Étude stratégique pour la conception d'un chatbot

Sié Hans Ouattara

Table des matières

1 Introduction Générale

1.1 Contexte du Projet

Dans un environnement économique de plus en plus digitalisé, les entreprises, notamment les Petites et Moyennes Entreprises (PME), cherchent à optimiser leurs processus internes tout en améliorant leur relation client. L'automatisation intelligente constitue aujourd'hui un levier stratégique permettant de gagner en efficacité, de réduire les coûts opérationnels et d'améliorer la qualité du service.

La société Flormar, disposant actuellement d'une infrastructure principalement basée sur WordPress, souhaite moderniser son écosystème numérique. L'objectif est d'intégrer une solution conversationnelle intelligente capable d'assister les différents services de l'entreprise (commercial, support, gestion interne, etc.) tout en restant compatible avec les contraintes techniques et budgétaires existantes.

1.2 Problématique

Malgré l'existence de nombreuses solutions de chatbots sur le marché, le choix d'une technologie adaptée à une PME soulève plusieurs questions :

— Quelle architecture technique adopter ?
— Faut-il privilégier une solution cloud ou locale ?
— Quel modèle de langage (LLM) choisir ?
— Comment garantir la disponibilité, la performance et la sécurité des données ?
— Est-il pertinent d'intégrer un ERP existant comme Odoo ou d'opter pour une solution alternative ?

La problématique centrale de ce travail est donc la suivante :

Comment concevoir et déployer un chatbot assistant intelligent, performant et évolutif, adapté aux besoins d'une PME, tout en optimisant les coûts, la disponibilité et l'efficacité opérationnelle ?

1.3 Objectifs du Projet

Ce projet vise plusieurs objectifs :

Objectifs Fonctionnels

— Automatiser certaines tâches répétitives.
— Améliorer l'accessibilité aux informations internes.
— Assister les utilisateurs via une interface conversationnelle intuitive.
— Intégrer le chatbot aux outils existants (site web WordPress, ERP, bases de données).

Objectifs Techniques

- Étudier les différentes architectures possibles (cloud vs local).
- Comparer plusieurs modèles LLM (open-source et propriétaires).
- Mettre en place une architecture basée sur le Retrieval-Augmented Generation (RAG).
- Garantir la sécurité, la scalabilité et la disponibilité du système.

1.4 Méthodologie Adoptée

Afin de répondre à la problématique posée, une démarche structurée sera adoptée :

1. Analyse des besoins et rédaction d'un cahier des charges détaillé.
2. Étude théorique des agents LLM et des architectures conversationnelles.
3. Conception d'une architecture technique adaptée au contexte de la PME.
4. Étude comparative des solutions ERP, des modèles LLM et des options d'hébergement.
5. Justification des choix technologiques retenus.

Cette approche permettra d'assurer une vision à la fois stratégique, technique et opérationnelle du projet, tout en garantissant sa cohérence et sa faisabilité.

2 Fondements Théoriques

2.1 Les Grands Modèles de Langage (LLM)

2.1.1 Définition Générale

Les Grands Modèles de Langage (Large Language Models – LLM) sont des modèles d'intelligence artificielle basés sur des réseaux de neurones profonds, entraînés sur de vastes corpus de données textuelles. Leur objectif est de comprendre, générer et manipuler le langage naturel avec un haut degré de cohérence et de pertinence.

Les LLM modernes reposent principalement sur l'architecture **Transformer**, introduite en 2017. Cette architecture utilise un mécanisme appelé *attention* permettant au modèle d'analyser les relations entre les mots d'une phrase, indépendamment de leur position. Cela améliore considérablement la compréhension du contexte par rapport aux modèles séquentiels traditionnels.

2.1.2 Fonctionnement Simplifié

Le fonctionnement d'un LLM peut être résumé en trois étapes principales :

1. **Tokenisation :** Le texte d'entrée est découpé en unités appelées *tokens*. Un token peut être un mot, une partie de mot ou un symbole.
2. **Encodage et Attention :** Les tokens sont transformés en vecteurs numériques. Le mécanisme d'attention calcule les relations entre ces vecteurs afin de déterminer l'importance relative de chaque mot dans le contexte global.
3. **Prédiction :** Le modèle prédit le token suivant le plus probable, en fonction du contexte précédent. La génération d'un texte complet résulte de la répétition de ce processus.

Ainsi, un LLM fonctionne essentiellement comme un modèle probabiliste sophistiqué capable d'estimer la distribution des mots dans une langue.

2.1.3 Limites des LLM "Nus"

Malgré leurs performances impressionnantes, les LLM présentent plusieurs limitations majeures lorsqu'ils sont utilisés seuls dans un contexte d'entreprise :

— **Hallucinations :** Le modèle peut générer des informations plausibles mais incorrectes, car il ne vérifie pas la véracité des données produites.
— **Obsolescence des données :** Un LLM pré-entraîné possède une base de connaissances figée à la date de fin de son entraînement. Il ne dispose pas d'informations actualisées en temps réel.
— **Absence de données spécifiques à l'entreprise :** Les modèles génériques ne connaissent ni les bases de données internes, ni les processus métiers spécifiques d'une organisation.
— **Manque de traçabilité :** Les réponses générées ne sont pas toujours explicables ni référencées à une source identifiable.

2.1.4 Implication pour le Contexte Entreprise

Dans un contexte professionnel, ces limitations rendent l'utilisation d'un LLM isolé insuffisante. Une entreprise nécessite :

— Des réponses fiables et vérifiables ;
— Un accès aux données internes actualisées ;
— Un contrôle sur la confidentialité des informations ;
— Une intégration avec les systèmes existants (ERP, CRM, bases de données).

Ainsi, l'utilisation d'un LLM seul (par exemple via une interface standard type ChatGPT) ne répond pas aux exigences opérationnelles d'une PME. Il devient nécessaire d'enrichir le modèle par des mécanismes complémentaires permettant d'intégrer des sources de données spécifiques et dynamiques.

Cette problématique conduit naturellement à l'étude du paradigme *Retrieval-Augmented Generation (RAG)*, présenté dans la section suivante.

2.2 Le Paradigme Retrieval-Augmented Generation (RAG)

2.2.1 Définition Générale

Le paradigme *Retrieval-Augmented Generation* (RAG) consiste à améliorer les performances d'un LLM en le connectant à une base de connaissances externe.

Au lieu de générer une réponse uniquement à partir de ses paramètres internes, le modèle commence par rechercher des informations pertinentes dans une source de données spécifique (documents internes, base ERP, fichiers PDF, base de données), puis utilise ces informations comme contexte pour produire une réponse plus fiable et contextualisée.

Le RAG permet ainsi de combiner :

— La capacité générative des LLM ;

— La précision d'une base de connaissances contrôlée.

2.2.2 Principe Fondamental : Les Embeddings et la Recherche Sémantique

Le fonctionnement du RAG repose sur un concept central : la **vectorisation** des données.

Embeddings Un *embedding* est une représentation vectorielle d'un texte dans un espace numérique de grande dimension. Chaque phrase ou document est converti en un vecteur de nombres réels capturant sa signification sémantique.

Ainsi, deux textes ayant un sens similaire auront des vecteurs proches dans cet espace.

Similarité Cosinus Pour mesurer la proximité entre deux vecteurs, on utilise généralement la **similarité cosinus**. Cette mesure calcule l'angle entre deux vecteurs :

— Une similarité proche de 1 indique une forte similarité sémantique ;

— Une valeur proche de 0 indique peu ou pas de relation.

Grâce à cette technique, le système peut rechercher non pas des mots-clés exacts, mais des contenus ayant un sens proche de la question posée.

2.2.3 Architecture Générale d'un Système RAG

Un système RAG typique suit les étapes suivantes :

1. **Chargement des données (Loader)** : Les documents internes (PDF, fichiers texte, données ERP, etc.) sont collectés et préparés.
2. **Découpage (Chunking)** : Les documents sont divisés en segments de taille adaptée afin d'optimiser la recherche.
3. **Vectorisation (Embedding)** : Chaque segment est transformé en vecteur numérique.

4. **Stockage (Vector Database)** : Les vecteurs sont enregistrés dans une base spécialisée (ex. : ChromaDB), permettant une recherche rapide.
5. **Récupération (Retrieval)** : Lorsqu'une question est posée, elle est vectorisée à son tour. Le système recherche les segments les plus proches sémantiquement.
6. **Génération** : Les segments récupérés sont injectés dans le prompt du LLM, qui génère alors une réponse contextualisée.

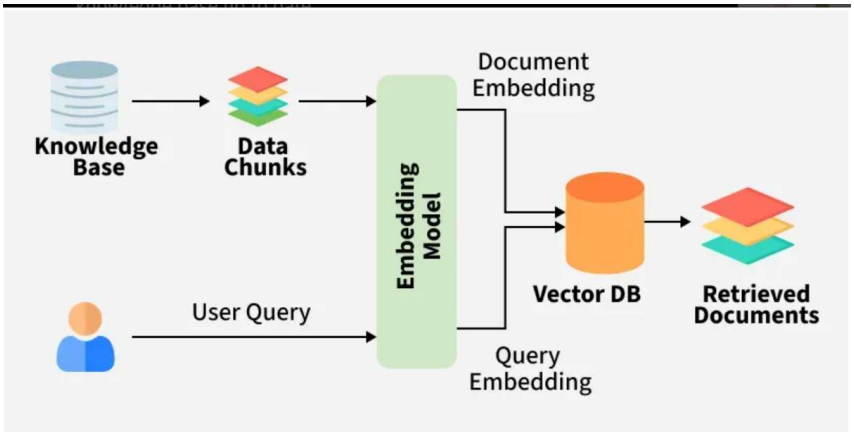

FIGURE 1 – Caption

2.2.4 Avantages du RAG en Contexte PME

L'approche RAG présente plusieurs avantages stratégiques pour une PME :

— **Réduction des hallucinations** grâce à l'ancrage sur des données internes ;
— **Actualisation dynamique** des connaissances (mise à jour de la base sans réentraîner le modèle) ;
— **Maîtrise des données sensibles** ;
— **Coût réduit** comparé au fine-tuning complet d'un modèle.

2.2.5 Lien avec Notre Implémentation

Dans notre architecture, le processus suit précisément ce paradigme :

— Un module de chargement (*loader*) extrait les données pertinentes ;
— Les données sont transformées en embeddings ;
— Les vecteurs sont stockés dans une base vectorielle (ChromaDB) ;
— Lors d'une requête utilisateur, les segments les plus pertinents sont récupérés ;
— Le LLM génère une réponse enrichie par ces informations.

Cette approche constitue le socle technique du système proposé et permet de dépasser les limites des LLM utilisés de manière isolée.

2.3 L'Ingénierie de Prompt (Prompt Engineering)

2.3.1 Définition et Rôle

L'ingénierie de prompt désigne l'ensemble des techniques permettant de formuler et structurer les instructions envoyées à un LLM afin d'obtenir des réponses pertinentes, précises et adaptées au contexte métier.

Un LLM ne comprend pas les intentions comme un humain ; il prédit du texte en fonction des probabilités apprises. La qualité de la réponse dépend donc fortement :

— De la clarté des instructions ;
— Du contexte fourni ;
— De la structure du prompt ;
— Des exemples éventuellement intégrés.

Dans un système RAG, le prompt devient un élément central : il sert de pont entre les données récupérées et la génération finale.

2.3.2 Le Rôle Fondamental du Contexte

Dans une architecture RAG, les segments récupérés via la recherche sémantique sont injectés dans le prompt avant la génération.

Le prompt prend généralement la forme suivante :

— Instruction système (rôle du modèle) ;
— Contexte récupéré (documents internes) ;
— Question utilisateur ;
— Contraintes de réponse (format, ton, précision).

L'ajout explicite du contexte permet :

— De limiter les hallucinations ;
— D'ancrer la réponse sur des données vérifiables ;
— D'améliorer la cohérence métier.

Ainsi, le modèle ne génère pas une réponse générique, mais une réponse guidée par des informations spécifiques à l'entreprise.

2.3.3 Techniques Principales

Zero-Shot Learning Le modèle reçoit uniquement une instruction sans exemple préalable.

Exemple :

« Réponds à la question suivante en utilisant exclusivement les informations fournies dans le contexte. »

Cette approche est simple mais dépend fortement de la qualité du prompt.

Few-Shot Learning Le prompt contient un ou plusieurs exemples de questions-réponses afin de guider le modèle vers le comportement attendu.

Exemple :

— Question exemple ;
— Réponse attendue ;
— Nouvelle question utilisateur.

Cette technique améliore la stabilité et la cohérence des réponses, notamment dans des tâches structurées (classification, extraction, reformulation).

Chain-of-Thought (Raisonnement Décomposé) La technique dite *Chain-of-Thought* consiste à inciter le modèle à expliciter les étapes de son raisonnement avant de produire la réponse finale.

Par exemple :

« Analyse la question étape par étape avant de donner la réponse finale. »

Cette méthode est particulièrement utile pour :

— Les calculs ;
— Les prises de décision complexes ;
— Les tâches nécessitant plusieurs critères.

2.3.4 Importance Stratégique dans Notre Projet

Dans le cadre de notre chatbot :

— Le contexte récupéré via ChromaDB est injecté dynamiquement dans le prompt ;
— Des instructions strictes limitent le modèle à répondre uniquement sur la base des données internes ;
— Le ton et le format des réponses sont contrôlés (professionnel, structuré).

L'ingénierie de prompt constitue ainsi un mécanisme de contrôle essentiel garantissant la fiabilité et l'alignement métier du système.

Elle joue un rôle complémentaire au RAG : tandis que le RAG fournit les bonnes informations, le prompt engineering guide la manière dont ces informations sont exploitées.

2.4 Des Chaînes aux Agents Autonomes

2.4.1 Du Pipeline Linéaire aux Systèmes Décisionnels

Les premiers systèmes basés sur des LLM reposent généralement sur un pipeline simple et linéaire :

Question → Recherche → Génération → Réponse

Dans ce schéma, le processus est prédéfini et ne varie pas en fonction de la nature de la requête. Ce type d'architecture correspond à une implémentation RAG classique et constitue une base solide pour un MVP (Minimum Viable Product).

Cependant, lorsque les besoins deviennent plus complexes (accès à des API, interactions avec un ERP, appels à des services externes), un simple pipeline linéaire devient insuffisant.

2.4.2 Définition d'un Agent

Un **agent** est un système basé sur un LLM capable de :

— Analyser une requête ;

— Déterminer une stratégie de résolution ;

— Sélectionner un ou plusieurs outils ;

— Exécuter des actions ;

— Produire une réponse finale.

Contrairement à un pipeline fixe, un agent possède une capacité décisionnelle dynamique.

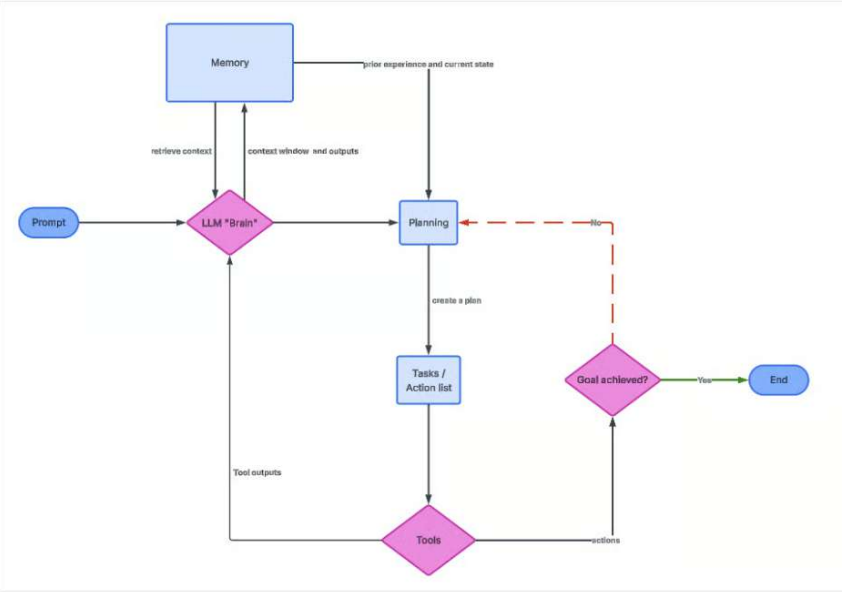

FIGURE 2 – Caption

2.4.3 Les Outils (Tools)

Les outils sont des fonctions ou services externes accessibles au modèle. Par exemple :

— API ERP (ex. : Odoo) ;

— Base de données interne ;

— Recherche web ;

— Calculateur ;
— Système d'envoi d'e-mails.

Dans un système agentique, le LLM ne se limite pas à générer du texte : il choisit d'appeler un outil lorsque cela est nécessaire.

2.4.4 Comparaison : Pipeline RAG vs Agent Réactif

Pipeline RAG (Approche actuelle)

$$\text{Question} \rightarrow \text{Recherche Vectorielle} \rightarrow \text{Injection Contexte} \rightarrow \text{Réponse}$$

Caractéristiques :
— Processus déterministe ;
— Architecture simple ;
— Faible complexité ;
— Idéal pour la consultation documentaire.

Agent Réactif

$$\text{Question} \rightarrow \text{Analyse} \rightarrow \text{Choix d'Outil} \rightarrow \text{Action} \rightarrow \text{Réponse}$$

Caractéristiques :
— Processus adaptatif ;
— Capacité à interagir avec plusieurs systèmes ;
— Plus grande autonomie ;
— Complexité plus élevée.

2.4.5 L'Orchestration

Pour coordonner ces différents composants, des frameworks d'orchestration sont utilisés (par exemple LangChain).

Le rôle de l'orchestrateur est de :
— Gérer le flux des données ;
— Connecter le LLM aux outils ;
— Structurer les chaînes de traitement ;
— Maintenir la cohérence du système.

L'orchestration agit comme un chef d'orchestre reliant le modèle, la base vectorielle, les API et les services externes.

2.4.6 Transition vers une Architecture Hybride

Dans un contexte PME, un agent entièrement autonome peut représenter :
— Un coût élevé ;

— Une complexité accrue ;
— Des risques supplémentaires (actions non contrôlées).

Une approche progressive consiste donc à démarrer par un système RAG robuste, puis à intégrer progressivement des capacités agentiques ciblées (accès ERP, consultation API, automatisations spécifiques).

Cette évolution permet de concilier maîtrise des risques, contrôle des coûts et montée en sophistication fonctionnelle.

2.5 Positionnement Stratégique de Notre Chatbot

2.5.1 Classification de la Solution Proposée

Au regard des concepts théoriques présentés précédemment, la solution développée peut être classifiée comme un :

Système Hybride RAG–Agent

Il ne s'agit ni d'un simple LLM isolé, ni d'un agent totalement autonome, mais d'une architecture intermédiaire combinant :
— Un socle RAG robuste pour l'accès aux connaissances internes ;
— Une structure modulaire permettant l'ajout progressif de capacités agentiques ;
— Une séparation claire entre interface utilisateur et logique métier.

Cette approche garantit un équilibre entre performance, contrôle et évolutivité.

2.5.2 Pourquoi le RAG plutôt que le Fine-Tuning ?

Deux stratégies principales permettent d'adapter un LLM à un contexte métier :

Fine-Tuning Le fine-tuning consiste à réentraîner le modèle sur des données spécifiques à l'entreprise. Cependant, cette approche présente plusieurs limites :
— Coût élevé en ressources de calcul ;
— Difficulté de mise à jour fréquente des données ;
— Risque de surapprentissage ;
— Complexité technique importante.

Approche RAG À l'inverse, le RAG :
— Permet une mise à jour dynamique des données ;
— Ne nécessite pas de réentraînement complet ;
— Réduit les coûts ;
— Offre une meilleure traçabilité des réponses.

Dans un contexte PME, où les données (produits, stocks, informations ERP) évoluent régulièrement, le RAG constitue une solution plus flexible et économiquement viable.

2.5.3 Architecture Modulaire

Le système proposé repose sur une architecture modulaire séparant :

— **Frontend :** Interface utilisateur intégrée au site WordPress ;

— **Backend :** API indépendante (Python) gérant la logique métier, le RAG et les appels au LLM ;

— **Base vectorielle :** Stockage des embeddings (ChromaDB) ;

— **Modèle LLM :** Service externe ou local selon la stratégie retenue.

Cette séparation présente plusieurs avantages :

— Maintenabilité accrue ;

— Possibilité de changer de modèle LLM sans modifier l'interface ;

— Scalabilité indépendante des composants ;

— Meilleure gestion de la sécurité.

2.5.4 Approche "Privacy-First"

La gestion des données constitue un enjeu central pour une PME.

L'architecture retenue adopte une approche dite *Privacy-First*, caractérisée par :

— Le contrôle des données internes au niveau du backend ;

— La possibilité d'opter pour un modèle local si nécessaire ;

— La limitation des informations sensibles transmises aux services externes ;

— La conformité aux exigences de protection des données (ex. RGPD).

Cette approche renforce la confiance dans le système et réduit les risques juridiques et opérationnels.

2.5.5 Vision d'Évolution

Le système est conçu comme une base évolutive :

— Phase 1 : Chatbot documentaire basé sur RAG ;

— Phase 2 : Intégration d'outils spécifiques (API ERP, automatisations) ;

— Phase 3 : Extension vers un agent décisionnel plus autonome.

Cette stratégie progressive permet de :

— Maîtriser les coûts ;

— Réduire les risques ;

— Tester l'acceptation utilisateur ;

— Adapter le niveau d'autonomie aux besoins réels.

Ainsi, la solution proposée n'est pas seulement une implémentation technique, mais une architecture stratégique alignée avec les contraintes et les objectifs d'une PME moderne.

3 Spécifications Fonctionnelles

Cette section décrit les fonctionnalités principales du chatbot Sales Pro pour Flormar. Elle couvre à la fois les interactions visibles pour les utilisateurs et les processus internes exécutés par le système. L'objectif est de fournir un cadre clair des fonctionnalités attendues avant d'implémenter les diagrammes, codes ou interfaces.

3.1 Acteurs et Cas d'Utilisation

Le système identifie deux types d'acteurs principaux :

- **Client (Visiteur) :** L'utilisateur final qui interagit avec le chatbot pour obtenir des informations sur les produits, comparer des articles ou vérifier leur disponibilité. Les cas d'utilisation typiques incluent :
  - Demander le prix d'un produit.
  - Comparer deux produits selon leurs caractéristiques.
  - Vérifier la disponibilité en stock d'un article.
  - Obtenir des conseils d'utilisation ou recommandations personnalisées.
- **Administrateur (Société) :** Responsable de la gestion et de la maintenance du système. Les cas d'utilisation comprennent :
  - Mettre à jour le catalogue produit via ingestion de documents et de données (PDF, JSON, API Odoo).
  - Consulter les logs de conversation et analyser l'activité pour optimiser le chatbot.
  - Superviser la performance et la disponibilité du système.

3.2 Le Pipeline RAG (Cœur du Système)

Le cœur fonctionnel du chatbot repose sur le paradigme *Retrieval-Augmented Generation* (RAG), combinant récupération d'informations et génération de réponses. Les étapes principales sont :

1. **Fonction d'Ingestion :** Le système lit différentes sources de données (PDF, fichiers JSON, API Odoo) et les segmente en portions exploitables (*chunking*) pour alimenter la base vectorielle.
2. **Fonction de Récupération (Retrieval) :** Lorsqu'une requête utilisateur est reçue, le système effectue une recherche sémantique dans la base vectorielle pour identifier les documents pertinents. Une approche hybride peut combiner la recherche par mots-clés et la recherche sémantique afin d'améliorer la précision.
3. **Fonction de Génération :** Les informations récupérées sont injectées dans un prompt structuré pour le LLM, qui génère une réponse cohérente et contextuelle. Le

prompt peut inclure le rôle du chatbot, les informations extraites et les instructions pour citer correctement les sources.

3.3 Flux Conversationnels et Expérience Utilisateur (UX)

La qualité de l'expérience utilisateur repose sur plusieurs mécanismes :

— **Gestion du Contexte :** Le chatbot peut mémoriser temporairement les échanges récents pour répondre de manière cohérente et pertinente. Cela inclut la mémorisation de la question précédente ou du produit consulté.

— **Gestion des Erreurs (Fallback) :** Si le chatbot ne trouve pas de réponse adéquate, il peut afficher un message générique ou proposer de transférer la conversation à un agent humain.

— **Personnalisation de la réponse :** Le chatbot adapte le ton et le style de ses réponses selon le profil du client ou le type de produit.

3.4 Intégrations et Interfaces (API)

Le chatbot interagit avec plusieurs composants externes via des interfaces définies :

— **Interface Front-end :** Widget Chatbot intégré sur le site WordPress, utilisant HTML, JavaScript et CSS pour l'affichage et la capture des messages.

— **Connecteurs Back-end :**

  — Lecture de données depuis l'API Odoo pour obtenir des informations sur les produits et leur disponibilité.

  — Écriture des logs de conversation et statistiques dans une base de données ou fichiers internes pour l'analyse.

  — Possibilité future de connecter des modules supplémentaires, comme des API externes pour enrichir les réponses.

3.5 Scénarios Authentifiés vs Anonymes

Dans le cadre de notre chatbot, la présence d'un mécanisme d'authentification permet de distinguer deux types d'utilisateurs : le **visiteur anonyme** et le **client connecté**. Cette distinction a un impact direct sur les fonctionnalités offertes par le système ainsi que sur les contraintes de sécurité et de performance.

3.5.1 Fonctionnalités additionnelles pour les clients connectés

Lorsque l'utilisateur est authentifié, le chatbot bénéficie de fonctionnalités avancées :

— **Personnalisation contextuelle :** Le bot peut accueillir l'utilisateur par son prénom et se souvenir de ses interactions passées. Par exemple : "Bonjour Karim, que

puis-je pour vous ?". La mémoire contextuelle inclut les commandes passées, les préférences et les requêtes précédentes.

- **Accès différencié (Tiered Access) :**
  - *Visiteur :* Prix publics, stock approximatif.
  - *Client connecté :* Prix négociés, stock précis, accès à des documents techniques ou certificats de conformité réservés aux professionnels.
- **Fonctionnalités transactionnelles (Self-Care) :**
  - Suivi de commande (*Order Tracking*) via l'API Odoo.
  - Réapprovisionnement automatique (*Re-order*) pour commander des produits déjà acquis.
  - Téléchargement sécurisé des factures ou documents associés.

3.5.2 Contraintes non-fonctionnelles liées à l'authentification

L'accès à des informations privées nécessite des mesures supplémentaires de sécurité et de performance :

- **Sécurité et authentification :** Le front-end (WordPress) transmet un *Token sécurisé* (ex : JWT) au back-end Python afin de vérifier l'identité de l'utilisateur. Un contrôle d'accès basé sur les rôles (RBAC) est nécessaire pour garantir qu'un client ne puisse jamais accéder aux données d'un autre.
- **Confidentialité des données :** Les informations personnelles (PII) doivent être filtrées ou anonymisées avant d'être transmises à un modèle LLM externe (ex : OpenAI), conformément aux exigences RGPD.
- **Performance et latence :** Les requêtes personnalisées nécessitent des appels supplémentaires à la base de données (Odoo), ce qui peut augmenter légèrement le temps de réponse. Une tolérance de latence est spécifiée (ex : < 3 secondes pour une réponse personnalisée) afin de maintenir une expérience utilisateur fluide.

3.5.3 Synthèse des scénarios utilisateur

Le tableau suivant illustre la distinction entre un visiteur anonyme et un client connecté :

Fonctionnalité	**Utilisateur Visiteur**	**Utilisateur Client (Logué)**
Accueil	Générique	Personnalisé (prénom)
Catalogue	Produits standards	Produits + Offres exclusives
Prix	Public	Négocié (Tarif Pro)
Stock	Approximatif	Précis
Support	FAQ générale	Suivi de commandes, factures, réapprovisionne
Prompt système	"Tu es un assistant commercial..."	"Tu es l'account manager de [Client]..."

TABLE 1 – Comparaison des fonctionnalités selon le type d'utilisateur

3.5.4 Implication technique

Dans le back-end Python, la variable `user_context` permet de déterminer le mode d'opération du chatbot :

- Si `user_context` est vide : mode visiteur.
- Si `user_context` contient des informations utilisateur : mode client connecté, permettant d'injecter les données personnelles dans le prompt avant d'interroger le LLM.

Exemple de prompt injecté :

> "Tu réponds à M. Alaoui. Il a acheté une Pompe à Chaleur modèle X il y a 3 mois. Prends cela en compte dans tes réponses."

Cette approche garantit à la fois la **personnalisation**, le **respect de la confidentialité** et la **sécurité** tout en maintenant un flux conversationnel fluide.

4 Spécifications Non Fonctionnelles

Cette section décrit les contraintes de qualité et les exigences techniques du chatbot. Elle définit les critères de performance, de sécurité, de fiabilité et de maintenabilité nécessaires pour garantir un service robuste et conforme aux besoins de Flormar.

4.1 Performance et Latence

La performance du chatbot est cruciale pour l'expérience utilisateur. Les objectifs principaux sont :

- **Temps de réponse :** Le chatbot doit commencer à générer une réponse en moins de 3 secondes pour garantir une interaction fluide.
- **Débit (Throughput) :** Le système doit pouvoir supporter plusieurs requêtes simultanées sans dégradation significative, en fonction de la capacité du serveur Python (FastAPI).
- **Streaming :** La génération de texte en flux continu (*streaming*) permet à l'utilisateur de recevoir la réponse dès le début de la production du texte.

4.2 Qualité et Fiabilité des Réponses

La fiabilité des réponses générées est essentielle pour la crédibilité du chatbot :

- **Taux d'Hallucination :** Le chatbot doit éviter de générer des informations incorrectes ou inventées. Un prompt explicite comme « Si vous ne savez pas, indiquez que vous ne savez pas » sera intégré.

— **Précision du Retrieval :** La récupération des documents doit être pertinente. La précision Top-k et la qualité des embeddings vectoriels sont vérifiées pour garantir des réponses correctes.
— **Constance :** Le chatbot doit fournir des réponses cohérentes sur des questions similaires.

4.3 Sécurité et Confidentialité

La sécurité et la protection des données sont critiques, en particulier dans un environnement professionnel :

— **Anonymisation :** Les données personnelles (PII) ne doivent pas être envoyées à un LLM public ou non sécurisé.
— **Protection contre l'injection de prompt :** Le système doit être capable de gérer les tentatives malveillantes d'influencer le comportement du chatbot.
— **Cloisonnement :** Les informations sensibles (RH, comptabilité) doivent rester isolées et accessibles uniquement aux administrateurs autorisés.

4.4 Maintenabilité et Scalabilité

Le système doit être flexible et évolutif :

— **Architecture Modulaire :** Permettre de remplacer ou de mettre à jour le modèle LLM (ex. passer d'OpenAI GPT à Llama 3) sans réécrire l'ensemble du code.
— **Mise à jour des données :** La base vectorielle (ChromaDB) doit pouvoir croître avec l'ajout de nouveaux documents sans affecter la performance globale.
— **Extensibilité :** Possibilité d'ajouter de nouveaux connecteurs API ou sources de données pour enrichir les réponses.

4.5 Accessibilité et Contraintes Techniques

L'accessibilité et la compatibilité avec différents environnements garantissent une utilisation optimale :

— **Support Multi-device :** Le chatbot doit être compatible avec desktop et mobile, et s'adapter aux différentes tailles d'écran.
— **Disponibilité du service (SLA) :** Le système doit prévoir un plan de continuité si le service LLM est indisponible, incluant des messages de fallback ou la redirection vers un agent humain.
— **Compatibilité des navigateurs :** Le widget WordPress doit fonctionner sur les principaux navigateurs modernes (Chrome, Firefox, Edge, Safari).

4.6 Sécurité et Confidentialité Avancée pour Utilisateurs Authentifiés

Lorsqu'un utilisateur est connecté au système, le chatbot manipule des données personnelles et professionnelles sensibles. Il est donc crucial d'adopter des mesures de sécurité renforcées et de garantir la confidentialité des informations traitées. Ce sous-point décrit les exigences principales pour assurer la protection des utilisateurs logués.

- **Gestion des Sessions Sécurisées :** Le front-end (WordPress) doit transmettre un token sécurisé à l'API Python pour authentifier l'utilisateur. Chaque requête personnalisée du chatbot doit vérifier la validité du token avant toute action.
- **Contrôle d'Accès basé sur les Rôles (RBAC) :** Le système doit empêcher qu'un utilisateur accède aux données d'un autre. Par exemple, un client A ne pourra jamais consulter les commandes, factures ou documents techniques réservés au client B, même en tentant de contourner le chatbot.
- **Filtrage et Protection des Données Personnelles (PII) :** Avant d'envoyer des données à un LLM externe (ex. : OpenAI), toutes les informations sensibles comme le nom, l'adresse, les coordonnées ou historiques d'achats doivent être filtrées ou anonymisées, sauf si un accord légal explicite autorise leur transmission.
- **Audit et Traçabilité :** Chaque requête personnalisée, modification ou consultation de données sensibles doit être enregistrée pour assurer un suivi et permettre des audits de conformité (RGPD, normes ISO/IEC 27001). Ces logs doivent être sécurisés et accessibles uniquement aux administrateurs habilités.
- **Prompt System Personnalisé :** Le chatbot doit injecter le contexte utilisateur uniquement après authentification. Par exemple :

  > "Tu réponds à M. Alaoui. Il a acheté une Pompe à Chaleur modèle X il y a 3 mois. Prends cela en compte dans tes réponses."

  Cette personnalisation permet au bot de fournir des réponses pertinentes tout en respectant la confidentialité.

5 Étude Comparative et Choix Technologiques

5.1 ERP et Front-End : WordPress vs Concurrents

5.1.1 Contexte

Le projet nécessite un système combinant plusieurs composantes :

- Une interface client (front-end) pour le catalogue et le chatbot.
- Un ERP pour la gestion des stocks, commandes et clients.
- Une API accessible pour le chatbot RAG.
- Une solution modulable et économique pour une PME.

WordPress peut servir de **point de départ pour le front-end** grâce à son écosystème, sa simplicité et sa compatibilité avec des plugins tels que WooCommerce. Cependant, un ERP complémentaire est nécessaire pour gérer efficacement le back-office.

5.1.2 Principaux concurrents et comparaison avec WordPress

Solution	Avantages	Limites / Inconvénients	Comparaison avec WordPress
WordPress + WooCommerce	Interface connue, plugins pour chatbot et catalogue, hébergement mutualisé possible (Hostinger), faible coût initial	Fonctionnalités ERP limitées, gestion de stocks et comptabilité simplifiée, API complexe à maintenir	Point de référence pour front-end et e-commerce léger, mais insuffisant pour ERP complet
Odoo	Open source, modulaire, codé en Python, API XML-RPC et REST, large communauté	Certaines fonctionnalités payantes, nécessite serveur dédié pour gros volumes	Complète WordPress côté back-office, intégration native avec Python et chatbot RAG, modularité et ERP complet
SAP / Oracle NetSuite	ERP très complet, support entreprise, fonctionnalités avancées (finance, supply chain)	Très cher, configuration complexe, difficile à interfacer rapidement avec un bot, licences lourdes	Trop rigide et coûteux pour PME, pas adapté pour intégration front-end rapide
Microsoft Dynamics 365	Intégration avec outils Microsoft (Excel, Teams, SharePoint), cloud hybride	Coût élevé, customisation complexe, API parfois lourdes	Plus lourd et complexe que WordPress, moins agile pour prototypage rapide et chatbot RAG
Excel / Google Sheets	Facile à utiliser, peu coûteux, rapide à mettre en place	Pas de gestion en temps réel, erreurs humaines fréquentes, pas d'API native	Prototype uniquement, très léger, non évolutif et non compatible RAG pour chatbot

TABLE 2 – Comparaison des solutions ERP et Front-End avec WordPress comme référence

5.1.3 Analyse comparative

1. **Coût et accessibilité** : WordPress + WooCommerce reste le plus économique pour le front-end et le chatbot, tandis qu'Odoo apporte un ERP complet open source. SAP, Oracle et Dynamics sont trop chers pour une PME. Excel/Sheets ne sont adaptés qu'aux prototypes.

2. **Compatibilité technique avec le chatbot** : WordPress offre le front-end et un widget chatbot simple. Odoo fournit les API XML-RPC/REST pour RAG et une intégration Python directe. Les autres ERP nécessitent des adaptateurs complexes.
3. **Modularité et scalabilité** : WordPress est modulaire grâce aux plugins, Odoo est scalable et modulaire pour le back-office. Les concurrents sont rigides et souvent surdimensionnés.
4. **Intégration rapide pour un projet RAG** : La combinaison WordPress + Odoo permet une mise en place rapide et cohérente pour le front-end et le back-office. Les solutions concurrentes impliquent un développement plus long et coûteux.

5.1.4 Conclusion provisoire

— **Front-end recommandé** : WordPress/WooCommerce pour la partie visible client et le chatbot.
— **ERP recommandé** : Odoo pour la gestion des stocks, commandes et clients, et pour fournir les API nécessaires au chatbot RAG.
— Les autres ERP (SAP, Oracle, Dynamics) sont surdimensionnés ou trop coûteux pour une PME.
— Excel/Sheets restent uniquement des outils de prototype ou pour de très petites structures.

Points clés à retenir :
— WordPress = interface front-end + widget chatbot facile à déployer.
— Odoo = ERP complet, compatible Python et RAG.
— Concurrents = coûteux, complexes, moins adaptés à une PME.
— Excel/Sheets = prototype uniquement.

6 Solutions LLM : Le dilemme Performance vs Confidentialité

Pour notre chatbot RAG, le choix du modèle de langage est crucial. Il s'agit de trouver un équilibre entre la performance et la qualité de génération, la confidentialité des données, et la scalabilité du système.

6.1 Option A : Modèles propriétaires (SaaS via API)

Exemples : OpenAI GPT-4o, Claude 3.5 Sonnet (Anthropic).

Avantages :
— Intelligence supérieure et capacités conversationnelles avancées.
— Aucun serveur local à gérer.

— Mise en place rapide pour un prototype.
— Optimisation automatique et continue par le fournisseur.

Inconvénients :

— Données envoyées vers les serveurs du fournisseur (souvent hors UE).
— Coût variable par token ou par usage.
— Moins de contrôle sur les mises à jour et les versions du modèle.

Cas d'utilisation :

— Idéal pour le prototypage rapide du chatbot.
— Permet d'obtenir un MVP fonctionnel et performant avec RAG.
— Facilite l'expérimentation sur différents types de prompts et chaînes d'agents.

6.2 Option B : Modèles Open-Source (Self-Hosted)

Exemples : Llama 3 (Meta), Mistral (France), Gemma (Google).

Avantages :

— Souveraineté des données : toutes les informations restent sur l'infrastructure locale.
— Coût initial limité, pas de facturation par token.
— Flexibilité pour modifier et adapter le modèle.
— Compatible avec RAG pour mettre à jour les connaissances sans réentrainement complet.

Inconvénients :

— Nécessite des ressources serveur importantes (GPU, RAM).
— Maintenance et mise à jour plus complexes.
— Peut être moins performant que les modèles SaaS en termes de fluidité ou compréhension.

Cas d'utilisation :

— Solution à moyen/long terme pour garantir la confidentialité totale.
— Compatible avec l'approche modulable Python et LangChain.
— Permet de sécuriser l'accès aux données sensibles (RH, clients).

6.3 Justification stratégique

— Choix initial : OpenAI via API pour prototypage rapide et haute qualité.
— Évolution possible : bascule sur Llama 3 ou un autre modèle local dès que l'infrastructure le permet.
— Architecture hybride : l'utilisation de LangChain et d'un RAG permet d'abstraire le modèle. Le même code Python fonctionne avec différents LLM sans réécriture.

Points clés à retenir :

1. Performance immédiate vs souveraineté des données.

2. Flexibilité et évolutivité grâce à l'approche RAG + LangChain.
3. Possibilité de déploiement hybride : SaaS pour test, Open-Source pour production sécurisée.
4. Cohérence avec le choix d'Odoo et WordPress : tout reste modulable, évolutif et maintenable.

Critère	**Modèles Propriétaires (SaaS, ex : OpenAI GPT-4o)**	**Modèles Open-Source (Self-Hosted, ex : Llama 3)**
Performance	Très performants, génération fluide et compréhension avancée	Bonnes performances, mais légèrement inférieures aux SaaS pour certains cas complexes
Confidentialité	Données envoyées vers les serveurs du fournisseur (souvent hors UE)	Données restent sur serveur local, souveraineté totale
Coût	Facturation par token ou usage, variable selon le volume	Gratuit (hors coût serveur et GPU), coût fixe infrastructure
Maintenance	Maintenance et mises à jour assurées par le fournisseur	Maintenance complexe, nécessite suivi des mises à jour et GPU performant
Déploiement	Très rapide, idéal pour prototypage	Nécessite installation, configuration et ressources locales
Flexibilité	Limitée aux options du fournisseur	Hautement flexible, possibilité d'adaptation et d'intégration personnalisée
Cas d'utilisation	Prototypage rapide, tests sur prompts et chaînes RAG	Production sécurisée, respect de la confidentialité, intégration locale avec Odoo/WordPress

TABLE 3 – Comparatif des Solutions LLM pour le Chatbot RAG