Université Tunis El Manar
École Nationale d'Ingénieurs de Tunis

Département des Technologies de l'Information et de la Communication

Rapport de projet de fin d'année 1

Réalisé par :

Mohamed Raslen AMDOUNI et Dhiya Eddine EL GABSI

Classe :

1 INFO 2

Agentic-Rag for appreciative Bot Improvement

Supervisé par :
Mme. Asma BAGHDADI

2025 – 2026

Approuvé par :

Mme. Asma BAGHDADI

Encadrante du projet

Signature

Remerciements

Nous tenons à exprimer notre sincère gratitude à Mme Asma Baghdadi, notre encadrante, pour son accompagnement attentif, ses conseils éclairés et sa disponibilité tout au long de ce projet.

Nous remercions également l'École Nationale d'Ingénieurs de Tunis et le Département des Technologies de l'Information et de la Communication pour les ressources et le cadre académique mis à notre disposition.

Enfin, nous adressons notre reconnaissance à nos familles et à nos proches pour leur soutien indéfectible durant cette année.

Résumé

Ce projet présente la conception et l'implémentation d'un chatbot intelligent baptisé *ResilienceBOT*, fondé sur une architecture Agentic-RAG orchestrée via la plateforme n8n. Le système détecte l'état émotionnel de l'utilisateur grâce au modèle `j-hartmann/emotion-english-distilroberta-base`, normalise les 27 émotions brutes en 7 catégories opérationnelles, puis génère des réponses personnalisées ancrées dans un corpus de connaissances en coaching appréciatif, vectorisé dans Supabase. Les contraintes éthiques — non-directivité, orientation vers les forces, bienveillance — sont intégrées directement dans le prompt système. Les tests fonctionnels de bout en bout ont validé le pipeline complet, confirmant la cohérence émotionnelle des réponses et le respect des règles du coaching appréciatif.

Mots-clés : chatbot, Agentic-RAG, détection d'émotions, coaching appréciatif, n8n, LLM, Supabase, prompt engineering.

Table des matières

Table des figures

Introduction Générale

L'essor fulgurant de l'intelligence artificielle et des grands modèles de langage (LLMs) a profondément transformé l'interaction entre les systèmes informatiques et les humains. Dans ce contexte, les chatbots ont évolué de simples systèmes à règles vers des agents capables de générer des réponses nuancées et contextualisées. Pourtant, la plupart restent limités à des réponses génériques, dépourvues de sensibilité émotionnelle et incapables de s'adapter aux besoins psychologiques profonds des utilisateurs.

C'est dans ce cadre que s'inscrit notre projet de fin d'études, réalisé à l'École Nationale d'Ingénieurs de Tunis. L'objectif est de développer un chatbot intelligent nommé Appreciative Bot, capable de détecter l'humeur et les besoins émotionnels de l'utilisateur, de mobiliser les principes de la psychologie positive, et de proposer des réponses personnalisées, non directives et orientées vers les forces, tout en assurant cohérence, sécurité et qualité via un mécanisme de vérification agentique.

Pour cela, notre solution repose sur une architecture Agentic-RAG combinant plusieurs agents spécialisés : un agent de détection d'émotions, un agent de génération de réponses basé sur un corpus de coaching appréciatif, et un mécanisme de vérification garantissant la qualité et le respect éthique. Le pipeline est orchestré avec n8n, en s'appuyant sur Groq, Supabase, HuggingFace et Google Drive.

Ce rapport s'articule en cinq chapitres progressifs. Le premier, « Cadre Général du Projet », introduit la problématique, analyse l'existant, présente la solution et la méthodologie. Le deuxième, « État de l'Art », pose les bases théoriques. Le troisième, « Analyse des Besoins », identifie les exigences fonctionnelles, non fonctionnelles et éthiques. Le quatrième, « Conception et Implémentation », détaille l'architecture Agentic-RAG et la réalisation technique. Le cinquième, « Tests et Évaluation », valide le système via des scénarios fonctionnels, émotionnels et éthiques, avant de conclure par un bilan et des perspectives.

Chapitre 1

Cadre Général du Projet

1.1 Introduction

Ce chapitre présente le cadre général de notre projet. Nous commençons par exposer la problématique à laquelle nous avons été confrontés, avant d'analyser les solutions existantes et leurs limites. Nous présentons ensuite la solution envisagée ainsi que la méthodologie adoptée pour sa mise en œuvre.

1.2 Contexte generale

L'intelligence artificielle connaît depuis ces dernières années une évolution sans précédent, transformant en profondeur la manière dont les humains interagissent avec les systèmes informatiques. L'émergence des modèles de langage de grande taille (LLMs) et leur intégration croissante dans des agents conversationnels ont ouvert de nouvelles perspectives dans des domaines aussi variés que l'éducation, la santé, le service client et le développement personnel. Selon les prévisions de Gartner, plus de 33% des applications logicielles d'entreprise intégreront des fonctionnalités d'IA agentique d'ici 2028, signalant une transition profonde vers des systèmes capables d'agir de manière autonome et adaptative .

Parallèlement à cet essor technologique, la question du bien-être mental et émotionnel occupe une place croissante dans les préoccupations sociétales. La demande mondiale en services de coaching et d'accompagnement psychologique dépasse largement l'offre disponible en professionnels qualifiés, créant un déficit d'accès au soutien émotionnel pour une large partie de la population. Des études récentes montrent que plus de 75% des personnes souffrant de détresse psychologique ne reçoivent aucun soutien professionnel, faute de disponibilité, d'accessibilité géographique ou de moyens financiers . Dans ce contexte, les technologies d'IA conversationnelle représentent une opportunité réelle de démocratiser l'accès au coaching et à l'accompagnement émotionnel, en offrant une disponibilité

permanente, une absence de jugement et une personnalisation des échanges.

1.3 Problématique

Avec la démocratisation des assistants conversationnels, de nombreux utilisateurs se tournent aujourd'hui vers ces outils pour obtenir un soutien moral, des conseils de développement personnel ou simplement pour être écoutés. Or, les chatbots classiques, même ceux fondés sur des modèles de langage de grande taille, souffrent d'une limite fondamentale : ils traitent chaque message de manière uniforme, sans tenir compte de l'état émotionnel de l'interlocuteur ni des principes du coaching appréciatif. En effet, si les LLMs modélisent des éléments d'empathie cognitive, il leur manque encore la capacité d'adapter leurs réponses de manière fine et personnalisée à l'état émotionnel réel de chaque utilisateur [1].

Cette absence de sensibilité émotionnelle engendre des réponses génériques, parfois inadaptées voire contre-productives dans un contexte de soutien psychologique. Des études ont montré que les chatbots de bien-être mental reposent encore largement sur des flux à base de règles prédéfinies, conduisant à des interactions génériques insuffisamment personnalisées [2]. Par ailleurs, les systèmes actuels ne disposent d'aucun mécanisme permettant de garantir que leurs réponses respectent des règles éthiques précises, telles que la non-directivité, l'orientation vers les forces ou la bienveillance.

C'est dans ce contexte que s'inscrit notre projet. La question centrale à laquelle nous cherchons à répondre est la suivante : comment concevoir un chatbot intelligent capable de détecter l'humeur et les besoins émotionnels de l'utilisateur, de mobiliser des connaissances issues de la psychologie positive, et de proposer des réponses personnalisées, non directives et orientées vers les forces, tout en garantissant cohérence, sécurité et qualité grâce à un mécanisme de vérification agentique ?

1.4 Étude de l'Existant

Plusieurs solutions conversationnelles ont été développées ces dernières années dans le domaine du bien-être et du soutien émotionnel. L'analyse de ces systèmes nous a permis de mieux comprendre leurs apports et leurs limites, et de positionner notre projet par rapport à l'état de l'art.

FIGURE 1.1 – Interface de Woebot [49]

Woebot est l'un des chatbots thérapeutiques les plus connus. Fondé sur les principes de la thérapie cognitivo-comportementale (TCC), il a été utilisé par plus d'un million et demi de personnes à travers le monde et a démontré des résultats mesurables dans la réduction des symptômes dépressifs chez les jeunes adultes [3].

Toutefois, Woebot repose sur des dialogues pré-scriptés et un système à base de règles, et non sur un modèle génératif. Ses réponses sont donc limitées à des scénarios anticipés, ce qui réduit considérablement sa capacité d'adaptation à des situations complexes ou imprévues [4].

Cette rigidité constitue l'une des raisons majeures de son arrêt en tant qu'application grand public en juin 2025.

Les chatbots généralistes basés sur des LLMs, tels que ChatGPT, offrent quant à eux une grande flexibilité dans la génération de réponses. Cependant, bien qu'ils exhibent des éléments d'empathie cognitive, il leur manque une base de connaissances spécialisée en coaching appréciatif et un mécanisme explicite de détection émotionnelle. Des recherches ont montré que ces systèmes peuvent produire des réponses manquant d'empathie authentique, de personnalisation et de nuance culturelle, offrant parfois des conseils génériques voire potentiellement inadaptés [5].

Ils ne disposent d'aucun agent de vérification garantissant la conformité éthique de leurs réponses.

1.5 Solution Proposée

Pour répondre à la problématique identifiée, nous avons conçu un chatbot intelligent reposant sur une architecture Agentic-RAG, orchestrée via la plateforme n8n. Cette solution s'articule autour de trois composants principaux qui s'enchaînent de manière fluide à chaque interaction.

Lorsqu'un utilisateur envoie un message, celui-ci est d'abord acheminé vers un agent de détection d'émotions, fondé sur le modèle pré-entraîné `j-hartmann/emotion-english-distilroberta-base` de HuggingFace, entraîné sur le dataset GoEmotions. Ce modèle est capable de reconnaître 28 catégories émotionnelles distinctes, allant de la joie

et la gratitude jusqu'à la tristesse, la peur ou la colère [6].

Le résultat de cette détection est ensuite normalisé et intégré au contexte du message avant d'être transmis à l'agent suivant.

Le message enrichi de son contexte émotionnel est alors pris en charge par un agent RAG agentique, qui interroge une base de connaissances vectorielle hébergée sur Supabase, constituée de livres et d'articles spécialisés en coaching appréciatif et en psychologie positive. Le coaching appréciatif, introduit par Cooperrider et Srivastva en 1987, est une approche fondée sur la découverte des forces et des possibilités plutôt que sur la résolution des problèmes, visant à orienter l'individu vers ce qui fonctionne bien plutôt que vers ce qui est défaillant [7].

Les documents du corpus sont ensuite vectorisés à l'aide d'embeddings multilingues de HuggingFace. La réponse est générée par le modèle Groq, en s'appuyant à la fois sur les passages récupérés et sur l'historique conversationnel maintenu par une mémoire tampon.

Enfin, nous avons choisi d'intégrer les exigences du coaching appréciatif directement dans le prompt de génération de la réponse (RAG), orchestré via la plateforme n8n. Grâce à une technique d'ingénierie de prompt (prompt engineering), ce prompt guide le modèle de génération pour qu'il produise une réponse qui respecte intrinsèquement les règles de non-directivité, d'orientation vers les forces, de bienveillance et de cohérence avec l'état émotionnel détecté. Cette approche préventive par l'ingénierie de prompt constitue l'une des contributions majeures de notre projet. Elle permet de garantir la fiabilité éthique des réponses produites par les systèmes d'IA dans des contextes sensibles [8], en intégrant ces contraintes dès la phase de création, plutôt que par une vérification a posteriori.

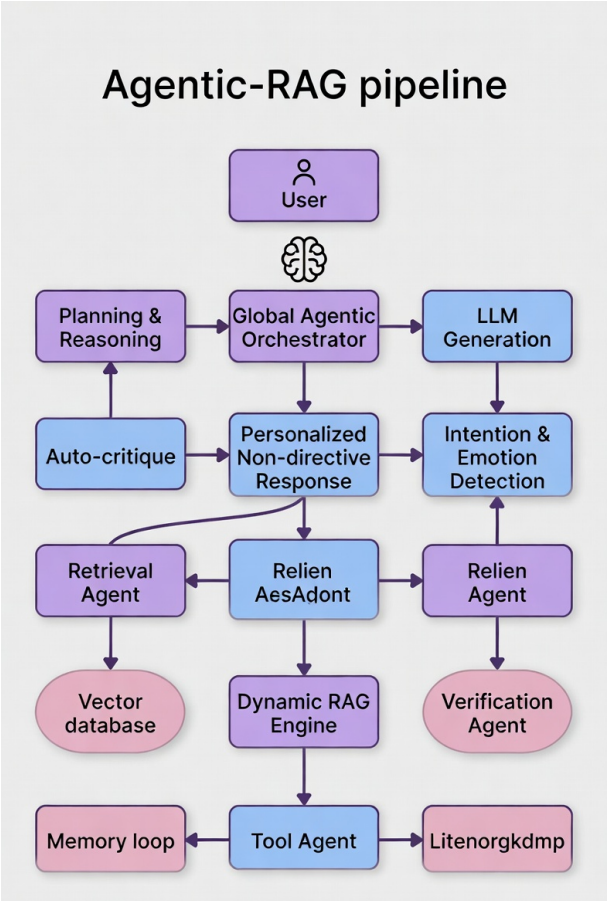

FIGURE 1.2 – Architecture globale du pipeline Agentic-RAG [50]

1.6 Conclusion

Ce chapitre nous a permis de dresser un aperçu global du projet en exposant la problématique à résoudre, en analysant les solutions existantes et leurs limites, et en présentant l'architecture générale de notre solution ainsi que la méthodologie adoptée. Le chapitre suivant approfondira les concepts théoriques sur lesquels repose notre approche, afin d'en mieux cerner les fondements techniques et scientifiques.

Chapitre 2

État de l'Art

2.1 Introduction

Ce chapitre présente les fondements théoriques et technologiques sur lesquels repose notre solution. Nous abordons successivement les concepts de machine learning et deep learning qui constituent le socle de l'intelligence artificielle moderne, les modèles de langage de grande taille et les techniques de fine-tuning et de prompt engineering associées, le paradigme RAG et son architecture, son évolution vers une approche agentique, la détection d'émotions, les embeddings multilingues et les bases de données vectorielles, la plateforme d'orchestration n8n, et enfin les technologies front-end utilisées pour l'interface web. L'ensemble de ces concepts constitue le socle indispensable à la compréhension des choix techniques réalisés tout au long de ce projet.

2.2 Machine Learning et Deep Learning

2.2.1 Machine Learning

Le machine learning (ML) est un sous-domaine de l'intelligence artificielle qui repose sur l'idée fondamentale de permettre aux machines d'apprendre à partir des données, sans avoir à être explicitement programmées pour chaque tâche. Comme le définit IBM, le machine learning implique « l'application d'algorithmes pour automatiser les processus de prise de décision à l'aide de modèles entraînés sur des données plutôt que manuellement programmés » [9]. On distingue trois grands paradigmes d'apprentissage : l'apprentissage supervisé, où le modèle apprend à partir de données labellisées ; l'apprentissage non supervisé, où il découvre des structures cachées dans des données non étiquetées ; et l'apprentissage par renforcement, où un agent apprend par essai-erreur en interagissant avec un environnement.

2.2.2 Deep Learning et Réseaux de Neurones

Le deep learning est un sous-ensemble du machine learning fondé sur des réseaux de neurones artificiels à multiples couches, dont l'architecture s'inspire du fonctionnement du cerveau humain. Comme le précise Wikipedia, « le deep learning se concentre sur l'utilisation de réseaux de neurones multicouches pour effectuer des tâches telles que la classification, la régression et l'apprentissage de représentations » [10]. Le terme « deep » désigne la profondeur de ces réseaux : un modèle de deep learning possède généralement plus de trois couches cachées, chacune extrayant des caractéristiques de plus en plus abstraites des données d'entrée [11].

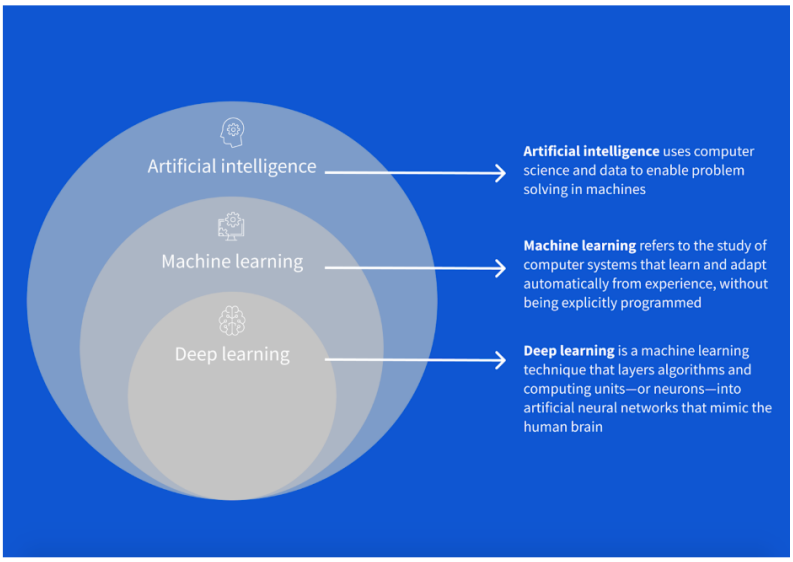

FIGURE 2.1 – Hiérarchie : Intelligence Artificielle → Machine Learning → Deep Learning [51]

Dans le contexte de notre projet, le deep learning intervient à deux niveaux essentiels. D'une part, le modèle de détection d'émotions `j-hartmann/emotion-english-distilroberta-base` est un réseau de neurones profond de type Transformer, pré-entraîné sur des millions de textes. D'autre part, le modèle de langage Llama-3.3-70b utilisé via Groq est lui-même un réseau de neurones profond comptant 70 milliards de paramètres, dont les représentations internes sont le fruit d'un entraînement intensif sur des corpus massifs [12].

2.3 Les Modèles de Langage de Grande Taille (LLMs)

Les modèles de langage de grande taille (LLMs) sont des systèmes d'intelligence artificielle entraînés sur des corpus textuels massifs, capables de comprendre et de générer du langage naturel avec un niveau de cohérence remarquable. Ils reposent sur l'architecture Transformer introduite par Vaswani et al. en 2017, qui a révolutionné le traitement automatique du langage naturel grâce au mécanisme d'attention permettant de modéliser les

dépendances à longue distance entre les tokens d'un texte [13]. Malgré leurs performances impressionnantes, les LLMs souffrent de limitations majeures : leurs connaissances sont figées à la date de leur entraînement, et ils peuvent produire des informations erronées — phénomène connu sous le nom d'hallucination [13].

Dans notre projet, nous utilisons Groq comme moteur d'inférence pour le modèle `llama-3.3-70b-versatile`. Groq est une entreprise spécialisée dans l'accélération de l'inférence, dont le composant phare est le Language Processing Unit (LPU), une puce conçue spécifiquement pour l'inférence des LLMs. Son architecture entièrement déterministe lui permet d'éliminer les sources de latence imprévisible, atteignant jusqu'à 300 tokens par seconde pour des modèles de grande taille [14]. Cette performance est particulièrement adaptée à notre contexte conversationnel, où la fluidité et la réactivité des réponses sont essentielles à l'expérience utilisateur.

2.4 Fine-tuning et Transfer Learning

2.4.1 Le Transfer Learning

Le transfer learning est une technique fondamentale du machine learning qui consiste à réutiliser les connaissances acquises par un modèle pré-entraîné sur une tâche source, afin de les appliquer efficacement à une nouvelle tâche cible. Comme le définit IBM, « le transfer learning permet de réduire considérablement la quantité de puissance de calcul et de données labellisées nécessaires pour obtenir des modèles adaptés à des cas d'usage spécialisés » [15]. L'idée centrale est qu'un modèle ayant appris des représentations générales du langage sur des milliards de tokens peut être réutilisé comme point de départ solide pour des tâches spécifiques, plutôt que d'entraîner un nouveau modèle depuis zéro.

2.4.2 Le Fine-tuning des LLMs

Le fine-tuning est une technique spécifique du transfer learning qui consiste à poursuivre l'entraînement d'un modèle pré-entraîné sur un jeu de données plus petit et spécialisé, afin de l'adapter à un domaine ou une tâche précise. DataCamp le définit comme « le processus consistant à prendre un modèle pré-entraîné et à le ré-entraîner davantage sur un jeu de données spécifique à un domaine » [16].

Cette méthode permet d'obtenir de bons résultats sur des tâches spécialisées sans avoir à entraîner un modèle depuis zéro, ce qui serait trop coûteux.

Dans notre projet, le fine-tuning est au cœur du modèle de détection d'émotions que nous utilisons. En effet, `j-hartmann/emotion-english-distilroberta-base` est un checkpoint fine-tuné à partir du modèle pré-entraîné DistilRoBERTa-base. Ce choix nous a permis de bénéficier d'un modèle performant sans entraînement supplémentaire, conformément aux bonnes pratiques du transfer learning [17].

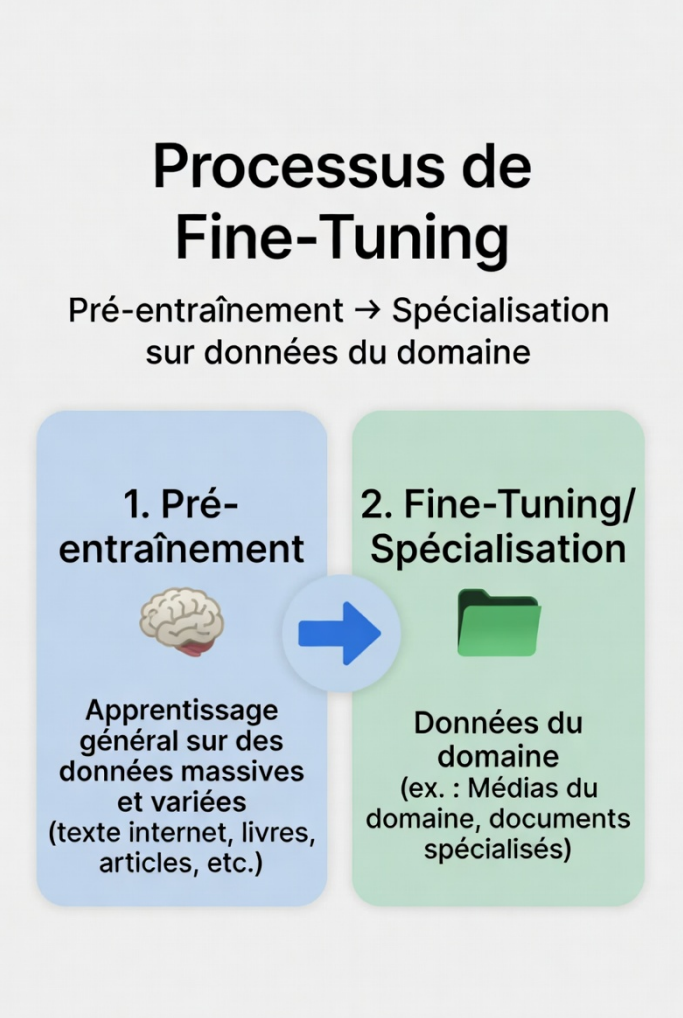

FIGURE 2.2 – Le processus de fine-tuning d'un LLM [52]

2.5 Le Prompt Engineering

Le prompt engineering est une discipline émergente qui consiste à concevoir et optimiser les instructions fournies aux LLMs afin de guider leurs sorties vers les résultats souhaités. Comme le définit Wikipedia, « le prompt engineering est le processus de structuration des entrées en langage naturel pour produire des sorties spécifiées d'un modèle d'IA générative » [18]. Une enquête systématique publiée en 2024 a identifié plus de 50 techniques distinctes de prompting textuel, soulignant la diversité et la richesse de ce domaine [19].

Parmi les techniques les plus importantes, on distingue le zero-shot prompting (demander directement sans exemple), le few-shot prompting (fournir des exemples dans le prompt), et le chain-of-thought prompting (demander au modèle de raisonner étape par étape avant de répondre). Cette dernière technique, proposée par des chercheurs de Google en 2022, a démontré une amélioration significative des capacités de raisonnement des LLMs sur des tâches complexes [20].

Dans notre projet, le prompt engineering joue un rôle central dans le comportement de l'agent RAG et de l'agent de vérification. Le prompt système de l'agent RAG lui impose d'utiliser obligatoirement l'outil de recherche Supabase avant toute génération de réponse, et de formuler ses réponses conformément aux principes du coaching appréciatif. L'agent de vérification est lui-même guidé par un prompt définissant précisément les critères éthiques à respecter : non-directivité, orientation vers les forces et bienveillance.

2.6 Du RAG Classique au RAG Agentique

2.6.1 Architecture du RAG Classique

La Retrieval-Augmented Generation (RAG) est un paradigme introduit en 2020 par Lewis et al., qui vise à enrichir la génération de texte par les LLMs en leur donnant accès à une base de connaissances externe. IBM le décrit comme « une architecture pour optimiser la performance d'un modèle d'IA en le connectant à des bases de connaissances externes » [21]. L'architecture du pipeline RAG classique s'articule autour de trois phases distinctes et séquentielles comme le montre la Figure 2.3.

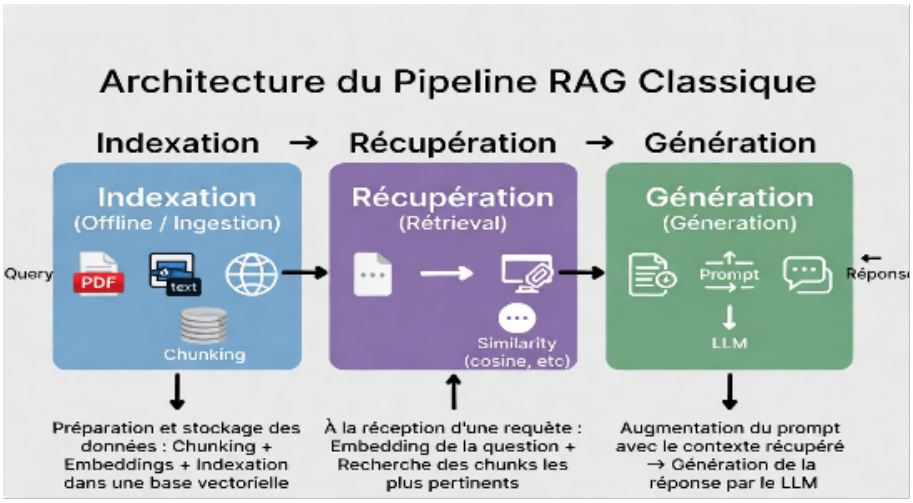

FIGURE 2.3 – Architecture du pipeline RAG classique (Indexation → Récupération → Génération) [53]

La première phase, dite d'indexation, consiste à découper les documents du corpus en fragments (chunks), à les transformer en vecteurs d'embeddings, puis à les stocker dans une base vectorielle. La deuxième phase, dite de récupération, consiste à encoder la requête de l'utilisateur sous forme vectorielle et à calculer sa similarité avec les chunks indexés afin de récupérer les passages les plus pertinents. La troisième phase, dite de génération, le contexte récupéré est ajouté au prompt du LLM, qui génère alors une réponse précise, fondée et moins sujette aux hallucinations [22].

Cette approche présente cependant des limites importantes : elle repose sur une récupération unique en un seul passage (one-shot), sans possibilité de raffiner la requête, et ne

dispose d'aucun mécanisme de validation des résultats récupérés, pouvant ainsi produire des réponses incomplètes face à des questions complexes [23].

2.6.2 Le RAG Agentique

Pour pallier ces limitations, le RAG agentique introduit des agents autonomes au sein du pipeline. Contrairement au RAG classique, le RAG agentique est capable de planifier, d'itérer, d'utiliser des outils externes et de valider ses propres résultats avant de produire une réponse finale. IBM le définit comme « le passage d'une interrogation statique basée sur des règles à une résolution de problèmes adaptative et intelligente » [24]. Le tableau suivant synthétise les différences entre les deux approches :

Critère	**RAG Classique**	**RAG Agentique**
Récupération	Un seul passage	Itérative et adaptative
Validation des résultats	Absente	Assurée par un agent dédié
Gestion de la complexité	Requêtes simples	Requêtes multi-étapes
Sources de données	Une seule base vectorielle	Multiples sources et outils
Adaptabilité	Statique	Dynamique

TABLE 2.1 – Comparaison RAG Classique vs RAG Agentique

2.7 La Détection d'Émotions

La détection automatique d'émotions dans le texte est un sous-domaine du traitement automatique du langage naturel (NLP) qui vise à identifier l'état affectif exprimé par un locuteur à partir de ses écrits. Cette tâche est fondamentalement plus complexe que l'analyse de sentiment classique (positive/négative/neutre), car elle cherche à discriminer des états émotionnels fins et parfois subtils [25].

Dans notre système, nous utilisons le modèle pré-entraîné `j-hartmann/emotion-english-distilroberta-base`, disponible sur HuggingFace. Il s'agit d'un checkpoint fine-tuné de DistilRoBERTa-base, entraîné sur six datasets couvrant des textes issus de Twitter, Reddit, de journaux d'étudiants et de dialogues télévisés, représentant près de 20 000 observations équilibrées. Ce modèle est capable de reconnaître 7 catégories émotionnelles — anger, disgust, fear, joy, neutral, sadness, surprise — et a été conçu pour être léger, rapide et facilement intégrable dans des pipelines de production. Comme les résultats initiaux n'étaient pas satisfaisants lors de l'évaluation, nous avons par conséquent implémenté une version du modèle directement depuis HuggingFace, à savoir `j-hartmann/emotion-english-distilroberta-base`, fine-tuné sur le dataset GoEmotions de Google, qui regroupe plus de 58 000 commentaires Reddit annotés manuellement en 27 catégories émotionnelles distinctes : admiration, amusement, anger, annoyance, approval, caring, confusion, curiosity, desire, disappointment, disapproval, disgust, embarrassment, excite-

ment, fear, gratitude, grief, joy, love, nervousness, neutral, nostalgia, optimism, pride, relief, remorse, sadness, surprise et trust [26]. Ce niveau de granularité émotionnelle est bien supérieur aux approches classiques d'analyse de sentiment qui se limitent à trois catégories (positive, négative, neutre), ce qui en fait un outil particulièrement adapté à un contexte de coaching appréciatif où la nuance émotionnelle est essentielle.

Cependant, l'exploitation directe de ces 27 catégories dans notre pipeline soulève trois problèmes concrets qui ont motivé l'introduction d'une étape de normalisation. Premièrement, plusieurs émotions présentent un fort chevauchement sémantique : annoyance et anger, par exemple, appellent le même style de réponse apaisante, tout comme joy, excitement et amusement méritent toutes une réponse chaleureuse et positive. Maintenir 27 styles de réponses distincts serait donc redondant et inutilement complexe. Deuxièmement, définir un style de réponse précis et non ambigu pour chacune des 27 émotions dans le prompt système de l'agent RAG représente une tâche complexe, avec un risque élevé de contradictions et d'incohérences dans le comportement du bot. Troisièmement, pour des émotions rares ou très spécifiques telles que nostalgia, remorse ou pride, le modèle dispose de moins de données d'entraînement, ce qui se traduit par des scores de confiance (confidence scores) plus faibles et donc un risque d'instabilité en production [27].

Pour remédier à ces limitations, nous avons implémenté dans n8n un nœud de normalisation qui regroupe les 27 émotions brutes en un ensemble réduit de 7 catégories opérationnelles : joy, sadness, anger, neutral, fear, surprise et love, chacune associée à un style de réponse clair et défini — respectivement positif et chaleureux, empathique et rassurant, calme et apaisant, professionnel, rassurant et encourageant, enthousiaste, et chaleureux et bienveillant [28].

Ce regroupement rend le système plus robuste, plus maintenable et garantit une cohérence permanente entre l'émotion détectée et le ton de la réponse générée.

2.8 Les Embeddings Multilingues et les Bases Vectorielles

2.8.1 Les Embeddings Multilingues

Les embeddings sont des représentations numériques de textes sous forme de vecteurs denses dans un espace de haute dimensionnalité, où des textes sémantiquement proches se trouvent géométriquement proches l'un de l'autre. Les modèles d'embeddings multilingues étendent ce principe à plusieurs langues simultanément, permettant de rapprocher des textes sémantiquement équivalents même s'ils sont rédigés dans des langues différentes [29].

Dans notre projet, nous utilisons le modèle `paraphrase-multilingual-MiniLM-L12-v2` de HuggingFace Inference, basé sur l'architecture Sentence-BERT introduite par Reimers

et Gurevych (2019). Ce modèle projette phrases et paragraphes dans un espace vectoriel dense de 384 dimensions, couvrant plus de 50 langues [30]. Ce choix garantit que notre base de connaissances en coaching appréciatif reste accessible quelle que soit la langue dans laquelle l'utilisateur s'exprime.

2.8.2 La Base de Données Vectorielle : Supabase

Supabase est une plateforme open-source bâtie sur PostgreSQL, qui intègre nativement l'extension pgvector pour le stockage et la recherche de vecteurs d'embeddings. Elle permet d'effectuer des recherches sémantiques par similarité cosinus directement au sein de la base de données relationnelle, sans nécessiter de service externe supplémentaire [31]. Dans notre pipeline, chaque document du corpus est découpé en chunks, vectorisé via les embeddings multilingues HuggingFace, puis stocké dans une table `documents` de Supabase, avec ses métadonnées (`fileId`, `fileName`) permettant la gestion du cycle de vie des documents [28].

2.9 L'Automatisation et l'Orchestration de Workflows : n8n

n8n est une plateforme d'automatisation de workflows open-source qui combine des capacités d'IA avec l'automatisation des processus métier. Comme le décrit la documentation officielle, n8n « aide à connecter n'importe quelle application dotée d'une API avec n'importe quelle autre, en manipulant ses données avec peu ou pas de code » [32]. Son architecture repose sur un système de nœuds (nodes) visuels, chacun représentant une action ou une intégration, que l'utilisateur connecte graphiquement pour construire des workflows automatisés complexes. Une étude académique publiée dans l'International Journal of Computer Applications confirme que n8n démontre « une scalabilité linéaire, une fiabilité supérieure à 98% et de solides capacités d'orchestration de l'IA » sous des charges de production [33].

Dans notre projet, n8n joue le rôle d'orchestrateur central de l'ensemble du pipeline Agentic-RAG. Il gère trois flux distincts : le pipeline de chat qui enchaîne la détection d'émotions, l'interrogation RAG et la génération de réponse ; le pipeline d'ingestion des documents depuis Google Drive vers Supabase ; et le pipeline de suppression et de mise à jour des vecteurs en cas de modification du corpus. La Figure 2.4 illustre l'architecture d'un workflow n8n typique.

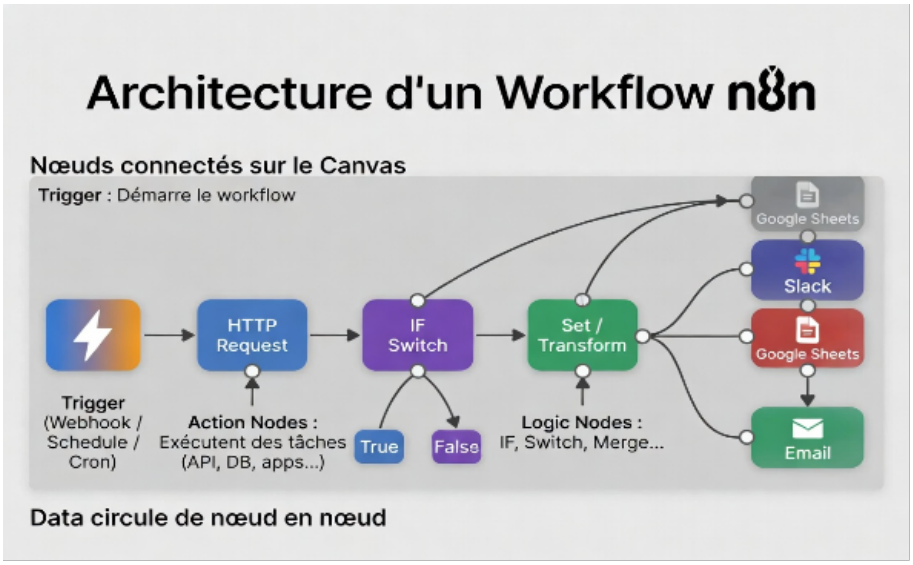

FIGURE 2.4 – Architecture d'un workflow n8n avec nœuds et connexions [54]

2.10 Synthèse et Positionnement du Projet

L'analyse des différents concepts présentés dans ce chapitre révèle que notre projet se situe à la confluence de plusieurs domaines complémentaires : le machine learning et le deep learning pour le fondement théorique des modèles utilisés, le fine-tuning et le prompt engineering pour leur adaptation à notre contexte spécifique, le RAG agentique pour la récupération contextuelle de connaissances spécialisées, la détection d'émotions pour la personnalisation de l'interaction, et n8n pour l'orchestration de l'ensemble du pipeline. Aucune solution existante ne combine toutes ces dimensions de manière intégrée et vérifiée éthiquement, ce qui constitue la contribution principale de notre travail.

Chapitre 3

Analyse des Besoins

3.1 Introduction

Ce chapitre est consacré à l'analyse et à la spécification des besoins de notre système. Nous identifions les acteurs du système, ses exigences fonctionnelles et non fonctionnelles, ses cas d'utilisation, ainsi que les contraintes éthiques encadrant le comportement du bot dans un contexte de coaching appréciatif. Ces spécifications constituent le référentiel à partir duquel l'architecture a été conçue et sera évaluée.

3.2 Identification des Acteurs

Notre système implique deux acteurs principaux dont les rôles sont clairement distincts.

L'utilisateur final est la personne qui interagit avec le chatbot via l'interface conversationnelle développée en React. Il soumet des messages en langage naturel, dans une ou plusieurs langues, et attend en retour des réponses personnalisées, bienveillantes et adaptées à son état émotionnel. Il n'a aucune connaissance technique du système sous-jacent et interagit uniquement à travers l'interface de chat exposée par n8n.

L'administrateur du corpus est la personne responsable de la gestion de la base de connaissances du système. Il alimente, met à jour et supprime les documents de coaching appréciatif stockés dans un dossier Google Drive dédié, déclenchant ainsi automatiquement les pipelines d'ingestion et de mise à jour vectorielle dans Supabase. Ce rôle ne nécessite aucune intervention directe sur le pipeline technique, ce qui constitue l'un des avantages majeurs de l'architecture choisie [34].

3.3 Besoins Fonctionnels

Les besoins fonctionnels décrivent l'ensemble des fonctionnalités que le système doit être capable d'accomplir pour répondre aux objectifs fixés dans le cahier des charges. La spécification rigoureuse de ces besoins constitue une étape fondamentale dans le développement de tout système conversationnel basé sur des LLMs, car elle conditionne directement les choix architecturaux et les critères d'évaluation [35].

BF1 — Réception et traitement des messages utilisateur : Le système doit être capable de recevoir des messages en langage naturel de la part de l'utilisateur via l'interface conversationnelle, quelle que soit la langue utilisée, et de les préparer pour les étapes de traitement suivantes au sein du pipeline n8n.

BF2 — Détection de l'émotion de l'utilisateur : À chaque message reçu, le système doit analyser le contenu textuel afin d'identifier l'état émotionnel de l'utilisateur parmi les 27 catégories émotionnelles du dataset GoEmotions, en s'appuyant sur le modèle `j-hartmann/emotion-english-distilroberta-base` [36].

BF3 — Normalisation de l'émotion détectée : Compte tenu du chevauchement sémantique entre plusieurs des 27 émotions brutes et de la nécessité de définir des styles de réponse clairs et non ambigus, le système doit normaliser le résultat brut de la détection en le regroupant en 7 catégories opérationnelles — joy, sadness, anger, neutral, fear, surprise et love — chacune associée à un style de réponse adapté au contexte du coaching appréciatif [28]. Ce résultat normalisé est ensuite injecté dans le contexte du message avant transmission à l'agent de génération.

BF4 — Récupération contextuelle depuis la base de connaissances : L'agent RAG doit obligatoirement interroger la base vectorielle Supabase avant toute génération de réponse, en utilisant les embeddings multilingues `paraphrase-multilingual-MiniLM-L12-v2` pour récupérer les passages les plus pertinents issus du corpus de coaching appréciatif. Cette obligation est explicitement inscrite dans la description de l'outil Supabase configuré dans n8n : « À utiliser obligatoirement avant toute réponse ».

BF5 — Génération de réponses personnalisées : Le système doit générer des réponses cohérentes, contextualisées et adaptées à l'émotion normalisée, en s'appuyant sur le modèle `llama-3.3-70b-versatile` via Groq, tout en maintenant un historique conversationnel via une mémoire tampon de fenêtre glissante (Simple Memory) garantissant la continuité du dialogue.

BF6 — Vérification agentique des réponses : Avant d'être restituée à l'utilisateur, chaque réponse générée doit être soumise à un agent de vérification qui s'assure de sa conformité avec les règles du coaching appréciatif : non-directivité, orientation vers les forces et bienveillance. Cet agent constitue l'une des contributions majeures de notre projet, répondant à un besoin clairement identifié dans le cahier des charges [37].

BF7 — Gestion dynamique du corpus : Le système doit détecter automatique-

ment tout ajout, modification ou suppression de document dans le dossier Google Drive dédié, et mettre à jour la base vectorielle Supabase en conséquence. En cas de modification, les anciens vecteurs associés au fichier modifié doivent être supprimés — via un filtre sur le `fileId` stocké dans les métadonnées — avant l'insertion des nouveaux vecteurs, garantissant ainsi la cohérence et la fraîcheur du corpus [28].

BF8 — Interface web interactive : Le système doit proposer une interface utilisateur moderne, développée en React, Three.js et CSS, permettant à l'utilisateur d'interagir de manière fluide et immersive avec le chatbot depuis un navigateur web standard.

3.4 Besoins Non Fonctionnels

Les besoins non fonctionnels définissent les contraintes de qualité, de performance et de fiabilité que le système doit respecter, indépendamment des fonctionnalités qu'il offre. Ces exigences sont déterminantes pour garantir une expérience utilisateur satisfaisante et un comportement système prévisible [38].

BNF1 — Performance et réactivité : Le système doit produire des réponses dans un délai raisonnable pour maintenir la fluidité de la conversation. L'utilisation de Groq comme moteur d'inférence, capable d'atteindre jusqu'à 300 tokens par seconde grâce à son architecture LPU, contribue directement à satisfaire cette exigence [14].

BNF2 — Multilingualité : Le système doit être capable de traiter et de répondre aux messages de l'utilisateur quelle que soit la langue employée. Cette exigence est assurée par l'utilisation d'embeddings multilingues couvrant plus de 50 langues pour la vectorisation du corpus et la recherche sémantique, conformément aux capacités du modèle `paraphrase-multilingual-MiniLM-L12-v2` [30].

BNF3 — Robustesse de la détection émotionnelle : La normalisation des 27 émotions brutes vers 7 catégories opérationnelles doit garantir la stabilité du système face aux émotions rares ou peu représentées dans les données d'entraînement, pour lesquelles le modèle peut retourner des scores de confiance faibles. Toute émotion non reconnue doit être automatiquement redirigée vers la catégorie neutral afin d'éviter tout comportement imprévisible du pipeline [27].

BNF4 — Maintenabilité du corpus : La base de connaissances doit pouvoir être mise à jour sans interruption de service et sans intervention technique directe sur le pipeline, grâce aux triggers automatiques connectés à Google Drive.

BNF5 — Modularité de l'architecture : Chaque composant du système — détection émotionnelle, normalisation, agent RAG, agent de vérification, gestion du corpus — doit être indépendant et remplaçable sans affecter le fonctionnement global du pipeline, conformément aux bonnes pratiques de conception des systèmes multi-agents [24].

BNF6 — Fiabilité et ancrage documentaire des réponses : Le système doit garantir que chaque réponse produite est ancrée dans le corpus de connaissances et ne

repose pas uniquement sur les paramètres internes du LLM, réduisant ainsi le risque d'hallucination. Le prompt système de l'agent RAG doit imposer explicitement l'utilisation de l'outil de recherche Supabase avant toute génération, conformément aux techniques de prompt engineering appliquées dans notre projet [19].

3.5 Contraintes Éthiques et Règles du Coaching Appréciatif

La définition de contraintes éthiques explicites constitue l'une des spécificités majeures de notre projet par rapport aux chatbots généralistes. Des études récentes ont mis en évidence la nécessité de doter les systèmes d'IA conversationnels de mécanismes de vérification garantissant leur alignement avec des valeurs humaines précises, particulièrement dans des contextes sensibles tels que le bien-être et l'accompagnement psychologique [8].

Les contraintes éthiques que notre agent de vérification doit faire respecter sont les suivantes.

Le **principe de non-directivité** impose que le bot n'impose jamais de décision, de jugement ou de directive à l'utilisateur : il doit accompagner, suggérer et ouvrir des perspectives sans prescrire.

Le **principe d'orientation vers les forces** stipule que, conformément aux fondements du coaching appréciatif définis par Cooperrider, les réponses doivent systématiquement mettre en valeur les ressources, les compétences et les réussites de l'utilisateur plutôt que ses manques ou ses échecs [7].

Le **principe de bienveillance et de sécurité** exige que le système détecte et évite tout contenu potentiellement nuisible, anxiogène ou contre-productif dans un contexte d'accompagnement émotionnel.

Enfin, le **principe de cohérence émotionnelle** impose que le ton et le contenu de chaque réponse soient cohérents avec l'émotion normalisée détectée chez l'utilisateur : une réponse à une émotion de tristesse ou de peur doit adopter un registre empathique et rassurant, tandis qu'une réponse à une émotion de joie doit être chaleureuse et positive, conformément au mapping défini dans l'étape de normalisation [28].

Chapitre 4

Conception et Implémentation

4.1 Introduction

Ce chapitre présente la conception détaillée et l'implémentation concrète de notre solution Agentic-RAG. En s'appuyant sur les besoins formalisés dans le chapitre précédent et sur les choix technologiques justifiés dans l'état de l'art, nous décrivons l'environnement de développement retenu, puis détaillons chacun des trois pipelines qui composent notre système : le pipeline de chat avec détection d'émotions, le pipeline d'ingestion du corpus depuis Google Drive, et le pipeline de gestion du cycle de vie des documents. Nous présentons également l'API de détection d'émotions développée en local, la base de données vectorielle Supabase, ainsi que les limites et les améliorations envisagées.

4.2 Environnement de Développement

La réalisation de notre projet repose sur un ensemble de technologies et d'outils complémentaires, soigneusement sélectionnés pour leur compatibilité et leur adéquation avec les besoins du système. Le tableau suivant récapitule la stack technique complète de notre implémentation.

4.3 Architecture Globale du Système

Notre système s'articule autour de trois pipelines indépendants mais complémentaires, tous orchestrés via n8n. La Figure 4.1 illustre l'architecture globale du système avec les flux de données entre chaque composant.

Le premier pipeline, dit pipeline de chat, traite chaque message entrant de l'utilisateur en cinq étapes séquentielles et retourne une réponse adaptée à l'émotion détectée, ancrée dans les documents indexés dans Supabase.

Composant	Technologie	Rôle
Orchestrateur de workflow	n8n 2.11.4 (Self Hosted)	Coordination de l'ensemble du pipeline
Base de données vectorielle	Supabase (PostgreSQL + pgvector)	Stockage et recherche sémantique
Embeddings	HuggingFace Inference (*paraphrase-multilingual-MiniLM-L12-v2*)	Vectorisation multilingue
Modèle de langage	Google Gemini	Génération de réponses
API d'émotions	FastAPI + uvicorn	Détection locale des émotions
Tunnel réseau	ngrok	Exposition publique de l'API locale
Source de documents	Google Drive	Stockage et déclenchement d'ingestion
Mémoire conversationnelle	Postgres Chat Memory	Historique persistant par session

TABLE 4.1 – Stack technique de l'implémentation

Le deuxième pipeline, dit pipeline d'ingestion, indexe automatiquement les nouveaux documents déposés dans Google Drive en les vectorisant et en les stockant dans Supabase.

Le troisième pipeline, dit pipeline de gestion documentaire, assure la mise à jour et la suppression des vecteurs en cas de modification ou de suppression d'un document dans Google Drive [28].

4.4 Pipeline de Chat — Détection d'Émotions et RAG

Le pipeline de chat constitue le cœur fonctionnel de notre système. Il est déclenché par un nœud Webhook exposant l'endpoint POST /chat, qui accepte les requêtes provenant de l'interface web et retourne les réponses via un nœud Respond to Webhook. Ce mécanisme de webhook, contrairement au Chat Trigger natif de n8n, permet une intégration flexible avec n'importe quelle interface front-end externe, notamment notre interface React [39].

Le flux complet de ce pipeline est le suivant :

```
Webhook Chat → Prepare User Message → HTTP Emotion API → Normalize
Emotion → AI Agent → Respond to Webhook
```

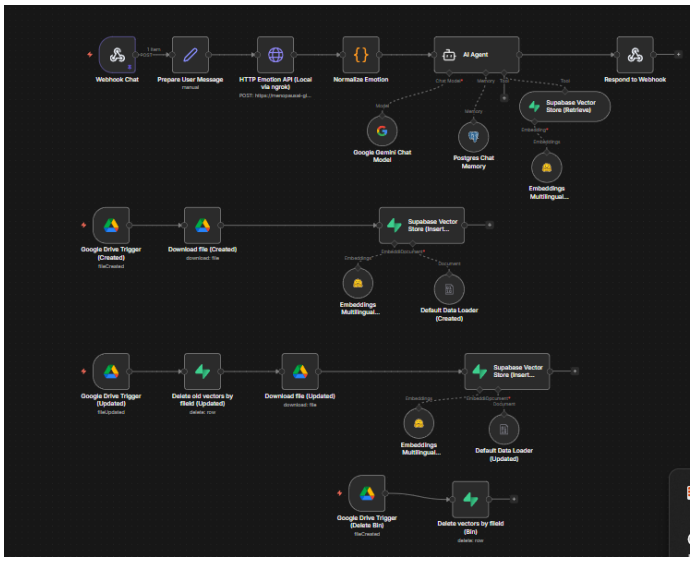

FIGURE 4.1 – Architecture globale du système Agentic-RAG

4.4.1 Préparation du Message Utilisateur

Le nœud *Prepare User Message* reçoit la requête HTTP entrante et extrait deux informations essentielles : le message textuel de l'utilisateur (`userMessage`) et son identifiant de session (`sessionId`). Ce dernier est utilisé par la mémoire conversationnelle pour maintenir des historiques distincts par utilisateur, garantissant ainsi la confidentialité et la continuité de chaque conversation.

4.4.2 Détection de l'Émotion via l'API Locale

Le nœud *HTTP Emotion API* envoie une requête POST à l'API locale de détection d'émotions, exposée publiquement via ngrok à l'adresse `https://xxxx.ngrok.io`. Le corps de la requête contient le message de l'utilisateur sous format JSON :

```json
{ "text": "{{ $json.userMessage }}" }
```

FIGURE 4.2 – Requête JSON envoyée à l'API de détection d'émotions

L'API retourne un résultat structuré contenant l'émotion primaire détectée et son score de confiance :

4.4.3 Normalisation de l'Émotion

Le nœud *Normalize Emotion*, implémenté en JavaScript, réalise le regroupement des 27 émotions brutes du modèle GoEmotions vers les 7 catégories opérationnelles définies.

```json
{
  "success": true,
  "results": {
    "0": {
      "primary_emotion": "sadness",
      "primary_confidence": 98.81
    }
  }
}
```


FIGURE 4.3 – Réponse JSON de l'API de détection d'émotions

Il extrait l'émotion primaire, vérifie qu'elle appartient à la liste des catégories autorisées — joy, sadness, anger, neutral, fear, surprise, love — et la redirige vers neutral si elle n'est pas reconnue. Il transmet également le score de confiance (confidence) à l'étape suivante pour enrichir le contexte de l'agent [28]. La Figure 4.4 illustre le flux de normalisation.

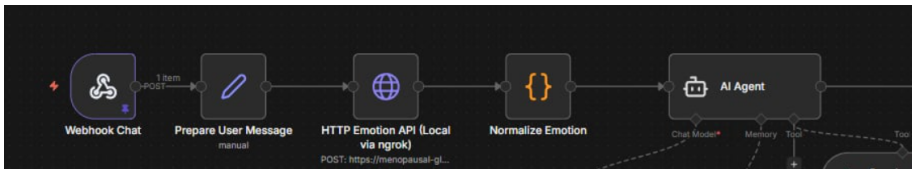

FIGURE 4.4 – Flux de normalisation des émotions (27 → 7 catégories)

4.4.4 L'Agent IA — ResilienceBOT

Le nœud *AI Agent* constitue le composant central de génération de réponses. Il reçoit en entrée l'émotion normalisée et son score de confiance, ainsi que le message original de l'utilisateur, structurés dans le prompt utilisateur suivant :

```
Émotion détectée : {{ $json.detectedEmotion }} ({{ $json.confidence }}%)
Message : {{ $json.userMessage }}
```

L'agent est configuré avec un prompt système détaillé définissant son identité, ses règles de comportement et son processus de traitement. Le bot se présente sous le nom de **ResilienceBOT**, un assistant de coaching émotionnel et de résilience.

Son comportement est régi par quatre règles absolues issues du prompt engineering appliqué dans notre projet [19] :

1. **Utilisation de l'outil :** Utilisation obligatoire de l'outil de recherche Supabase avant toute génération.
2. **Ancrage documentaire :** Les réponses doivent s'inspirer exclusivement des résultats retournés.
3. **Protocole de questionnement :** Maximum 1 à 2 questions par message (max 5 au total).

4. **Format des réponses :** Toujours en français, 4 phrases maximum, avec mention de la source.

L'adaptation du ton selon l'émotion normalisée est également définie dans le prompt système :

- **Sadness :** ton empathique et rassurant ;
- **Anger :** ton calme et apaisant ;
- **Joy :** ton positif et chaleureux ;
- **Neutral :** ton professionnel et neutre ;
- **Fear :** ton rassurant et encourageant ;
- **Surprise :** ton enthousiaste ;
- **Love :** ton chaleureux et bienveillant [28].

L'agent dispose de trois composants attachés : le Supabase Vector Store en mode retrieve-as-tool, permettant la recherche sémantique dans la base documentaire ; les Embeddings Multilingues HuggingFace pour vectoriser les requêtes de recherche ; et la Postgres Chat Memory pour maintenir l'historique conversationnel de manière persistante entre les sessions, identifiée par le `sessionId` de l'utilisateur [28].

4.4.5 Choix du Modèle de Langage : Google Gemini

Après une phase d'évaluation comparative, nous avons migré du modèle `llama-3.3-70b-versatile` de Groq vers Google Gemini comme modèle de génération principal. Cette décision a été motivée par deux raisons concrètes identifiées lors des tests : d'une part, Gemini suit plus fidèlement les instructions d'outils définies dans le prompt système, réduisant significativement les cas où le LLM ignorait l'outil de recherche Supabase pour répondre depuis ses connaissances générales ; d'autre part, le plan gratuit de Google Gemini offre jusqu'à 1 500 requêtes par jour, contre un taux limite (rate limit) bien plus contraignant sur le plan gratuit de Groq [48]. La Figure 4.5 illustre la configuration de l'agent IA dans n8n.

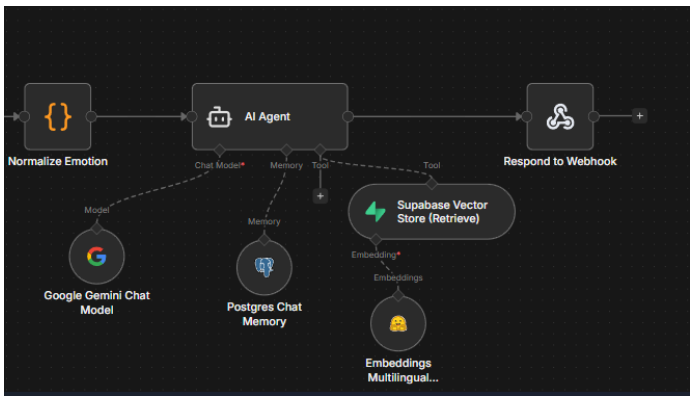

FIGURE 4.5 – Configuration de l'agent IA dans n8n

4.5 Pipeline d'Ingestion — Google Drive vers Supabase

Ce pipeline assure l'indexation automatique des documents déposés dans un dossier Google Drive spécifique (test rag, identifiant : `1Gnfr-SDJMKKrAxhdDPxjqbXsv6ileRKi`). Il est déclenché de manière événementielle, sans intervention manuelle, dès qu'un fichier est ajouté dans ce dossier [40].

Le flux de ce pipeline est le suivant :

`Google Drive Trigger (Created) → Download file → Default Data Loader → Supabase Vector Store (Insert)`

À la détection d'un nouveau fichier, le nœud *Download file* télécharge le document via son identifiant Google Drive. Le nœud *Default Data Loader* charge le contenu binaire du fichier et le découpe automatiquement en chunks de texte exploitables. Ces chunks sont ensuite vectorisés par le modèle d'embeddings `paraphrase-multilingual-MiniLM-L12-v2` de HuggingFace Inference, puis insérés dans la table `documents` de Supabase avec leurs métadonnées associées (`fileId`, `fileName`).

Les formats de fichiers supportés par ce pipeline sont : PDF (.pdf), Word (.docx), texte brut (.txt), CSV (.csv), Markdown (.md) et HTML (.html). Les formats non supportés — Excel, PowerPoint, JSON, images — doivent être convertis en PDF avant l'upload dans Google Drive.

4.6 Pipeline de Gestion du Cycle de Vie des Documents

Ce pipeline comporte deux sous-pipelines distincts qui assurent la cohérence de la base vectorielle lors des modifications et suppressions de documents.

4.6.1 Sous-pipeline de Mise à Jour

Lorsqu'un fichier existant est modifié dans le dossier Google Drive, le trigger *fileUpdated* déclenche le sous-pipeline suivant :

`Google Drive Trigger (Updated) → Delete old vectors → Download file → Default Data Loader → Supabase Vector Store (Insert)`

La première étape consiste à supprimer les anciens vecteurs associés au fichier modifié via un filtre Supabase sur le `fileId` stocké dans les métadonnées :

```
metadata->>fileId=eq.{{ $json.id }}
```

Cette suppression préalable garantit qu'aucun chunk obsolète ne subsiste dans la base vectorielle, évitant ainsi toute incohérence entre les documents Google Drive et le corpus indexé. Le fichier mis à jour est ensuite re-téléchargé, re-découpé, re-vectorisé et réinséré dans Supabase selon le même processus que lors de la création.

4.6.2 Sous-pipeline de Suppression

Lorsqu'un fichier est déplacé vers la corbeille Google Drive (dossier poubelle, identifiant : `1GTIXSP2bGCZKTew6vuwZB0lqFJPudR3V`), le trigger correspondant déclenche la suppression directe des vecteurs associés dans Supabase via un filtre sur le nom du fichier :

```
metadata->>fileName=eq.{{ $json.name }}
```

Ce mécanisme garantit que le corpus vectoriel reste toujours synchronisé avec le contenu effectif du dossier Google Drive, sans nécessiter aucune intervention manuelle de l'administrateur [28].

4.7 API de Détection d'Émotions

4.7.1 Architecture de l'API

L'API de détection d'émotions a été développée en Python en utilisant le framework FastAPI, reconnu pour ses performances élevées, sa prise en charge native de la programmation asynchrone et la génération automatique de documentation OpenAPI [41]. Le serveur est lancé via uvicorn, un serveur ASGI (Asynchronous Server Gateway Interface) asynchrone et haute performance, parfaitement adapté aux besoins de notre pipeline temps réel [42].

L'API expose un endpoint POST /predict qui reçoit le message textuel de l'utilisateur, l'analyse via le modèle `j-hartmann/emotion-english-distilroberta-base` chargé depuis HuggingFace, et retourne l'émotion primaire détectée ainsi que son score de confiance en pourcentage. Le service tourne localement sur le port 8000 et est lancé via la commande :

```
python -m uvicorn api_server:app --host 0.0.0.0 --port 8000
```

4.7.2 Exposition via ngrok

Afin de rendre l'API locale accessible depuis n8n, qui peut tourner sur un réseau différent, nous utilisons ngrok comme service de tunneling. ngrok est un proxy inverse qui

crée un tunnel sécurisé entre un endpoint local et une URL publique accessible depuis n'importe où sur internet, sans nécessiter de configuration réseau complexe ni d'ouverture de ports [43]. La commande de lancement du tunnel est :

```
ngrok http 8000
```

ngrok génère alors une URL publique HTTPS unique — par exemple `https://xxxx.ngrok.io` — qui redirige tout le trafic entrant vers le serveur FastAPI local sur le port 8000. Cette URL est ensuite renseignée dans le nœud *HTTP Emotion API* de n8n pour permettre la communication entre le workflow et l'API [28].

Une limitation connue de cette approche est que ngrok génère une nouvelle URL aléatoire à chaque redémarrage sur le plan gratuit, nécessitant une mise à jour manuelle du nœud n8n. La solution long terme envisagée est le déploiement du modèle sur Hugging Face Spaces, qui fournirait une URL fixe et permanente.

4.8 Tentative de Fine-tuning du Modèle de Détection d'Émotions

Dans l'optique d'améliorer les performances de détection émotionnelle sur des textes spécifiques au contexte du coaching appréciatif — qui diffèrent sensiblement des textes issus de Twitter ou Reddit sur lesquels le modèle `j-hartmann/emotion-english-distilroberta-base` a été initialement entraîné — nous avons entrepris une tentative de fine-tuning de ce modèle sur une base de données spécialisée constituée manuellement.

Cette démarche s'inscrit dans les principes du transfer learning présentés dans l'état de l'art : il s'agissait de tirer parti des représentations générales déjà apprises par le modèle pré-entraîné, tout en l'adaptant à notre domaine spécifique via un entraînement complémentaire sur des données annotées plus proches de notre contexte applicatif [15]. Le processus a suivi les étapes classiques d'un fine-tuning : chargement du checkpoint pré-entraîné depuis HuggingFace, préparation et tokenisation du corpus d'entraînement, définition des hyperparamètres (taux d'apprentissage, nombre d'époques, taille des batchs), puis entraînement et évaluation sur un jeu de test séparé.

Cependant, les résultats obtenus n'ont pas été satisfaisants. Le modèle fine-tuné affichait une précision inférieure à celle du modèle original sur les données de test, suggérant soit un volume de données d'entraînement insuffisant pour surpasser les représentations apprises sur les 58 000 exemples du dataset GoEmotions, soit un déséquilibre dans la distribution des classes émotionnelles dans notre corpus. Face à ces résultats, nous avons pris la décision de revenir au modèle pré-entraîné original, qui offre des performances stables et satisfaisantes dans notre contexte. Cette expérience nous a néanmoins permis d'acquérir une compréhension pratique approfondie des techniques de fine-tuning et de transfer learning des LLMs, constituant un apport pédagogique significatif du projet [16].

4.9 Base de Données Supabase

4.9.1 Structure de la Table Documents

Toutes les données vectorisées sont stockées dans une table `documents` hébergée sur Supabase. Chaque ligne de cette table représente un chunk de texte extrait d'un document, accompagné de son vecteur d'embedding et de ses métadonnées. Les requêtes SQL suivantes permettent de vérifier l'état de la base et de contrôler les documents indexés :

```
-- Lister tous les documents indexés avec leur nombre de chunks
SELECT metadata->>'fileName' AS fichier, COUNT(*) AS nb_chunks
FROM documents
GROUP BY metadata->>'fileName'
ORDER BY nb_chunks DESC;

-- Vérifier les vecteurs associés à un fichier spécifique
SELECT id, metadata->>'fileName', metadata->>'fileId'
FROM documents
WHERE metadata->>'fileName' = 'nom_du_fichier.pdf'
LIMIT 10;
```

4.9.2 Recherche Sémantique

La recherche sémantique est réalisée par pgvector, l'extension PostgreSQL de Supabase, qui calcule la similarité cosinus entre le vecteur de la requête utilisateur et les vecteurs des chunks indexés, et retourne les passages les plus pertinents. Cette recherche est déclenchée automatiquement par l'agent IA à chaque interaction, conformément à la règle absolue n°1 définie dans son prompt système [44].

4.10 Conclusion

Ce chapitre a présenté en détail la conception et l'implémentation de notre système Agentic-RAG. Nous avons décrit l'architecture globale du système et ses trois pipelines principaux, le développement de l'API de détection d'émotions en FastAPI exposée via ngrok, la configuration de ResilienceBOT avec son prompt système structuré, la gestion du corpus documentaire via Google Drive et Supabase, ainsi que les problèmes rencontrés et les solutions apportées. Le chapitre suivant sera consacré aux tests et à l'évaluation de l'ensemble de la solution.

Chapitre 5

Tests et Évaluation

5.1 Introduction

Ce chapitre présente la stratégie de test adoptée pour valider le bon fonctionnement de notre système Agentic-RAG, ainsi que les résultats obtenus lors des campagnes de tests réalisées. Les tests fonctionnels constituent le cœur de notre démarche d'évaluation, car ils permettent de vérifier que chaque composant du pipeline se comporte conformément aux spécifications définies dans le cahier des charges, et que l'enchaînement global des modules produit les résultats attendus [45]. Nous présentons successivement l'environnement de test, les scénarios testés et leurs résultats.

5.2 Stratégie et Environnement de Test

Notre stratégie de test repose sur une approche fonctionnelle de bout en bout (end-to-end testing), qui consiste à valider le pipeline complet depuis la réception d'un message utilisateur jusqu'à la génération et la restitution de la réponse finale, en passant par toutes les étapes intermédiaires : détection d'émotions, normalisation, récupération RAG et vérification éthique [46]. Cette approche permet de détecter non seulement les erreurs isolées dans chaque module, mais également les problèmes d'intégration et de communication entre les composants du système.

Les tests ont été réalisés directement dans l'environnement de développement, en utilisant l'interface de chat exposée par le webhook n8n. L'ensemble du pipeline — API FastAPI locale, tunnel ngrok, workflow n8n, base vectorielle Supabase et modèle Google Gemini — a été actif et opérationnel durant toute la phase de test. Les scénarios de test ont été construits de manière à couvrir les cas nominaux (happy path), les cas limites et les situations potentiellement problématiques identifiées lors de l'implémentation [47].

5.3 Tests Fonctionnels du Pipeline de Chat

Les tests fonctionnels du pipeline de chat visent à valider les cinq étapes séquentielles du flux principal : réception du message, détection d'émotion, normalisation, génération RAG et restitution de la réponse. Chaque étape a été testée individuellement, puis le flux complet a été validé de manière intégrée.

5.3.1 Test du Déclenchement et de la Préparation du Message

Le premier test a consisté à envoyer une requête POST à l'endpoint /chat du webhook n8n et à vérifier que le message est correctement extrait et structuré par le nœud *Prepare User Message*, avec la présence des champs `userMessage` et `sessionId` dans le contexte du workflow.

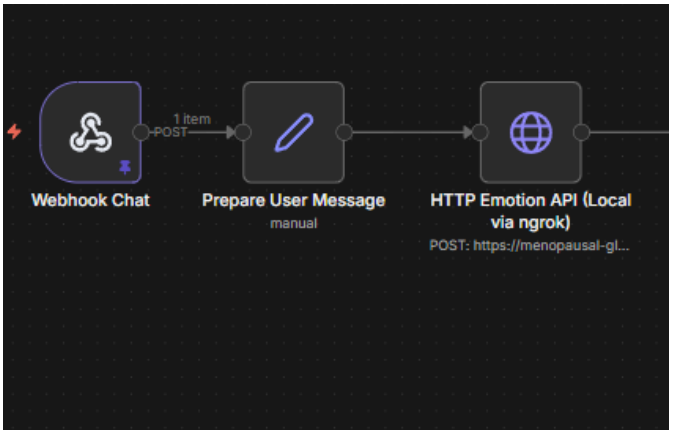

FIGURE 5.1 – Requête POST au webhook et extraction du message utilisateur

Résultat : Le nœud extrait correctement le message et l'identifiant de session, et les transmet au nœud suivant dans le format attendu.

5.3.2 Test de la Détection d'Émotions

Ce test a consisté à envoyer plusieurs messages représentatifs de différents états émotionnels à l'API FastAPI via le nœud *HTTP Emotion API*, et à vérifier que l'émotion détectée correspond à l'état émotionnel exprimé dans le message.

Résultat : L'API retourne des résultats cohérents avec les états émotionnels exprimés, avec des scores de confiance élevés pour les émotions clairement exprimées.

5.3.3 Test de la Normalisation de l'Émotion

Ce test a vérifié que le nœud *Normalize Emotion* regroupe correctement les émotions brutes vers les 7 catégories opérationnelles, et que les émotions non reconnues sont redi-

Message de test	Émotion attendue	Émotion détectée	Score de confiance
« Je me sens complètement dépassé par les événements »	sadness	sadness	~95%
« Je suis tellement en colère contre cette situation »	anger	anger	~92%
« Je viens d'obtenir une promotion, je suis aux anges ! »	joy	joy	~97%
« Je ne sais pas vraiment comment je me sens »	neutral	neutral	~78%
« J'ai peur de l'avenir, tout est incertain »	fear	fear	~88%

FIGURE 5.2 – Résultats de la détection d'émotions sur différents messages de test

rigées vers la catégorie neutral par défaut.

Résultat : La normalisation fonctionne correctement. Le nœud transmet au nœud AI Agent les champs `detectedEmotion`, `confidence` et `userMessage` dans le format attendu.

5.3.4 Test de l'Agent RAG — Utilisation Obligatoire de l'Outil

Ce test est l'un des plus critiques du système. Il vise à vérifier que l'agent IA respecte la règle absolue n°1 de son prompt système, c'est-à-dire qu'il appelle systématiquement l'outil de recherche Supabase avant de formuler toute réponse, sans jamais répondre depuis ses connaissances générales. Le test a consisté à envoyer des messages variés — questions directes sur le coaching appréciatif, demandes d'aide émotionnelle, questions hors corpus — et à vérifier dans les logs n8n la présence du champ `toolcall` avant chaque génération de réponse [28].

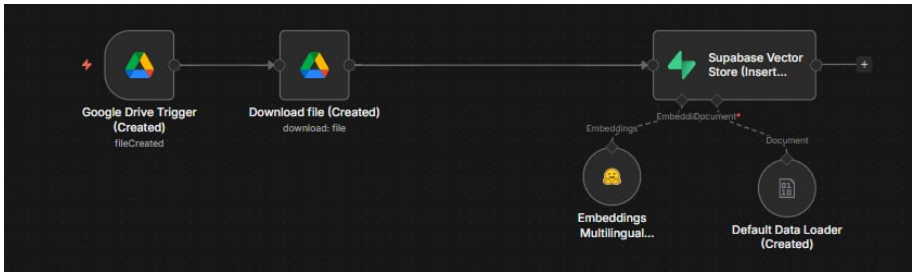

FIGURE 5.3 – Log n8n confirmant l'appel à l'outil Supabase

Résultat : Avec Google Gemini, l'outil de recherche est systématiquement appelé avant toute génération de réponse. Le champ `toolcall` est présent dans les logs pour tous les messages testés, confirmant le respect de la règle absolue n°1.

5.3.5 Test de l'Adaptation du Ton selon l'Émotion

Ce test a vérifié que ResilienceBOT adapte effectivement le ton et le registre de ses réponses en fonction de l'émotion normalisée transmise dans le prompt utilisateur. Plusieurs échanges ont été réalisés avec des messages exprimant des émotions différentes, et les réponses ont été analysées pour confirmer la cohérence entre l'émotion détectée et le registre employé.

FIGURE 5.4 – Exemple de réponse de ResilienceBOT à un message exprimant de la joie

FIGURE 5.5 – Exemple de réponse de ResilienceBOT à un message exprimant de la peur

Résultat : Le ton des réponses est cohérent avec l'émotion détectée. Un message exprimant de la tristesse génère une réponse empathique et rassurante, tandis qu'un message de joie génère une réponse chaleureuse et positive.

5.3.6 Test du Protocole de Questionnement

Ce test a vérifié que ResilienceBOT respecte la règle absolue n°3 de son prompt système : poser au maximum 1 à 2 questions par message, sans dépasser 5 questions au total sur l'ensemble de la conversation. Un scénario conversationnel multi-tours a été simulé pour valider ce comportement.

FIGURE 5.6 – Scénario conversationnel multi-tours illustrant le protocole de questionnement

Résultat : Le bot respecte scrupuleusement le protocole de questionnement défini. Après 5 questions, il formule directement un conseil sans poser de question supplémentaire.

5.3.7 Test de la Mémoire Conversationnelle

Ce test a vérifié que la Postgres Chat Memory maintient correctement le contexte de la conversation entre les messages successifs d'un même utilisateur, identifié par son `sessionId`. Un scénario en plusieurs échanges a été simulé pour vérifier que le bot se souvient des informations partagées dans les messages précédents.

Résultat : La mémoire conversationnelle fonctionne correctement. Le bot fait référence aux informations partagées dans les messages précédents, garantissant la continuité et la cohérence de la conversation.

5.4 Tests Fonctionnels du Pipeline d'Ingestion

5.4.1 Test d'Ajout d'un Document

Ce test a consisté à déposer un nouveau document PDF dans le dossier Google Drive *test rag* et à vérifier que le pipeline d'ingestion se déclenche automatiquement, télécharge le fichier, le vectorise et l'insère dans la table `documents` de Supabase.

La requête SQL suivante a été utilisée pour vérifier l'insertion des vecteurs dans Supabase :

```
SELECT metadata->>'fileName', COUNT(*)
FROM documents
GROUP BY metadata->>'fileName'
ORDER BY COUNT(*) DESC;
```

Résultat : Le pipeline s'exécute automatiquement à la détection du nouveau fichier. Les chunks du document sont correctement vectorisés et insérés dans Supabase avec leurs métadonnées.

5.4.2 Test de Mise à Jour d'un Document

Ce test a vérifié que la mise à jour d'un document existant dans Google Drive déclenche bien la suppression des anciens vecteurs associés avant l'insertion des nouveaux, évitant toute duplication ou incohérence dans la base vectorielle.

Résultat : Les anciens vecteurs sont supprimés via le filtre `fileId` avant l'insertion des nouveaux chunks. La base vectorielle reste cohérente après la mise à jour.

5.4.3 Test de Suppression d'un Document

Ce test a vérifié que le déplacement d'un document vers la corbeille Google Drive déclenche automatiquement la suppression de tous les vecteurs associés dans Supabase.

Résultat : Les vecteurs correspondants sont supprimés via le filtre `fileName`. Le corpus reste synchronisé avec le contenu effectif du dossier Google Drive.

5.4.4 Conclusion

Ce chapitre a présenté la stratégie de test adoptée et les résultats obtenus lors des campagnes de tests fonctionnels menées sur l'ensemble du système. L'ensemble des dix scénarios de test réalisés a produit des résultats satisfaisants, validant le bon fonctionnement de chaque composant du pipeline ainsi que leur intégration globale. Les limites identifiées — dépendance à ngrok, gestion des formats non supportés, absence d'un agent de vérification formel — ouvrent des perspectives d'amélioration concrètes qui feront l'objet de travaux futurs. La conclusion générale du rapport synthétisera les apports de ce projet et tracera les perspectives d'évolution à court et long terme.

Conclusion Générale

Ce projet de fin d'année avait pour ambition de concevoir et de mettre en œuvre un chatbot intelligent capable de détecter l'état émotionnel de l'utilisateur, de mobiliser des connaissances issues du coaching appréciatif, et de produire des réponses personnalisées et éthiquement conformes. L'ensemble de ces objectifs a été atteint grâce à une architecture Agentic-RAG orchestrée via n8n, combinant le modèle de détection d'émotions `j-hartmann/emotion-english-distilroberta-base`, les embeddings multilingues `paraphrase-multilingual-MiniLM-L12-v2`, la base vectorielle Supabase et le modèle de génération Google Gemini.

Parmi les contributions majeures de ce projet, deux méritent d'être soulignées. D'une part, l'intégration des contraintes éthiques du coaching appréciatif directement dans le prompt système de l'agent RAG garantit la non-directivité, l'orientation vers les forces et la bienveillance des réponses produites. D'autre part, la normalisation des 27 émotions brutes vers 7 catégories opérationnelles a rendu le système plus robuste et plus cohérent dans l'adaptation du ton de ses réponses.

Toutefois, ce projet n'est pas exempt de limites. L'exposition de l'API via ngrok génère une URL temporaire à chaque redémarrage, et la tentative de fine-tuning du modèle de détection d'émotions n'a pas abouti aux résultats escomptés en raison d'un volume de données insuffisant. En perspective, le déploiement du modèle sur Hugging Face Spaces, la constitution d'un corpus d'entraînement plus large et l'ajout de fonctionnalités de suivi longitudinal des émotions constituent des axes d'amélioration naturels pour faire évoluer ce système vers une solution plus robuste et complète.

En définitive, ce projet nous a permis d'acquérir une expérience pratique approfondie dans des domaines technologiques en pleine effervescence — les LLMs, le RAG agentique, la détection d'émotions et l'orchestration de workflows d'IA — tout en nous confrontant aux défis concrets de la conception de systèmes conversationnels éthiquement responsables. Il constitue ainsi une base solide et prometteuse pour de futurs développements dans le domaine de l'intelligence artificielle au service du bien-être humain.

Bibliographie

[1] Sorin, V. et al. *Large Language Models and Empathy : Systematic Review*. Journal of Medical Internet Research, 2024. https://www.jmir.org/2024/1/e52597/

[2] Gaffney, H. et al. *Empathy by Design : Reframing the Empathy Gap Between AI and Humans in Mental Health Chatbots*. Information, 16(12), 1074, MDPI, 2025. https://www.mdpi.com/2078-2489/16/12/1074

[3] Fitzpatrick, K.K., Darcy, A. & Vierhile, M. *Delivering Cognitive Behavior Therapy to Young Adults With Symptoms of Depression and Anxiety Using a Fully Automated Conversational Agent (Woebot) : A Randomized Controlled Trial*. JMIR Mental Health, 4(2), e19, 2017. https://mental.jmir.org/2017/2/e19

[4] STAT News. *Woebot Health shuts down pioneering therapy chatbot*. Juillet 2025. https://www.statnews.com/2025/07/02/woebot-therapy-chatbot-shuts-down/

[5] Lissak et al. *AI in Mental Health : Emotional and Sentiment Analysis of Large Language Models' Responses*. arXiv, 2025. https://arxiv.org/html/2508.11285v1

[6] Hartmann, J. *Emotion English DistilRoBERTa-base*. HuggingFace, 2022. https://huggingface.co/j-hartmann/emotion-english-distilroberta-base/

[7] Cooperrider, D.L. & Srivastva, S. *Appreciative inquiry in organizational life*. Research in Organizational Change & Development, 1, 129–169, 1987. Cité dans : The Positive Psychology People, 2023. https://www.thepositivepsychologypeople.com/appreciative-inquiry-in-coaching/

[8] Passmore, J. & Tee, D. *A systematic literature review of artificial intelligence (AI) in coaching*. Journal of Work-Applied Management, Emerald Publishing, 2025. https://www.emerald.com/jwam/article/doi/10.1108/JWAM-11-2024-0164/

[9] Wnuk, M. et al. *What Is Machine Learning, Artificial Neural Networks and Deep Learning ? — Examples of Practical Applications in Medicine*. Diagnostics, MDPI, 13(15), 2582, 2023. https://www.mdpi.com/2075-4418/13/15/2582

[10] Wikipedia. *Deep Learning*. https://en.wikipedia.org/wiki/Deep_learning

[11] IBM. *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks*. IBM Think, 2025. https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

[12] IBM. *What Is Deep Learning ?* IBM Think, 2025. https://www.ibm.com/think/topics/deep-learning

[13] Gao, Y. et al. *Retrieval-Augmented Generation for Large Language Models : A Survey*. arXiv :2312.10997, 2023. https://arxiv.org/abs/2312.10997

[14] Groq. *New AI Inference Speed Benchmark for Llama 3.3 70B, Powered by Groq*. Groq Blog, 2024. https://groq.com/blog/new-ai-inference-speed-benchmark-for-llama-3-3-70b-powered-by-groq

[15] IBM. *What is Fine-Tuning ?* IBM Think, 2025. https://www.ibm.com/think/topics/fine-tuning

[16] DataCamp. *Fine-Tuning Large Language Models : A Guide With Examples*. 2024. https://www.datacamp.com/tutorial/fine-tuning-large-language-models

[17] Hartmann, J. *Emotion English DistilRoBERTa-base* (checkpoint fine-tuné DistilRoBERTa-base). HuggingFace, 2022. https://huggingface.co/j-hartmann/emotion-english-distilroberta-base

[18] Wikipedia. *Prompt Engineering*. https://en.wikipedia.org/wiki/Prompt_engineering

[19] Sahoo, P. et al. *A Systematic Survey of Prompt Engineering in Large Language Models : Techniques and Applications*. arXiv :2402.07927, 2024. https://arxiv.org/abs/2402.07927

[20] Chourey, L. *Steering LLMs Toward Desired Outputs With Prompt Engineering*. ISACA Journal, Vol. 6, 2024. https://www.isaca.org/resources/isaca-journal/issues/2024/volume-6/steering-llms-toward-desired-outputs-with-prompt-engineering

[21] IBM. *What is RAG (Retrieval Augmented Generation) ?* IBM Think, 2024. https://www.ibm.com/think/topics/retrieval-augmented-generation

[22] Lewis, P. et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. NeurIPS, 33, 9459–9474, 2020.

[23] DigitalOcean. *RAG, AI Agents, and Agentic RAG : An In-Depth Review and Comparative Analysis*. 2025. https://www.digitalocean.com/community/conceptual-articles/rag-ai-agents-agentic-rag-comparative-analysis

[24] IBM. *What is Agentic RAG ?* IBM Think, 2025. https://www.ibm.com/think/topics/agentic-rag

[25] Hartmann, J. Op. cit. .

[26] Hartmann, J. Op. cit. .

[27] Venkiteela, P. *Documentation technique du workflow n8n — RAG Pipeline Multilang + Emotion*. Version 2.0, Mars 2026. (Documentation interne du projet)

[28] Venkiteela, P. *Documentation technique du workflow n8n — RAG Pipeline Multilang + Emotion*. Version 2.0, Mars 2026. (Documentation interne du projet)

[29] Reimers, N. & Gurevych, I. *Sentence-BERT : Sentence Embeddings using Siamese BERT-Networks*. EMNLP, 2019. https://arxiv.org/abs/1908.10084

[30] HuggingFace. *paraphrase-multilingual-MiniLM-L12-v2*. https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2

[31] Supabase. *pgvector : Embeddings and vector similarity*. Supabase Docs, 2024. https://supabase.com/docs/guides/database/extensions/pgvector

[32] n8n. *Documentation officielle n8n*. https://docs.n8n.io/

[33] Venkiteela, P. *n8n : An Open-Source Workflow Automation Platform for Enterprise Integration and AI-Driven Orchestration*. International Journal of Computer Applications, 187(63), 2025. https://ijcaonline.org/archives/volume187/number63/

[34] n8n. Op. cit. .

[35] Gonçalves, L.P., Canedo, E.D. & Santos, G. *A Meta-model for Documenting Conversational Requirements in Chatbots*. In : Requirements Engineering : Foundation for Software Quality, Springer, 2024. https://link.springer.com/chapter/10.1007/978-3-031-70245-7_5

[36] Hartmann, J. Op. cit. [6].

[37] Gupta, S., Ranjan, R. & Singh, S.N. *Comprehensive Framework for Evaluating Conversational AI Chatbots*. arXiv :2502.06105, 2025. https://arxiv.org/abs/2502.06105

[38] Gupta et al. Op. cit. [37].

[39] n8n. *Documentation officielle n8n — Webhook node*. https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.webhook/

[40] Venkiteela, P. Op. cit. [27].

[41] FastAPI. *Documentation officielle FastAPI*. https://fastapi.tiangolo.com/

[42] Wikipedia. *FastAPI*. https://en.wikipedia.org/wiki/FastAPI

[43] Sendbird. *What is ngrok ? Using ngrok tunneling*. https://sendbird.com/developer/tutorials/what-is-ngrok

[44] Supabase. *Semantic search using Supabase Vector*. https://supabase.com/docs/guides/ai/semantic-search

[45] Cekura. *Chatbot Testing Tools and Techniques : A Complete Guide*. 2025. https://www.cekura.ai/blogs/complete-chatbot-testing-guide-ai-agents

[46] KiwiQA. *An Ultimate Guide To Chatbot Testing Checklist*. 2023. https://www.kiwiqa.com/guide-to-chatbot-testing-checklist/

[47] AIMMultiple Research. *Chatbot Testing : A/B, Auto & Manual Testing*. 2025. https://research.aimultiple.com/chatbot-testing-frameworks/

[48] Google. *Gemini API — Rate Limits and Quotas*. Google AI for Developers, 2024. https://ai.google.dev/gemini-api/docs/rate-limits

[49] Woebot Health. *Woebot — Your Mental Health Ally* [capture d'écran de l'interface]. Woebot Health, 2023. https://woebothealth.com/

[50] Venkiteela, P. *Diagramme : Architecture globale du pipeline Agentic-RAG*. Documentation interne du projet, Mars 2026.

[51] IBM. *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks* [figure hiérarchique]. IBM Think, 2025. https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

[52] DataCamp. *Fine-Tuning Large Language Models : A Guide With Examples* [figure du processus de fine-tuning]. DataCamp, 2024. https://www.datacamp.com/tutorial/fine-tuning-large-language-models

[53] Gao, Y. et al. *Retrieval-Augmented Generation for Large Language Models : A Survey* [figure architecture RAG classique]. arXiv :2312.10997, 2023. https://arxiv.org/abs/2312.10997

[54] n8n. *Documentation officielle n8n* [figure architecture d'un workflow n8n]. https://docs.n8n.io/