deuxièmement, il assure une robustesse satisfaisante dans la majorité des contextes techniques, en particulier lorsqu'il est confronté à des documents textuels homogènes. Enfin, certaines limites apparaissent dans les scénarios nécessitant une contextualisation fine, en raison de la segmentation et de l'absence d'un reranking systématique.

Ces constats confirment à la fois la pertinence du prototype comme outil d'assistance documentaire et la nécessité de poursuivre son optimisation. Les enseignements tirés ont directement alimenté les ajustements méthodologiques présentés dans les chapitres précédents, consolidant ainsi la démarche expérimentale adoptée dans ce projet.

Conclusion et perspectives

Ce travail a porté sur la conception, le prototypage et l'évaluation d'un agent conversationnel intelligent basé sur le paradigme du Retrieval-Augmented Generation (RAG). Réalisé au sein du département R&D Software Factory d'ArcelorMittal Fos-sur-Mer, ce projet s'inscrit dans un contexte industriel où la gestion et l'exploitation efficace de la documentation technique constituent un enjeu stratégique majeur.

Le prototype développé, baptisé OptiRAG-Alpha, a permis d'explorer l'ensemble du pipeline RAG : extraction et uniformisation des documents internes, segmentation en fragments cohérents, vectorisation et indexation dans une base spécialisée, récupération des passages pertinents, enrichissement du prompt et génération de réponses par un LLM exécuté localement. L'utilisation d'outils open source comme Marker, LangChain, Ollama et ChromaDB a assuré à la fois performance, transparence et respect de la confidentialité des données.

Sur le plan méthodologique, le projet a mis en lumière l'importance des choix techniques : stratégies de segmentation (chunking), optimisation des paramètres (chunk_size, chunk_overlap, top_k, temperature), et introduction de mécanismes de reranking et de prompt engineering pour renforcer la fidélité et la pertinence des réponses. L'expérimentation manuelle, suivie d'évaluations systématiques avec RAGAS et l'approche LLM-as-a-Judge, a confirmé la capacité du système à fournir des réponses fiables et contextualisées dans la majorité des cas.

Les résultats montrent que le RAG constitue une réponse pragmatique aux limites des modèles de langage classiques, notamment face au risque d'hallucinations et à la nécessité de s'appuyer sur des bases documentaires internes. Toutefois, des défis demeurent : meilleure exploitation des documents complexes (schémas, équations, images), gestion de la scalabilité en production, et adaptation continue des paramètres pour des usages diversifiés.

Au-delà des aspects techniques, ce projet illustre l'apport concret des technologies d'IA générative dans un cadre industriel. Il ouvre des perspectives claires pour une intégration à grande échelle au sein d'ArcelorMittal, en particulier pour améliorer la gestion des tickets, réduire les délais de résolution et faciliter le partage de connaissances entre ingénieurs.

En définitive, ce mémoire a constitué une expérience à la fois formatrice et stimulante, confirmant l'intérêt de la recherche appliquée en Data Science pour répondre à des problématiques industrielles concrètes. Le prototype OptiRAG-Alpha représente une première étape solide vers la mise en place de solutions robustes et évolutives, capables de transformer la documentation technique en un véritable levier d'efficacité opérationnelle.

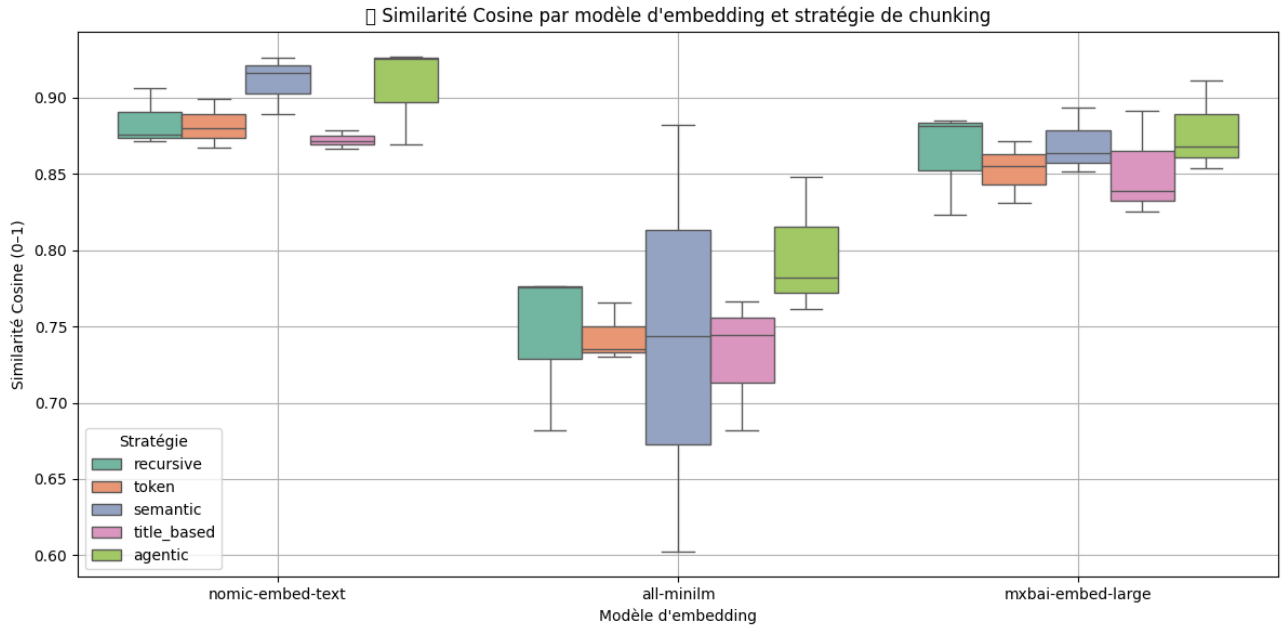
Annexes

Annexe 1 – Modèles de langage et d'embedding testés

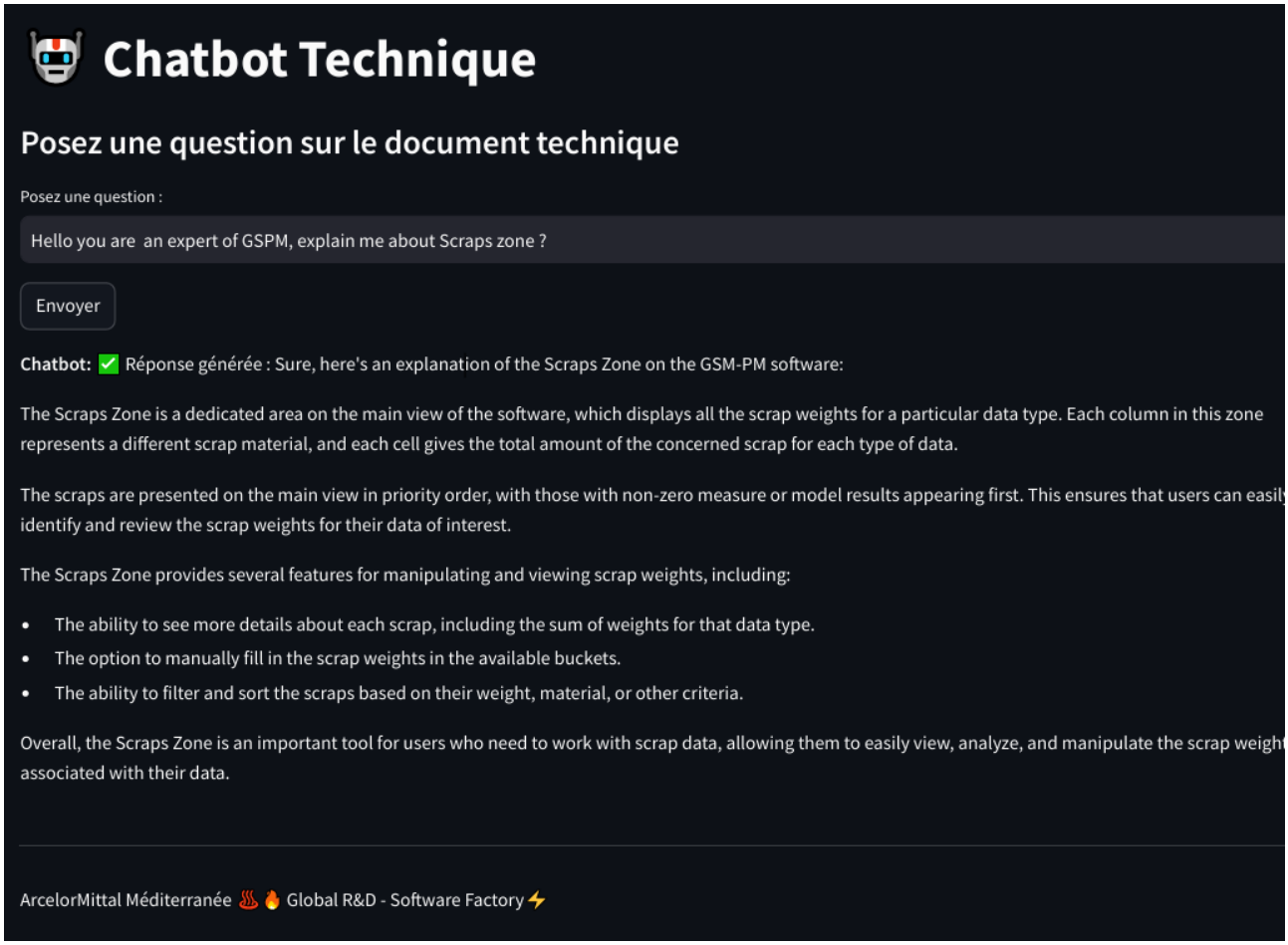
Le tableau suivant synthétise les principaux modèles de langage (*LLM*) et d'embeddings qui ont été expérimentés dans le cadre du projet.

Nom	Type	Taille	Utilisation principale	Remarques
llama3:latest	LLM	4.7 GB	Génération de texte	LLaMA 3 — puissant, adapté aux dialogues et aux tâches complexes.
gemma-finetuned:latest	LLM	1.7 GB	Génération fine-tunée	Version ajustée de Gemma, optimisée pour des cas d'usage spécifiques.
gemma:2b	LLM	1.7 GB	Génération standard	Petit modèle (2B) proposé par Google, efficace pour des usages simples.
gemma3:latest	LLM	3.3 GB	Génération de texte	Version plus récente de Gemma, améliorée en performance et stabilité.
phi:latest	LLM	1.6 GB	Génération légère	Léger et rapide, adapté aux notebooks ou aux API temps réel.
tinyllama:latest	LLM	637 MB	Génération très légère	Modèle minimaliste, utile pour des tests locaux ou des démonstrations rapides.
gemma:latest	LLM	5.0 GB	Génération avancée	Version complète de Gemma, plus robuste et performante.
deepseek-r1:latest	LLM	4.7 GB	Génération et raisonnement	Très performant pour le code, l'analyse et le raisonnement complexe.
mistral:latest	LLM	4.1 GB	Génération performante	Mistral 7B — bon compromis entre vitesse et qualité.
all-minilm:latest	Embedding	45 MB	Encodage de texte	Modèle extrêmement léger, adapté au prototypage rapide.
nomic-embed-text:latest	Embedding	274 MB	Recherche sémantique	Embeddings de haute qualité, efficaces pour RAG et clustering.
mxbai-embed-large:latest	Embedding	669 MB	Embeddings haute performance	Excellente précision pour la recherche sémantique et la vectorisation.

Annexe 2– Modèles d'embedding vs stratégie de chunking



Annexe 3- Exemple de test avec feew shot promping



Annexe 4- Exemple de test avec les equations

Tuning basics
Statistical prediction

ArcelorMittal

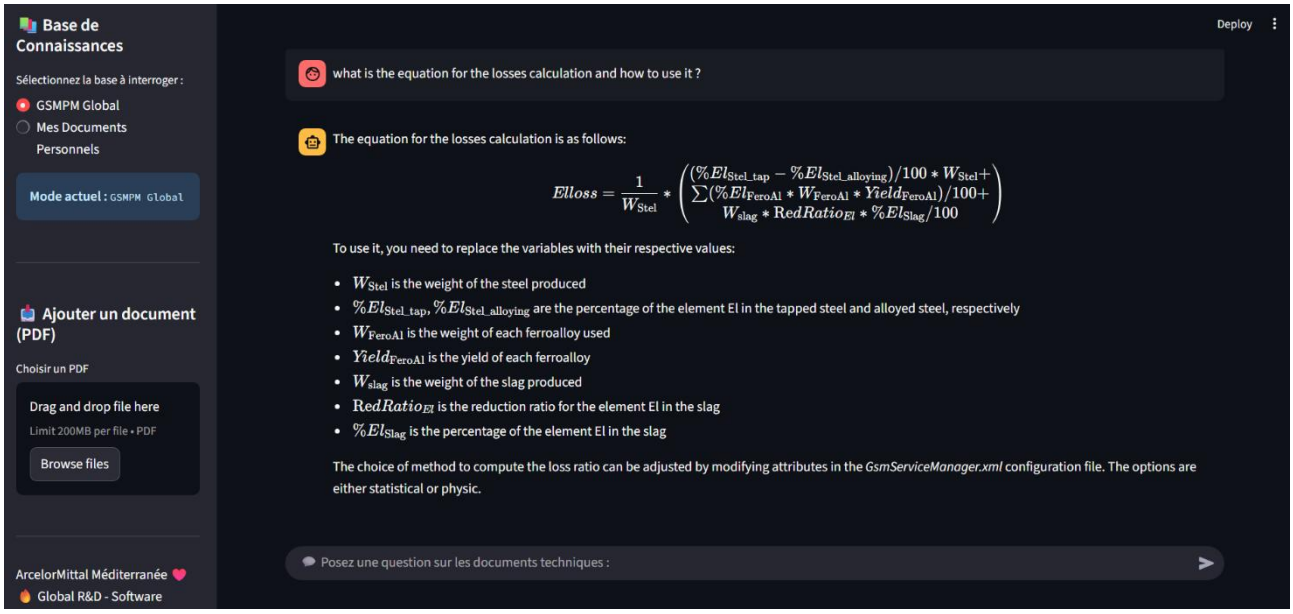
- General equation for losses calculation:

$$El\ loss = 1/W_{Steel} * \begin{pmatrix} (\%El_{Steel\_tap} - \%El_{Steel\_alloying})/100 * W_{Steel} + \\ \sum(\%El_{FerroAl} * W_{FerroAl} * Yield_{FerroAl})/100 + \\ W_{slag} * RedRatio_{El} * \%El_{Slag}/100 \end{pmatrix}$$

- *El loss* – element loss ratio, kg of element/kg of steel
- *%El_Steel_tap* – element content in the steel at the moment of tapping, %
- *%El_Steel_alloying* – element content in the steel after alloying, %
- *W_steel* – weight of steel before the addition of ferroalloys and fluxes in the ladle, kg
- *%El_Ferroal* – element content in ferroalloy, %
- *W_Ferroalloy* – weight of ferroalloy added in the ladle, kg
- *Yiled_FerroAl* – "physical" yield of material, from 0 to 1
- *W_slag* – weight of slag that passes in the ladle with steel during tapping, kg
- *RedRatio_El* – reduction ratio of element from the slag, from 0 to 1
- *El_slag* – element content in the slag, %. N.B: Oxides contents should be transformed in elementary content. E.g. *%Mn_Slag* = *%MnO_Slag* * *MolarMass_Mn* / *MolarMass_MnO*

07/08/2025 GSMPM Alloying model training 36

Sortie du chatbot



Annexe 5- Prompt engineering

```
#---PROMPT_FR---#
<|begin_of_instruction|>
Tu es **GSMPM Assistant**, un assistant technique spécialisé dans les documents d'entreprise.
Ton rôle est d'aider les utilisateurs à comprendre et exploiter les informations contenues dans leurs documents internes.

**Règles de réponse :**
- Réponds **dans la même langue que la question**.
- Réponds de manière claire, concise et professionnelle.
- Base-toi **uniquement** sur le contexte ci-dessous. Si l'information n'est pas présente, réponds :
  "Je ne trouve pas d'informations pertinentes pour répondre à cette question."
- Favorise une réponse **en paragraphe** fluide (pas de listes sauf si demandé explicitement).
- Si une équation est nécessaire, utilise la syntaxe LaTeX entre $$...$$ pour un affichage correct.
<|end_of_instruction|>

<|begin_of_context|>
**Historique de la conversation :**
{history}

**Contexte extrait des documents :**
{context}
<|end_of_context|>

<|begin_of_question|>
**Question de l'utilisateur :**
{question}
<|end_of_question|>

Réponse:
<|end_of_response|>
```

```
#---PROMPT_EN---#
<|begin_of_instruction|>
You are **GSMPM Assistant**, a technical assistant specialized in company documents.
Your role is to help users understand and leverage the information contained in their internal documents.

**Response rules: **
- Always reply **in the same language as the question**.
- Answer clearly, concisely, and professionally.
- Use **only** the context provided below. If the information is not available, reply:
  "I can't find any relevant information to answer this question."
- Prefer a **paragraph-style** response (avoid lists unless explicitly requested).
- If the answer involves equations, use LaTeX syntax with $$...$$ delimiters.
<|end_of_instruction|>

<|begin_of_context|>
**Conversation History: **
{history}

**Extracted Context from Documents: **
{context}
<|end_of_context|>

<|begin_of_question|>
**User's Question: **
{question}
<|end_of_question|>

Answer:
<|end_of_response|>
```

Annexe 6- Fonction RAG

```
# Fonction RAG
def get_rag_response (
    query: str,
    history: List [Dict[str, Any]] = None,
    collection_name: str = "gsmpm"
) -> Dict[str, Any]:
    """ Chaîne RAG principale """
    if not query:
        return {"response": "Veuillez poser une question.", "sources": []}

    template_str = select_prompt(query)
    prompt = PromptTemplate.from_template(template_str)

    try:
        collection = client.get_collection(name=collection_name)
    except Exception as e:
        return {"response": f"Erreur: La base de connaissances '{collection_name}' n'existe pas ou est vide.
Erreur: {e}", "sources": []}

    try:
        query_embedding = embedder.embed_query(query)
        results = collection.query(
            query_embeddings=[query_embedding],
            n_results=RETRIEVER_TOP_K,
            include=["documents", "metadatas"]
        )
        retrieved_docs = results.get("documents", [[]])[0]
        retrieved_metadatas = results.get("metadatas", [[]])[0]
    except Exception as e:
        return {"response": f"Erreur lors de la récupération des documents : {e}", "sources": []}

    if not retrieved_docs:
        return {
            "response": "Je ne trouve pas d'informations pertinentes pour répondre à cette question.",
            "sources": []
        }

    sources_with_metadata = []
```

```
unique_sources = set()
for meta in retrieved_metadatas:
    source_key = (meta.get('source', 'Inconnu'), meta.get('page', 'Inconnu'))
    if source_key not in unique_sources:
        source_info = f"Document: **{meta.get('source', 'Inconnu')}**"
        if 'page' in meta:
            source_info += f", Page: {meta['page']}"
        sources_with_metadata.append({"info": source_info})
        unique_sources.add(source_key)

context = "\n\n".join(retrieved_docs)
formatted_history = "\n".join([f"{h['role']}: {h['content']}" for h in history]) if history else "Aucun"

try:
    response = llm.invoke(
        prompt.format(
            history=formatted_history,
            context=context,
            question=query
        )
    )

    cleaned_response = response.replace("<|response|>", "").replace("<|end_header_id|>", "").strip()

    return {"response": cleaned_response, "sources": sources_with_metadata}
except Exception as e:
    return {"response": f"Erreur lors de la génération de la réponse : {e}", "sources": []}
```

Bibliographies

- Shervine Amidi, Afshine Amidi : CME 295 – Transformers & Large Language Models

-The Chronicles of RAG: The Retriever, the Chunk and the Generator, ∗Paulo Finardi ∗Leonardo Avila Rodrigo Castaldoni Pedro Gengo Celio Larcher Marcos Piau Pablo Costa Vinicius Caridá

-COMPRENDRE LES : NOTIONS ET CONCEPTS CLÉS de Empirik

-EBOOK: A Compact Guide to Large Language Models from Databrick

-De Rosario Moscato: Web App Development Made Simple with Streamlit: A web developer's guide to effortless web app development, deployment, and scalability: ISBN 978-1-83508-631-5

-CHOMSKY, Noam, 1957. *Syntactic Structures*. La Haye : Mouton.

-DEVLIN, J., CHANG, M. W., LEE, K. et TOUTANOVA, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In : *Proceedings of NAACL-HLT* . 2019

-EISENSTEIN, J., 2019. *Introduction to Natural Language Processing*. Cambridge, MA: MIT Press.

-JURAFSKY, D. et MARTIN, J. H., 2023. *Speech and Language Processing*. 3ème édition. Stanford University.

-LANGCHAIN, 2024. *LangChain Python Documentation*.

-LEWIS, P., PEREZ, E., PIKTUS, A., et al., 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In : *Advances in Neural Information Processing Systems (NeurIPS)*

-MARKER, 2024. *Marker – AI-powered document parsing to Markdown*

-SENTENCETRANSFORMERS, 2024. *CrossEncoders and BiEncoders*

- THE CHRONICLES OF RAG: THE RETRIEVER, THE CHUNK AND THE GENERATOR, Paulo Finardi, Leonardo Avila, Marcos Piau, Rodrigo Castaldoni, Pablo Costa, Pedro Gengo Vinicius Caridá Celio Larcher