

Dédicaces

*Nous dédions ce projet à toutes les personnes qui ont contribué à sa réalisation.*

*Tout d'abord, nous le dédions à nos encadrants, en reconnaissance de leur soutien indéfectible, de leurs précieux conseils, et de la confiance qu'ils ont placée en nos capacités. Leur expertise et leurs encouragements ont été des moteurs essentiels tout au long de ce parcours.*

*Nous dédions également ce projet à nos familles respectives, dont l'amour, la patience, et la compréhension ont été notre source de force. Leur soutien moral et émotionnel nous a permis de surmonter les défis et de rester motivés.*

*Enfin, nous dédions ce projet à nos collègues, pour leur collaboration, leurs idées innovantes, et leur esprit d'équipe. Leur engagement et leur travail acharné ont grandement contribué au succès de ce projet.*

*À tous, nous dédions ce travail avec nos plus sincères remerciements*.

Remerciements

La réalisation de travail est due à l’aide précieuse et aux conseils reçus de la part des enseignants et encadrants, qui ont grandement contribué par leurs suggestions et critiques. Ainsi, nous tenons à exprimer notre gratitude et à présenter nos chaleureux remerciements à :

* Monsieur **MEKOUAR Khalid**, président et directeur pédagogique de **l’École Supérieure d'Ingénierie en Sciences Appliquées**.
* Monsieur **LAHMER Mohamed**, notre professeur encadrant, pour nous avoir supervisés tout au long de ce stage, pour ses conseils avisés, son soutien continu, et la bienveillante attention qu’il a accordée à notre stage
* Monsieur **CHENFOUR Noureddine**, professeur encadrant, pour ses conseils précieux, son dévouement, et son engagement à garantir la réussite de notre travail.
* À **l’École Supérieure d’Ingénierie en Sciences Appliquées (ESISA)**, qui nous a offert l’opportunité d’acquérir une formation professionnelle de qualité.
* **NTT DATA MAROC,** pour nous avoir donné l’opportunité d’effectuer notre stage dans un environnement stimulant.
* Monsieur **FRÉDÉRIC SABBAH**, Directeur Général et représentant de **NTT DATA MAROC** (TETOUAN), pour nous avoir accordé l’opportunité d’effectuer ce stage.
* Monsieur **NANNA BADRE**, Développeur Senior et notre encadrant, pour nous avoir donné l'opportunité d'acquérir des compétences et de les enrichir en acceptant de réaliser notre stage de fin d'études sous sa supervision chez **NTT DATA Maroc**. Nous le remercions également pour l'intérêt qu'il a porté à notre travail, pour ses encouragements et ses conseils. Aucune parole ne saurait exprimer notre sympathie, notre profonde et sincère gratitude.
* Madame **IBTISSAM BENSALEH**, Responsable administratif et financier chez **NTT DATA MAROC** (TETOUAN), pour son accueil chaleureux et ses encouragements ainsi pour son intérêt à dynamiser les jeunes étudiants au sein du monde professionnel.
* Monsieur **DANIEL LENTON**, fondateur et PDG de **Unify.ai** (startup spécialisée dans le benchmarking des LLM) pour ses précieux renseignements lors de notre consultation. Grâce à son expertise, il nous a orientés vers le meilleur modèle de langage pour notre projet, ce qui a considérablement facilité notre prise de décision et l'optimisation de nos ressources en intelligence artificielle.
* Monsieur **ZILONG WANG,** doctorant à **l'Université de Californie**, à **San Diego** (UCSD) pour sa brillante masterclass sur le Speculative **RAG** (Retrieval-Augmented Generation). Ses explications sur l’utilisation d’un modèle spécialisé pour générer des brouillons et d’un modèle généraliste pour la vérification nous ont apporté une compréhension précieuse et ont considérablement éclairé notre vision pour l’intégration de cette technique dans nos propres projets.

Résumé

Ce rapport présente le travail effectué dans le cadre d'un projet de fin d'études réalisé chez **NTT DATA**, pour l'obtention du Diplôme de l'ESISA BAC+3 en **Ingénierie Logicielle**.

Le rapport débute par une présentation détaillée de l'entreprise **NTT DATA**, dans laquelle le stage a été réalisé, en expliquant sa structure organisationnelle et ses principaux domaines d'activité. Par la suite, il présente notre projet, définissant son contexte, ses objectifs et finalement son architecture.

L'objectif du projet était de concevoir et développer un chatbot intégré utilisant l'architecture **RAG (Retrieval-Augmented Generation)** afin de fournir une assistance intelligente aux utilisateurs.

Le rapport poursuit avec une étude fonctionnelle qui détaille les besoins et exigences du projet. Ensuite, une étude conceptuelle est présentée avant de passer à l'implémentation technique de la solution développée.

Tout au long du rapport, la méthodologie adoptée durant les différentes phases de développement est mise en lumière.

En conclusion, il est précisé que le projet a permis de concevoir avec succès un chatbot intégré basé sur l'architecture **RAG**, offrant une assistance intelligente et efficace aux utilisateurs.

Abstract

This report presents the work carried out as part of an end-of-study project at **NTT DATA**, for the ESISA BAC+3 Diploma in **Software Engineering**.

The report begins with a detailed presentation of **NTT DATA**, the company where the internship was carried out, explaining its organizational structure and main areas of activity. It then presents our project, defining its context, objectives and finaly its architecture.

The aim of the project was to design and develop an integrated chatbot **using RAG (Retrieval-Augmented Generation)** architecture to provide intelligent assistance to users.

The report continues with a functional study detailing the project's needs and requirements. Next, a conceptual study is presented, before moving on to the technical implementation of the solution developed.

Throughout the report, the methodology adopted during the various development phases is highlighted.

In conclusion, it is stated that the project successfully designed an integrated chatbot based on **RAG** architecture, offering intelligent and efficient assistance to users.

Table des matières

[Dédicaces 1](#_Toc177437202)

[Remerciements 2](#_Toc177437203)

[Résumé 4](#_Toc177437204)

[Abstract 5](#_Toc177437205)

[Chapitre I : 14](#_Toc177437206)

[Présentation Générale 14](#_Toc177437207)

[Introduction 15](#_Toc177437208)

[I- Présentation de l’établissement d’accueil 15](#_Toc177437209)

[1. Présentation de NTT DATA 15](#_Toc177437210)

[1.1. Présentation Générale 15](#_Toc177437211)

[1.2. Domaine d’expertise de NTT DATA 16](#_Toc177437212)

[1.3. Valeurs de NTT DATA 16](#_Toc177437213)

[2. Présentation de NTT DATA Maroc 17](#_Toc177437214)

[2.1. L’Intégration de NTT DATA Maroc 17](#_Toc177437215)

[2.2. Les Ressources de NTT DATA MAROC 17](#_Toc177437216)

[2.3. Organigramme de NTT DATA MAROC 18](#_Toc177437217)

[2.4. Fiche d'identité de NTT DATA MAROC 19](#_Toc177437218)

[II- Exploration et Documentation Technique 19](#_Toc177437219)

[1. Introduction 19](#_Toc177437220)

[2. Introduction aux Architectures de Chatbots 20](#_Toc177437221)

[2.1. Définition d’un Chatbot 20](#_Toc177437222)

[2.2. Evolution des Architectures de Chatbots 20](#_Toc177437223)

[2.3. Architectures de Chatbots Traditionnels 20](#_Toc177437224)

[2.4. L'Émergence des Architectures Basées sur l'Intelligence Artificielle 21](#_Toc177437225)

[2.5. Comparaison des Différentes Architectures 21](#_Toc177437226)

[3. Le Traitement du Langage Naturel (NLP) 22](#_Toc177437227)

[3.1. Introduction au Traitement du Langage Naturel (NLP) 22](#_Toc177437228)

[3.2. Les Composantes Clés du NLP dans les Chatbots 22](#_Toc177437229)

[4. Présentation de l’Architecture RAG (Retrieval-Augmented Generation) 24](#_Toc177437230)

[4.1. Architecture RAG : Un Nouveau Paradigme pour les Chatbots 24](#_Toc177437231)

[4.2. Fonctionnement de l'Architecture RAG 25](#_Toc177437232)

[4.3. Avantages de l'Architecture RAG 25](#_Toc177437233)

[4.4. Exemples Concrets d'Utilisation de RAG 26](#_Toc177437234)

[4.5. Défis et Limites de l'Architecture RAG 26](#_Toc177437235)

[4.6. Perspectives d’Évolution de l’Architecture RAG 27](#_Toc177437236)

[4. Conception et Développement d’un Chatbot Intégré à l’Architecture RAG 27](#_Toc177437237)

[4.1. Phases de la Conception d’un Chatbot Intégré à l’Architecture RAG 27](#_Toc177437238)

[4.2. Technologies et Outils Utilisés 28](#_Toc177437239)

[4.3. Défis Techniques et Solutions 29](#_Toc177437240)

[4.4. Cas d’Utilisation et Perspectives Futures 30](#_Toc177437241)

[Conclusion 30](#_Toc177437242)

[III- Le Contexte du Projet du Stage 30](#_Toc177437243)

[1. Contexte et Importance du Projet 30](#_Toc177437244)

[1.1. Problématique 31](#_Toc177437245)

[1.2. Solution Proposée 31](#_Toc177437246)

[2. Objectifs du Projet 31](#_Toc177437247)

[3. Planification du Projet 31](#_Toc177437248)

[3.1. Etapes du projet 31](#_Toc177437249)

[3.2. Diagramme de Gantt 33](#_Toc177437250)

[Conclusion 33](#_Toc177437251)

[Chapitre II : 35](#_Toc177437252)

[Conception et Modélisation 35](#_Toc177437253)

[Introduction 36](#_Toc177437254)

[I- Choix des Outils et Logiciels 36](#_Toc177437255)

[1. Introduction à L’UML 36](#_Toc177437256)

[1.1. Définition et Objectifs de L’UML 36](#_Toc177437257)

[1.1.1. Définition 36](#_Toc177437258)

[1.1.2. Objectifs 36](#_Toc177437259)

[1.2. Avantages de l'UML dans le Projet 37](#_Toc177437260)

[2. Présentation de PlantUML 37](#_Toc177437261)

[2.1 Pourquoi choisir PlantUML ? 37](#_Toc177437262)

[2.2 Fonctionnalités Clés de PlantUML 38](#_Toc177437263)

[II- Conception et Modélisation du projet 38](#_Toc177437264)

[1. Spécification des besoins 38](#_Toc177437265)

[1.1. Identification des acteurs 38](#_Toc177437266)

[1.2. Besoins fonctionnels 39](#_Toc177437267)

[1.3. Besoins non-fonctionnels 39](#_Toc177437268)

[2. Modélisation des Cas d’Utilisation 40](#_Toc177437269)

[2.1. Diagramme de cas d’utilisation 40](#_Toc177437270)

[2.1.1. Diagramme de Cas d'Utilisation pour le développeur 40](#_Toc177437271)

[2.1.2. Diagramme de Cas d'Utilisation pour l’Utilisateur Final 41](#_Toc177437272)

[Conclusion 41](#_Toc177437273)

[3. Modélisation Dynamique et Statique 41](#_Toc177437274)

[3.1. Introduction aux Diagrammes UML dans la Conception du Système 41](#_Toc177437275)

[3.2. Modélisation Statique 42](#_Toc177437276)

[3.2.1. Diagramme de Classes 42](#_Toc177437277)

[3.2.2. Diagramme de Composants 43](#_Toc177437278)

[3.3. Modélisation Dynamique 44](#_Toc177437279)

[3.3.1. Diagramme de Séquence 44](#_Toc177437280)

[3.3.2. Diagramme d'Activité 46](#_Toc177437281)

[Conclusion 48](#_Toc177437282)

[Chapitre III : Etude Technologique 49](#_Toc177437283)

[Introduction 50](#_Toc177437284)

[I- Contexte Technologique et Choix des Outils 50](#_Toc177437285)

[1. Introduction 50](#_Toc177437286)

[2. Technologies et Outils Utilisés 51](#_Toc177437287)

[2.1. Langage de Programmation : Python 51](#_Toc177437288)

[2.1.1. Présentation de Python pour le Traitement du Langage Naturel (NLP) 51](#_Toc177437289)

[2.1.2. Modules Python utilisés 52](#_Toc177437290)

[2.1.3. Avantages de Python pour la gestion des modèles d'IA et des API 55](#_Toc177437291)

[2.2. Benchmarking des LLM 57](#_Toc177437292)

[2.2.1 Présentation de UNIFY.AI 58](#_Toc177437293)

[2.2.2 Processus de Benchmarking des LLM 58](#_Toc177437294)

[2.2.3 Intégration du modèle LLM via OpenAI API 59](#_Toc177437295)

[2.2.3.1 Fonctionnement de l'API OpenAI pour GPT-4o 60](#_Toc177437296)

[2.2.3.2 Étapes d’Intégration de GPT-4o dans le Pipeline RAG 60](#_Toc177437297)

[2.3. Environnement de Développement avec Lightning AI Studio 62](#_Toc177437298)

[2.3.1. Gestion Simplifiée des Ressources GPU et des Modules 62](#_Toc177437299)

[2.3.2. Optimisation de l’Utilisation des GPUs avec Lightning AI 64](#_Toc177437300)

[II. Architecture RAG : Détails Techniques et Implémentation 65](#_Toc177437301)

[1. Présentation de l'Architecture RAG (Retrieval-Augmented Generation) 65](#_Toc177437302)

[2. Étape de Récupération d'Information (Retrieval) 66](#_Toc177437303)

[2.1. Extraction des documents 66](#_Toc177437304)

[2.1.1. Utilisation de PyMuPDF pour la lecture et la segmentation des documents 67](#_Toc177437305)

[2.1.2. Prétraitement des documents et conversion en chunks de texte 67](#_Toc177437306)

[2.1.2.1. Segmentation des documents en chunks 67](#_Toc177437307)

[2.1.2.2. Tokenization des chunks 68](#_Toc177437308)

[2.1.2.3. Analyse des données sur les chunks 68](#_Toc177437309)

[2.1.2.4. Optimisation du chunking 68](#_Toc177437310)

[2.2. Encodage des Documents avec Sentence-Transformers 69](#_Toc177437311)

[2.2.1. Fonctionnement de l'Encodage dans BERT et Génération des embeddings 69](#_Toc177437312)

[2.2.2. Formule d’Encodage. 70](#_Toc177437313)

[2.2.3. Optimisation des Embeddings pour la Recherche Vectorielle 70](#_Toc177437314)

[2.3. Stockage et Indexation dans FAISS 71](#_Toc177437315)

[3. Étape de Génération 72](#_Toc177437316)

[3.1. Sélection de LLM optimale pour la génération de texte 73](#_Toc177437317)

[3.2. Optimisation des paramètres de génération pour une meilleure qualité de réponse 73](#_Toc177437318)

[3.3. Gestion des erreurs et optimisation du flux RAG 73](#_Toc177437319)

[3.4. Analyse de la latence et performances du pipeline RAG 74](#_Toc177437320)

[Conclusion 74](#_Toc177437321)

[CHAPITRE IV : Réalisation et Interaction 75](#_Toc177437322)

[I- Introduction à l'Interface Utilisateur de l'Application 76](#_Toc177437323)

[1. Objectifs et Description de l'Application 76](#_Toc177437324)

[2. Fonctionnalités de Streamlit utilisées dans l'application 77](#_Toc177437325)

[3. Présentation des éléments de l'interface 77](#_Toc177437326)

[4. Limitations et Contexte d'Utilisation du Chatbot 79](#_Toc177437327)

[4.1. Focalisation sur le Langage Python 79](#_Toc177437328)

[4.2. Personnalisation et Scalabilité du Chatbot 79](#_Toc177437329)

[5. Adaptation du Chatbot aux Différents Niveaux de Complexité des Questions 80](#_Toc177437330)

[Conclusion 81](#_Toc177437331)

[CHAPITRE V : Conclusion Générale 82](#_Toc177437332)

[Webologie 84](#_Toc177437333)

Liste des Abréveation

|  |  |
| --- | --- |
| Abréviation | Signification |
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| BERT | Bidirectional Encoder Representations from Transformers |
| FAQ | Frequently Asked Questions |
| GPT | Generative Pre-trained Transformer |
| LLM | Large Language Model |
| NLP | Natural Language Processing |
| PDF | Portable Document Format |
| PyMuPDF | Python binding for MuPDF (a PDF library) |
| RAG | Retrieval-Augmented Generation |
| GPU | Graphics Processing Unit |
| NLU | Natural Language Understanding |
| NTT DATA | Nippon Telegraph and Telephone Data |
| NER | Named Entity Recognition |
| FAISS | Facebook AI Similarity Search |
| PCA | Principal Component Analysis |
| k-NN | k-Nearest Neighbors |
| IVF | Inverted File System (used in FAISS for efficient search) |
| HNSW | Hierarchical Navigable Small World (a graph-based search algorithm) |

Liste de Figures

[Figure 1: NTT DATA Logo 14](#_Toc177180874)

[Figure 2 : Les domaines d'expertise de NTT DATA 15](#_Toc177180875)

[Figure 3 : Les Ressources de NTT DATA MAROC 16](#_Toc177180876)

[Figure 4: Organigramme de NTT DATA MAROC 17](#_Toc177180877)

[Figure 5 : Chatbot ELIZA 19](#_Toc177180878)

[Figure 6: Composantes Clés du NLP dans les Chatbots 22](#_Toc177180879)

[Figure 7 : Composantes Clés de l'Architecture RAG 24](#_Toc177180880)

[Figure 8 : Avantages de l'Architecture RAG 25](#_Toc177180881)

[Figure 9 : Logo de Jira 32](#_Toc177180882)

[Figure 10 : Diagramme de GANTT 33](#_Toc177180883)

[Figure 11 : Logo de l'UML 35](#_Toc177180884)

[Figure 12: Logo de PlantUML 36](#_Toc177180885)

[Figure 13 : Les Acteurs du chatbot 37](#_Toc177180886)

[Figure 14 : Diagramme de Cas d'Utilisation pour le développeur 39](#_Toc177180887)

[Figure 15 : Diagramme de Cas d'Utilisation pour l'Utilisateur Final 40](#_Toc177180888)

[Figure 16 : Diagramme de Classes 41](#_Toc177180889)

[Figure 17 : Diagramme de Composants 42](#_Toc177180890)

[Figure 18 : Diagramme de Séquence 43](#_Toc177180891)

[Figure 19 : Diagramme d'Activité 45](#_Toc177180892)

[Figure 20 : Logo Python 50](#_Toc177180893)

[Figure 21 : Logo de LangChain 51](#_Toc177180894)

[Figure 22 : Logo FAISS 52](#_Toc177180895)

[Figure 23 : Logo OpenAI 52](#_Toc177180896)

[Figure 24 : Logo Hugging Face 53](#_Toc177180897)

[Figure 25 : Logo PYMuPDF 53](#_Toc177180898)

[Figure 26 : Logo Streamlit 54](#_Toc177180899)

[Figure 27 : Logo Unify.ai 57](#_Toc177180900)

[Figure 28 : Les principaux Axes de performances chez Unify.ai 57](#_Toc177180901)

[Figure 29 : Fonctionnement de l'API OpenAI 59](#_Toc177180902)

[Figure 30 : RAG Pipeline 60](#_Toc177180903)

[Figure 31 : Logo Lightning AI 61](#_Toc177180904)

[Figure 32 : Les différentes points forts de Lightning AI Studio 61](#_Toc177180905)

[Figure 33 : Les différents Plugins de Lightning AI 62](#_Toc177180906)

[Figure 34 : Les différentes Ressources GPU que propose Lightning AI 62](#_Toc177180907)

[Figure 35 : RAG Architecture 65](#_Toc177180908)

[Figure 36 : Exemple de représentation de vecteurs dans un repère orthonormé 68](#_Toc177180909)

[Figure 37 : BERT Architecture 68](#_Toc177180910)

[Figure 38 : Étapes de Création des Embeddings avec BERT Transformer 69](#_Toc177180911)

[Figure 39 : Exemple de Représentation des embeddings à travers PCA 70](#_Toc177180912)

[Figure 40 : Représentation des Différentes Couches HNSW 70](#_Toc177180913)

[Figure 41 : Architecture de l'étape de Génération 71](#_Toc177180914)

[Figure 42 : Interface de PYBOT 75](#_Toc177180915)

[Figure 43 : Figure de champ de saisie de texte 76](#_Toc177180916)

[Figure 44 : Exemple de conversation entre le PYBOT et l’Utilisateur 77](#_Toc177180917)

[Figure 45 : Liste de Navigation PYBOT 77](#_Toc177180918)

[Figure 46 : Exemple de Limitation de PYBOT par rapport au contenu du prompt 78](#_Toc177180919)

[Figure 47 : Exemple de conversation débutante avec PYBOT 79](#_Toc177180920)

[Figure 48 : Exemple de conversation Intermédiaire avec PYBOT 79](#_Toc177180921)

[Figure 49 : Exemple de conversation Avancé avec PYBOT 80](#_Toc177180922)

Liste de Tables

[Table 1 : Fiche D'Identité de NTT DATA MAROC 18](#_Toc177180923)

[Table 2 : Comparaison des Différentes Architectures des Chatbots 20](#_Toc177180924)

[Table 3 : Tableau comparatif des performances des modèles 58](#_Toc177180925)

[Table 4 : Macro Analyse des Données exploitées 66](#_Toc177180926)

[Table 5 : Statistiques des données des documents 67](#_Toc177180927)

Liste d’Équation

[Equation 1 : BERT Function 70](#_Toc177180937)

[Equation 2 : Fonction Cosinus Similarité et Distance Euclidienne 71](#_Toc177180938)

Chapitre I :

Présentation Générale

## Introduction

Ce chapitre est consacré à la présentation de l'établissement d'accueil où on a réalisé notre stage, ainsi qu'à une introduction au projet développé au cours de cette période.

Nous détaillerons d'abord le contexte général de l'organisation, en passant en revue son historique, son organisation interne, ainsi que ses missions et activités principales. Ensuite, nous présenterons le cadre du projet, en définissant la problématique à résoudre, les objectifs à atteindre, et la méthodologie de gestion du projet.

# Présentation de l’établissement d’accueil

## Présentation de NTT DATA

### ****Présentation Générale****

A blue text on a white background

Description automatically generated

Figure 1: NTT DATA Logo

**NTT DATA** est une entreprise mondiale de services numériques et de conseil en informatique, spécialisée dans la transformation digitale. Basée à Tokyo et fondée en 1988, elle fait partie des dix plus grands prestataires de services informatiques au monde. L'entreprise compte plus de **150 000 employés** répartis dans **plus de 50 pays**, notamment aux États-Unis, en France, en Allemagne, et au Maroc.

L’entreprise propose une large gamme de services qui couvre le conseil en gestion des processus métier, les technologies avancées, et les services cloud. NTT DATA s’engage à accompagner les entreprises et les gouvernements dans leur transformation numérique, en leur fournissant des solutions innovantes et adaptées à leurs besoins spécifiques.

NTT DATA est classée 6eme plus grand fournisseur de services informatiques au monde. Cette position est basée à la fois sur la valeur de la marque et sur les revenus. L'entreprise a progressé, passant de la 8eme place lors des années précédentes, ce qui témoigne de sa croissance continue et de son leadership dans le secteur des services numériques et informatiques.

### Domaine d’expertise de NTT DATA

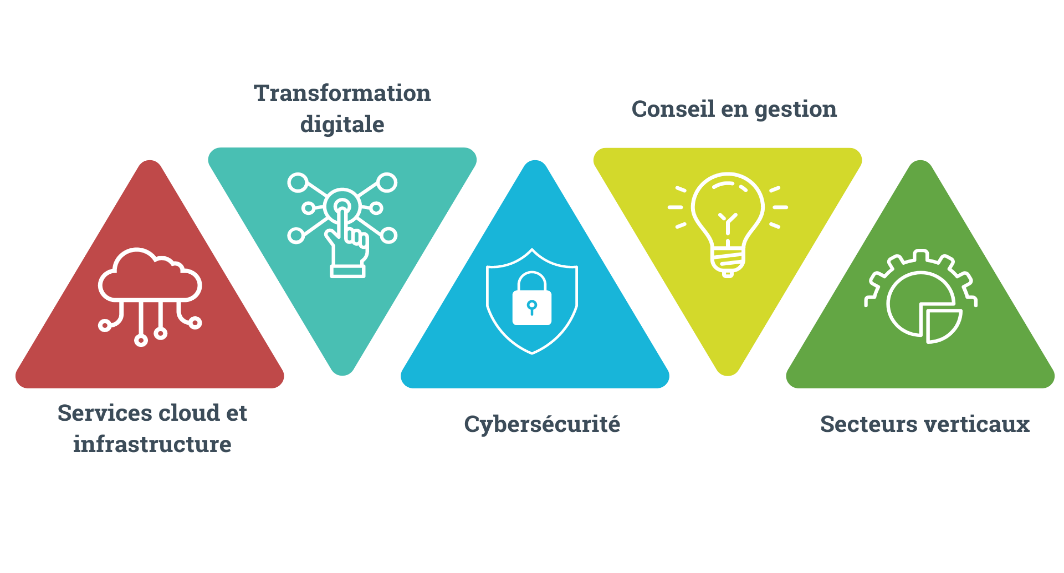


Figure 2 : Les domaines d'expertise de NTT DATA

**NTT DATA** intervient dans plusieurs domaines technologiques et industriels, offrant une expertise diversifiée adaptée aux différents secteurs économiques. Parmi ses principaux domaines d'expertise, on retrouve :

* **Services cloud et infrastructure** : NTT DATA propose des solutions cloud innovantes pour aider les entreprises à moderniser leur infrastructure, à réduire leurs coûts et à gagner en flexibilité.
* **Transformation digitale** : L'entreprise accompagne ses clients dans leur transformation numérique à travers l'intégration de technologies avancées telles que l'intelligence artificielle, l'automatisation et l'analyse de données.
* **Cybersécurité** : NTT DATA fournit des services de sécurité informatique pour protéger les infrastructures numériques des entreprises contre les cybermenaces et les attaques.
* **Conseil en gestion** : L'entreprise propose des services de conseil stratégiques pour aider les organisations à optimiser leurs processus et à améliorer leur efficacité opérationnelle.
* **Secteurs verticaux** : Grâce à une connaissance approfondie de secteurs spécifiques tels que la santé, la finance, l'industrie et le secteur public, NTT DATA propose des solutions personnalisées et adaptées aux besoins spécifiques de chaque industrie.

### Valeurs de NTT DATA

Les valeurs de **NTT DATA** sont au cœur de son engagement envers ses clients et la société. Elles guident son approche et ses actions pour créer des solutions durables et innovantes. Les valeurs clés de l’entreprise incluent :

* **Orientation client** : Placer les besoins des clients au centre de chaque projet et décision.
* **Innovation et anticipation** : Adopter une approche avant-gardiste en matière de technologies et de solutions, tout en anticipant les besoins futurs du marché.
* **Responsabilité sociale** : S'engager à promouvoir des pratiques éthiques et durables, notamment en matière d'environnement et d'énergie.
* **Diversité et inclusion** : Valoriser la diversité culturelle et promouvoir un environnement de travail inclusif où chacun peut s’épanouir.

## Présentation de NTT DATA Maroc

### L’Intégration de NTT DATA Maroc

L'un des objectifs clés de NTT DATA est de bâtir un environnement collaboratif et créatif, où l'innovation technologique se conjugue avec une forte connexion humaine. Son ambition est de créer des solutions qui ne se limitent pas à la technologie, mais qui améliorent véritablement la vie quotidienne de chacun.

**NTT DATA Maroc**, en tant que membre intégré de **NTT DATA EMEAL**, bénéficie des grands changements organisationnels qui ont eu lieu au sein de l'entité mondiale NTT DATA.

La fusion entre Everis et NTT DATA EMEAL a simplifié et renforcé l'organisation, favorisé une collaboration plus large et plus fluide, et promu une vision et des valeurs communes sous la marque unique NTT DATA. Cette intégration positionne NTT DATA comme la sixième plus grande entreprise de services informatiques au monde, offrant une proposition de valeur complète, allant du conseil en management à la modernisation digitale et informatique.

Au Maroc, NTT DATA était déjà présente à travers deux entités indépendantes, NTT DATA Services et Everis. Cette intégration renforce la position de NTT DATA Maroc en tant que leader du marché et ouvre de nouvelles opportunités en attirant de nouveaux talents. La filiale continuera de poursuivre ses activités dans le support, le service-desk, le développement et la maintenance applicative. En tant que NTT DATA, l'entreprise continuera de recruter activement, en maintenant sa politique de diversité et en offrant des perspectives de carrière et de développement personnel dans un environnement de plus en plus anglophone.

NTT DATA Maroc s'engage à développer ses talents et à devenir un acteur majeur sur le marché des technologies de l'information, tout en maintenant ses valeurs et en répondant aux exigences internationales. La transition vers NTT DATA ouvre de nouvelles perspectives passionnantes pour l'entreprise.

### Les Ressources de NTT DATA MAROC

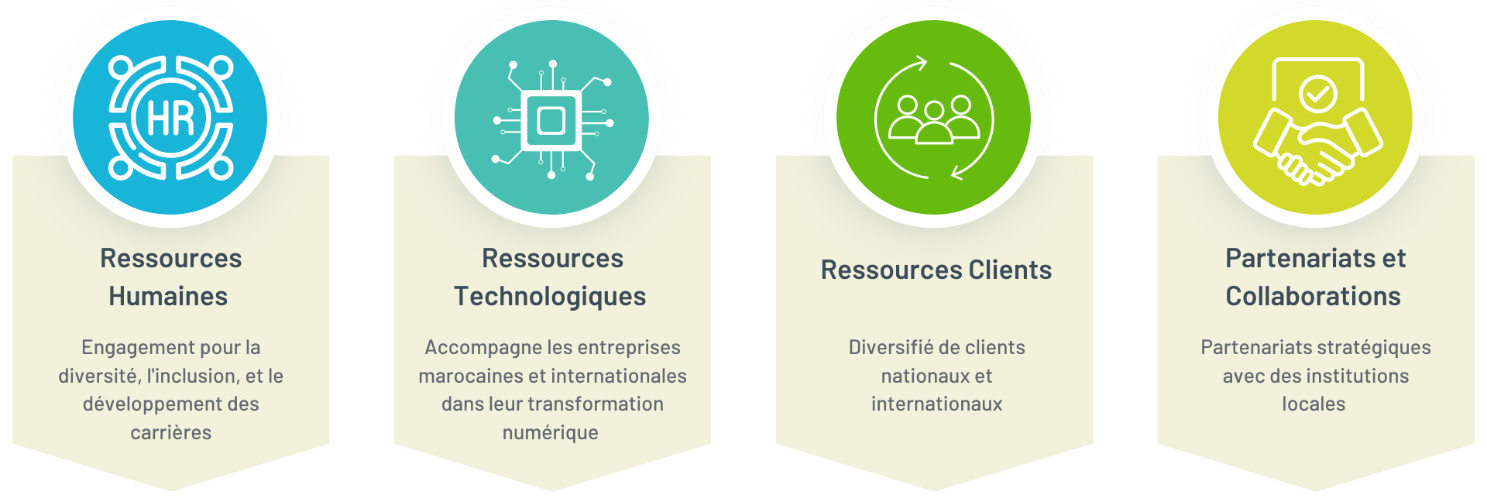


Figure 3 : Les Ressources de NTT DATA MAROC

NTT DATA Maroc est un acteur clé dans le domaine des services numériques et informatiques au Maroc. L'entreprise s'engage non seulement à fournir des solutions technologiques de pointe, mais aussi à jouer un rôle moteur dans la création d'emplois et le développement des talents locaux. Avec une forte présence dans plusieurs secteurs et un accent mis sur l'innovation, NTT DATA Maroc continue de renforcer son influence grâce à des ressources humaines, technologiques, et des partenariats stratégiques. Les principales ressources sont :

* **Ressources Humaines :** NTT DATA Maroc met l'accent sur la diversité de ses talents, avec plus de 500employés dans le domaine de l'informatique et l'objectif de créer 1 000 emplois supplémentaires d'ici 2025. Ces nouveaux postes incluent des profils en informatique, mais également dans l'externalisation des processus métiers (BPO). L'entreprise a également été reconnue comme Top Employer Global 2024, une distinction qui reflète son engagement pour la diversité, l'inclusion, et le développement des carrières​
* **Ressources Technologiques :** NTT DATA est à la pointe des technologies avancées, offrant des solutions en **intelligence artificielle (IA)**, **cloud computing**, **cybersécurité**, et **Internet des objets (IoT)**. Elle accompagne les entreprises marocaines et internationales dans leur transformation numérique en leur fournissant des **infrastructures** et des **solutions technologiques** adaptées aux défis actuels.

**Ressources Clients :** NTT DATA Maroc dispose d'un portefeuille diversifié de clients nationaux et internationaux, avec une expertise couvrant des secteurs variés tels que la finance, la santé, et l'industrie. L'entreprise fournit des services de support informatique, de gestion de la relation client, ainsi que des solutions d'externalisation des fonctions métiers (BPO) pour répondre aux besoins spécifiques de chaque client​.

* **Partenariats et Collaborations :** NTT DATA a noué des partenariats stratégiques avec des institutions locales et le gouvernement marocain. Ces collaborations permettent de développer les compétences des jeunes diplômés et d'encourager l'innovation technologique au Maroc. L'accès aux ressources mondiales du groupe NTT DATA renforce également la capacité de l'entreprise à répondre aux défis locaux tout en intégrant des solutions globales​.

### Organigramme de NTT DATA MAROC

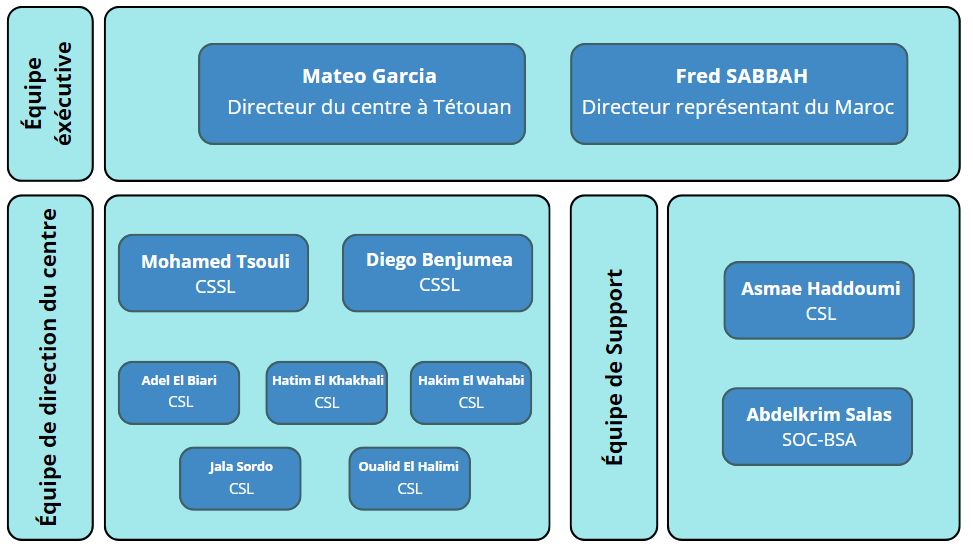
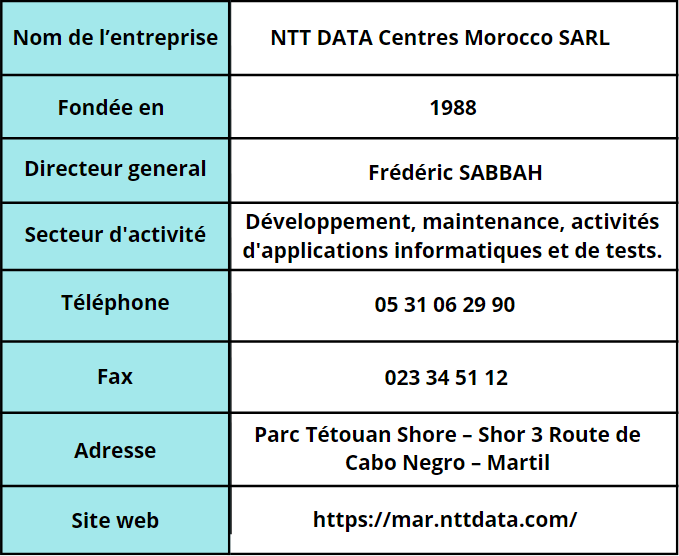
****

Figure 4: Organigramme de NTT DATA MAROC

L'organigramme de **NTT DATA Maroc** met en avant une structure claire, divisée en plusieurs   
équipes clés. **Mateo Garcia** occupe le poste de Directeur du centre à Tétouan, tandis que **Fred Sabbah** est le Directeur représentant du Maroc. L'équipe de direction du centre est composée de **Mohamed Tsouli**, **Diego Benjumea** , **Adel El Biari** , **Hatim El Khakhali** , **Hakim El Wahabi** , **Jala Sordo** , et **Oualid El Halimi** , tous responsables des activités des Centres de Services (CSSL et CSL). L'équipe de support inclut **Asmae Haddoumi** et **Abdelkrim Salas,** qui veillent au bon fonctionnement du centre.

### Fiche d'identité de NTT DATA MAROC

Table 1 : Fiche D'Identité de NTT DATA MAROC



**NTT DATA Centres Morocco SARL**, fondée en 1988, est une filiale spécialisée dans le développement, la maintenance, ainsi que dans les activités liées aux applications informatiques et aux tests. La société est dirigée par **Frédéric Sabbah** en tant que directeur général. Elle offre une gamme complète de services dans le secteur des technologies de l'information, avec un accent particulier sur la gestion et la maintenance des infrastructures informatiques. Les bureaux de NTT DATA Maroc (Tétouan) sont situés dans le **Parc Tétouan Shore – Shor 3 Route de Cabo Negro** à Martil

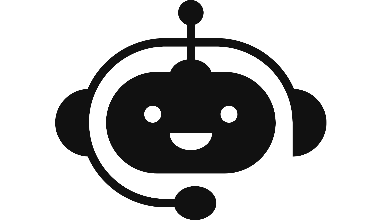
# Exploration et Documentation Technique

## Introduction

Avant de commencer le développement du chatbot, une analyse approfondie des technologies disponibles était essentielle. Le projet est basé sur l'intégration d'une architecture **RAG** (Retrieval-Augmented Generation**)**, qui combine des systèmes de récupération d'informations et des modèles de génération de langage naturel. Pour ce faire, plusieurs composants technologiques clés sont exploités. L'architecture RAG repose sur la capacité à interroger une base de connaissances pour récupérer les informations les plus pertinentes, avant de générer une réponse contextuelle à partir de celles-ci. Dans ce cadre, nous avons choisi d'utiliser des embeddings et transformers de Hugging Face pour transformer les données en vecteurs de manière efficace, ainsi que l'outil FAISS (Facebook AI Similarity Search) pour indexer et rechercher rapidement ces vecteurs. De plus, l'intégration du modèle de langage OpenAI (LLM) assure la partie génération du texte, qui est enrichie par les informations récupérées

## Introduction aux Architectures de Chatbots

### Définition d’un Chatbot

Un chatbot, aussi appelé "agent conversationnel", est un programme informatique conçu pour interagir avec les utilisateurs via une interface textuelle ou vocale. Ces systèmes simulent des conversations humaines et peuvent être utilisés pour accomplir des tâches variées telles que répondre à des questions, guider des utilisateurs à travers des processus, ou même effectuer des recommandations personnalisées. Les chatbots s'appuient sur des techniques de traitement du langage naturel (NLP) pour comprendre et formuler des réponses, tout en évoluant avec l'utilisation de technologies d’intelligence artificielle (IA) pour offrir des expériences de plus en plus complexes et interactives.

### Evolution des Architectures de Chatbots



Les architectures de chatbots ont évolué considérablement depuis leur création, passant de systèmes basés sur des scripts simples à des modèles complexes alimentés par l'IA. Les premières versions de chatbots, comme **ELIZA** dans les années 1960, étaient purement basées sur des règles prédéfinies. Au fil du temps, les architectures ont intégré des technologies avancées comme le traitement du langage naturel (NLP), le machine learning, et plus récemment des approches hybrides qui combinent récupération d'informations et génération de contenu, comme c’est le cas avec l'architecture **RAG (Retrieval-Augmented Generation)**.

Figure 5 : Chatbot ELIZA

### Architectures de Chatbots Traditionnels

Avant l'arrivée de l'intelligence artificielle avancée, les chatbots reposaient principalement sur deux types d'architectures : les chatbots basés sur des règles et les chatbots de correspondance de motifs.

* **Chatbots Basés sur des Règles** : Ces systèmes suivent des règles prédéfinies pour répondre aux utilisateurs. Chaque interaction avec l'utilisateur est régie par un ensemble de conditions programmées à l'avance. Cela les rend faciles à implémenter, mais extrêmement limités dans leur capacité à gérer des requêtes complexes ou des conversations non prévues.
* **Chatbots Basés sur la Correspondance de Motifs** : Ces chatbots identifient des motifs dans les requêtes des utilisateurs pour générer des réponses. Bien qu’ils soient plus flexibles que ceux basés sur des règles, ils souffrent d’un manque de compréhension réelle du langage et sont incapables de générer des réponses dynamiques.

Ces deux types d'architectures sont particulièrement utiles pour les tâches simples et répétitives, telles que la gestion de FAQ ou l’assistance dans des environnements très structurés.

### L'Émergence des Architectures Basées sur l'Intelligence Artificielle

Avec l'évolution des technologies de l'intelligence artificielle et du machine learning, les chatbots ont acquis une capacité à traiter des données complexes et à apprendre de leurs interactions avec les utilisateurs. Les chatbots basés sur l'IA utilisent des réseaux neuronaux pour comprendre les nuances du langage et générer des réponses plus naturelles. Ces chatbots peuvent :

* **Analyser le contexte** : Ils sont capables de maintenir une cohérence dans les conversations en tenant compte des interactions précédentes.
* **Apprendre et s'améliorer** : Grâce à l’apprentissage automatique, ils deviennent plus efficaces au fil du temps, en affinant leurs réponses selon les comportements des utilisateurs.
* **Traiter de grandes quantités de données** : L'IA permet aux chatbots d'exploiter des bases de données massives pour fournir des informations contextuelles et pertinentes en temps réel.

Parmi les architectures basées sur l’IA, le transformer est une innovation majeure, utilisée dans des modèles avancés tels que GPT (Generative Pre-trained Transformer), qui repose sur des réseaux neuronaux de type attention, capable de traiter efficacement de grandes séquences de texte.

### Comparaison des Différentes Architectures

Table 2 : Comparaison des Différentes Architectures des Chatbots

| **Type d'Architecture** | **Avantages** | **Inconvénients** |
| --- | --- | --- |
| **Basée sur des Règles** | Simple à implémenter, rapide pour les tâches définies. | Très limité, ne gère pas les requêtes complexes. |
| **Basée sur la Correspondance de Motifs** | Flexible par rapport aux règles fixes. | Ne comprend pas le contexte, ne génère pas de réponses naturelles. |
| **Basée sur l'IA (Modèles Transformeurs)** | Compréhension avancée du langage, capable de s'améliorer. | Nécessite de grandes quantités de données et de ressources de calcul. |
| **Architecture RAG** | Combinaison de récupération d'informations et de génération, réponses plus précises et contextualisées. | Complexe à implémenter, exigeante en ressources |

La *table 2* met en évidence les forces et faiblesses des différentes architectures de chatbots. Les chatbots basés sur des règles sont simples et rapides à implémenter, mais limités face aux requêtes complexes. Ceux basés sur la correspondance de motifs sont plus flexibles mais manquent de compréhension contextuelle et de naturel dans leurs réponses. Les modèles basés sur l'IA, comme les transformeurs, offrent une meilleure compréhension du langage et une capacité d'apprentissage, mais nécessitent des ressources importantes. Enfin, l'architecture RAG allie récupération d'informations et génération, offrant des réponses précises et contextuelles, bien qu'elle soit plus complexe à déployer et demande des ressources significatives.

## Le Traitement du Langage Naturel (NLP)

### Introduction au Traitement du Langage Naturel (NLP)

Le **traitement du langage naturel (NLP)** est une branche de l'intelligence artificielle qui permet aux machines de comprendre, interpréter et générer le langage humain. Il s'agit d'une composante essentielle dans la création des chatbots modernes, car elle leur permet de traiter les requêtes des utilisateurs de manière fluide et naturelle. Grâce au NLP, les machines peuvent analyser les mots, la grammaire, et le contexte d'une conversation pour générer des réponses pertinentes.

Dans le contexte des chatbots, le NLP joue un rôle clé en facilitant l'interaction homme-machine, en transformant le texte ou la parole en données exploitables par les systèmes d'IA. Les progrès dans ce domaine, notamment avec les réseaux de neurones profonds et les modèles de type **transformer**, ont considérablement amélioré la capacité des chatbots à fournir des réponses plus précises et contextuelles.

### Les Composantes Clés du NLP dans les Chatbots

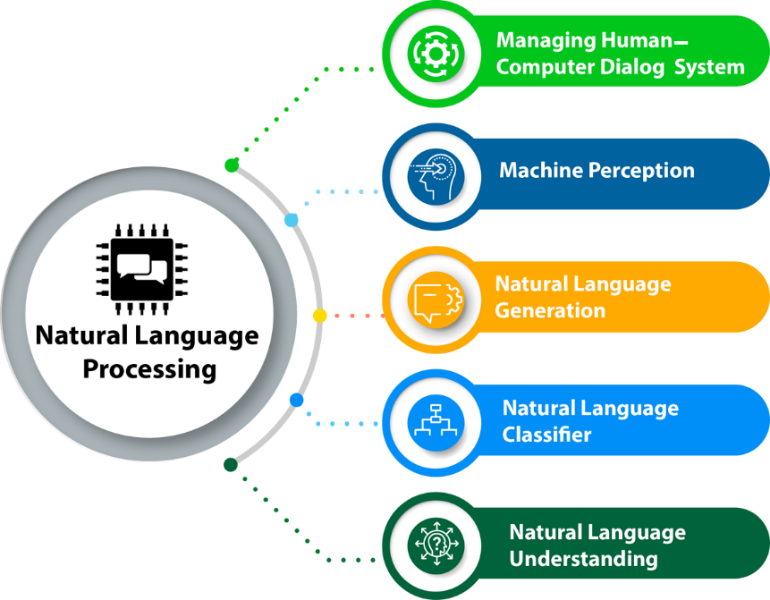


Figure 6: Composantes Clés du NLP dans les Chatbots

Le NLP repose sur plusieurs techniques et sous-disciplines pour comprendre et générer du langage naturel comme nous le démontre la *figure 6*. Voici les principales composantes impliquées dans le traitement des conversations des chatbots :

1. **Perception et Analyse Linguistique**

Le traitement du langage naturel (NLP) commence par la perception de la machine, qui permet au chatbot d'analyser et de comprendre les signaux linguistiques reçus (texte ou voix). Cela inclut plusieurs étapes importantes :

* + **Tokenisation** : Le texte est divisé en unités plus petites, appelées tokens, pour faciliter l’analyse. Chaque mot ou phrase devient ainsi plus facile à traiter indépendamment.
  + **Analyse Syntaxique (Parsing)** : Le chatbot examine la structure grammaticale de la phrase en identifiant les relations entre les mots. Cela permet de comprendre l'agencement des mots pour interpréter la signification globale.

Ces techniques constituent la base du traitement du langage dans les chatbots, permettant de convertir une entrée utilisateur en une structure exploitable par l'algorithme.

1. **Compréhension du Langage Naturel (NLU)**

La **compréhension du langage naturel (Natural Language Understanding - NLU)** est essentielle pour permettre au chatbot de saisir le sens des mots dans leur contexte et de comprendre les intentions de l’utilisateur :

* + **Reconnaissance des Entités Nommées (NER)** : Cette technique permet de détecter des entités spécifiques dans le texte, comme des noms de personnes, des lieux ou des dates. La reconnaissance d'entités aide à identifier les informations clés pour personnaliser la réponse.
  + **Compréhension Sémantique** : Au-delà de l'analyse syntaxique, la compréhension sémantique vise à identifier le sens profond des mots et l'intention réelle de l'utilisateur. Cela permet au chatbot d'interpréter correctement des requêtes complexes ou ambigües.

Ces techniques permettent au chatbot de comprendre le sens global de la conversation et de répondre de manière plus intelligente et pertinente.

1. **Désambiguïsation et Classification**

Le processus de désambiguïsation est crucial pour les chatbots, car les mots peuvent avoir des significations multiples selon le contexte. Le chatbot utilise des modèles NLP pour déterminer quelle interprétation d'un mot est la plus pertinente dans une phrase donnée.

* + **Classification des Requêtes (Natural Language Classifier)** : Une fois que la requête est comprise, elle doit être classée pour identifier l’intention de l’utilisateur. Cette classification aide à orienter la requête vers une réponse ou une action spécifique.

La désambiguïsation et la classification sont donc des étapes importantes pour éviter les erreurs de compréhension et garantir que le chatbot réponde correctement à la demande de l’utilisateur.

1. **Génération de Langage Naturel (NLG)**

Une fois que la requête a été comprise et analysée, la génération de langage naturel (Natural Language Generation - NLG) intervient pour formuler une réponse :

* + **Modèles de Génération Séquentielle** : Le chatbot génère une réponse en prédisant chaque mot successivement, en se basant sur le contexte de la conversation. Cela permet de créer des réponses fluides et naturelles.
  + **Personnalisation des Réponses** : Le chatbot peut personnaliser les réponses en fonction des informations fournies par l'utilisateur, rendant l'interaction plus engageante et pertinente.

La NLG permet ainsi au chatbot de fournir des réponses naturelles et cohérentes, imitant au mieux le dialogue humain.

1. **Gestion du Contexte** :

La gestion du contexte est une composante essentielle qui permet au chatbot de maintenir une conversation fluide et cohérente. En s'appuyant sur les interactions passées, le chatbot est capable de répondre de manière plus précise et en phase avec le fil de la discussion.

* **Modèles basés sur les transformeurs** : Des modèles avancés comme **GPT** permettent au chatbot de maintenir un dialogue cohérent sur plusieurs échanges, améliorant ainsi l’expérience utilisateur.

Grâce à la gestion du contexte, le chatbot est en mesure de suivre le fil des conversations longues, assurant ainsi une continuité et une pertinence dans ses réponses.

## Présentation de l’Architecture RAG (Retrieval-Augmented Generation)

### Architecture RAG : Un Nouveau Paradigme pour les Chatbots

L’architecture **RAG (Retrieval-Augmented Generation)** représente une avancée dans le domaine des chatbots intelligents. Contrairement aux chatbots traditionnels qui se contentent de générer des réponses à partir de ce qu’ils ont appris, l'architecture RAG permet aux chatbots de :

Figure 7 : Composantes Clés de l'Architecture RAG

* **Récupérer des informations en temps réel** à partir de bases de données ou de sources externes. Cela garantit que les réponses ne se limitent pas à ce qui est appris lors de l'entraînement du modèle, mais peuvent être enrichies par des informations actuelles et spécifiques à la requête.
* **Générer des réponses** en combinant ces informations récupérées avec une génération de texte basée sur un modèle de langage. Cette approche garantit que la réponse est à la fois contextuelle et précise.

Cette capacité à combiner récupération et génération donne aux chatbots basés sur RAG un avantage dans la gestion de requêtes complexes et dans la personnalisation des interactions avec les utilisateurs. L’intégration de l’architecture RAG dans le développement des chatbots est un élément clé pour optimiser l’assistance contextuelle, permettant aux chatbots d’offrir des expériences plus dynamiques et adaptées aux besoins de l’utilisateur.

### Fonctionnement de l'Architecture RAG

Le processus d’interaction avec un chatbot utilisant l'architecture RAG suit généralement plusieurs étapes successives :

1. **Entrée Utilisateur** : L'utilisateur formule une question ou une requête textuelle.
2. **Recherche d’Informations** : Le module de récupération analyse la requête de l’utilisateur et interroge une base de données ou une source de connaissances pour obtenir les documents les plus pertinents. Par exemple, dans un système d'assistance technique, il peut rechercher des manuels d'utilisation ou des rapports techniques.
3. **Génération de Réponses** : Le module de génération, généralement un modèle de type transformer, prend les documents récupérés et les utilise pour générer une réponse contextuelle. Il s'assure que l’information est présentée de manière cohérente, concise et adaptée à la requête.
4. **Réponse Utilisateur** : Le chatbot renvoie la réponse générée à l’utilisateur, souvent sous une forme textuelle ou vocale.

### Avantages de l'Architecture RAG

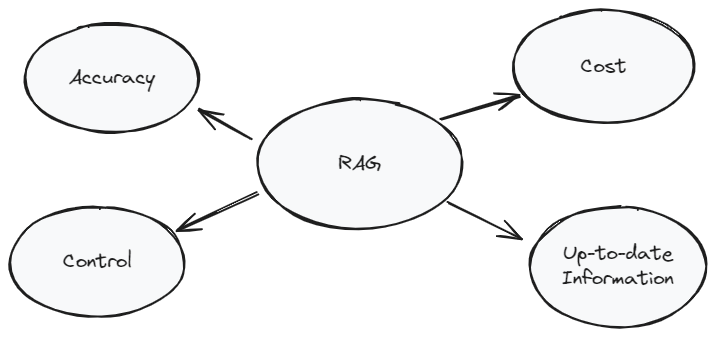


Figure 8 : Avantages de l'Architecture RAG

Comme illustré dans la *figure 8*, l’architecture RAG offre plusieurs avantages distincts par rapport aux architectures traditionnelles de chatbots et aux modèles de génération de texte autonomes :

1. **Amélioration de la Précision des Réponses** :
   * En combinant la récupération d’informations à la génération de texte, RAG permet de produire des réponses beaucoup plus précises, car le modèle utilise des données actuelles ou spécifiques.
   * Les chatbots traditionnels s'appuient sur des informations pré-apprises et peuvent parfois fournir des réponses dépassées ou incorrectes, ce qui est évité grâce à la récupération en temps réel dans RAG.
2. **Contextualisation Dynamique** :
   * L'architecture RAG est capable d’adapter ses réponses en fonction du contexte en récupérant des informations en temps réel, ce qui est essentiel pour des domaines comme le service client, la gestion des connaissances techniques, ou encore les assistants virtuels dans des environnements évolutifs.
3. **Réduction des Limites des Modèles de Génération** :
   * Les modèles de génération seuls, même avancés, sont limités par la taille de leur corpus de données d'entraînement. En s’appuyant sur des bases de connaissances externes via la récupération, RAG permet de surmonter ces limitations et de produire des réponses plus riches et plus diversifiées.
4. **Adaptabilité à Divers Cas d’Utilisation** :
   * RAG peut être utilisé dans un large éventail d'applications, allant des chatbots orientés vers l'assistance technique, où des informations spécifiques et détaillées sont requises, jusqu'aux assistants intelligents pour la recherche documentaire ou la recommandation de contenu.

### Exemples Concrets d'Utilisation de RAG

* **Service Client Technique** : Dans les centres de support client, un chatbot RAG peut interroger des bases de données internes pour fournir des réponses précises à des questions techniques spécifiques, comme le dépannage de matériel ou l’utilisation d’un logiciel.
* **Recherche Médicale** : Dans le domaine médical, un chatbot RAG peut récupérer des articles de recherche, des publications récentes, ou des informations sur des traitements, afin de fournir des recommandations basées sur des preuves à des professionnels de santé ou à des patients.
* **Assistance Juridique** : Un chatbot intégré à l'architecture RAG peut rechercher dans des bases de données légales et des textes réglementaires pour fournir des réponses détaillées à des questions juridiques complexes, en s’assurant que les informations sont à jour.

### Défis et Limites de l'Architecture RAG

Bien que l'architecture RAG offre de nombreux avantages, elle présente également certains défis techniques :

1. **Exigences en Ressources** :
   * L'architecture RAG nécessite des capacités de calcul significatives, notamment en raison de la récupération et de la génération simultanées de texte. Il est essentiel d’optimiser l’infrastructure pour que le chatbot puisse répondre rapidement, sans latence excessive.
2. **Gestion des Données** :
   * La qualité des réponses dépend fortement des sources de données utilisées pour la récupération. Si les bases de données sont obsolètes ou peu fiables, les réponses générées risquent d'être inexactes ou trompeuses.
3. **Complexité d’Implémentation** :
   * L'implémentation de l'architecture RAG dans un chatbot nécessite des compétences techniques avancées, tant dans la gestion des modèles de récupération que dans l’optimisation du module de génération de texte.
4. **Risques de Redondance** :
   * En récupérant des passages similaires dans différentes sources, il existe un risque de redondance ou d’incohérence dans les réponses générées. Une attention particulière doit être portée à la sélection et au traitement des documents récupérés.

### 4.6. Perspectives d’Évolution de l’Architecture RAG

L'architecture RAG est en constante évolution, notamment avec les avancées dans les modèles de génération et les techniques de récupération d'information. Les futures améliorations se concentreront probablement sur :

* **L’optimisation de la rapidité de récupération** pour réduire la latence et offrir des réponses encore plus instantanées.
* **L'intégration de modèles de génération plus sophistiqués**, capables de mieux comprendre le contexte et les nuances du langage.
* **La personnalisation accrue**, où le chatbot peut adapter les réponses non seulement en fonction de la requête, mais également des préférences et de l'historique des interactions de chaque utilisateur.

## Conception et Développement d’un Chatbot Intégré à l’Architecture RAG

### Phases de la Conception d’un Chatbot Intégré à l’Architecture RAG

La conception et le développement d'un chatbot intégré à l'architecture RAG suivent un ensemble de phases structurées, de la définition des objectifs fonctionnels au déploiement final. Voici les principales étapes de ce processus.

1. **Définition des Objectifs et des Cas d'Utilisation** :
   * La première étape consiste à définir clairement les objectifs du chatbot et les cas d'utilisation dans lesquels il sera déployé. Il est essentiel de comprendre les attentes des utilisateurs finaux et les types de requêtes auxquelles le chatbot devra répondre.
   * Exemples de cas d’utilisation : service client, support technique, recommandation de produits, FAQ dynamique, etc.
2. **Choix des Sources de Connaissances et des Bases de Données** :
   * Le succès d'un chatbot basé sur l'architecture RAG repose sur la qualité des informations qu'il peut récupérer. Il est donc crucial d’identifier et d’intégrer les bases de données et les sources externes pertinentes pour répondre aux besoins des utilisateurs.
   * Ces sources peuvent inclure des bases de connaissances internes, des documents techniques, des articles scientifiques, des FAQ, des manuels d’utilisation, ou des ressources externes (comme des API ou des services cloud).
   * **Création d'un index de recherche** : Une fois les sources identifiées, un index de recherche est créé pour permettre une récupération rapide et efficace des documents pertinents.
3. **Entraînement du Modèle de Langage** :
   * Le modèle de génération de texte (comme **GPT**) doit être entraîné sur des données textuelles variées et représentatives des domaines d’application du chatbot. Cet entraînement est crucial pour que le chatbot puisse générer des réponses naturelles et cohérentes en fonction des informations récupérées.
   * L'entraînement est basé sur des techniques de **machine learning** et de **deep learning**, où le modèle apprend à identifier des motifs dans les données textuelles et à produire des réponses en conséquence. Il est essentiel de veiller à ce que le modèle soit suffisamment général pour comprendre une large gamme de requêtes tout en étant spécifique aux données récupérées.
4. **Intégration de l'Architecture RAG** :
   * L’intégration de l’architecture RAG implique de relier le module de récupération d’informations et le module de génération de texte. Lorsque l’utilisateur pose une question, le chatbot commence par interroger l'index des connaissances (module de récupération) pour extraire des informations pertinentes.
   * Ces informations sont ensuite introduites dans le modèle de génération de texte, qui les transforme en une réponse cohérente et fluide, prête à être présentée à l’utilisateur.
5. **Optimisation et Ajustements** :
   * L'optimisation du chatbot implique plusieurs niveaux d’ajustements, notamment la mise à jour des bases de connaissances, le réentraînement du modèle pour s’adapter à de nouvelles requêtes, et la gestion du contexte conversationnel afin de garantir des réponses toujours pertinentes.
   * Il est également important d’ajuster les hyperparamètres du modèle pour maximiser la performance (précision des réponses, temps de latence, etc.).

### Technologies et Outils Utilisés

La conception d’un chatbot utilisant l’architecture RAG nécessite une combinaison de technologies de traitement du langage naturel (NLP), de gestion des données, et de machine learning. Voici un aperçu des principales technologies et outils utilisés :

1. **Frameworks de NLP** :
   * Des frameworks tels que spaCy, Hugging Face Transformers ou TensorFlow NLP sont utilisés pour entraîner les modèles de traitement du langage naturel. Ces frameworks facilitent l’entraînement de modèles de compréhension et de génération de langage naturel.
2. **Technologies de Recherche et d'Indexation** :
   * Pour la récupération des informations, des outils comme Elasticsearch ou Faiss sont couramment utilisés. Ils permettent de créer un index de recherche rapide et de récupérer les documents les plus pertinents en fonction des requêtes utilisateurs.
3. **Modèles de Langage Préentraînés (Transformers)** :
   * Les modèles basés sur les transformers (comme **GPT** ou **BERT**) sont au cœur de la génération de texte. Ces modèles sont préentraînés sur de larges ensembles de données et peuvent être ajustés spécifiquement pour s’adapter aux besoins du chatbot.
4. **API et Bases de Données** :
   * Les bases de données et les sources d’informations externes peuvent être connectées au chatbot via des API. Par exemple, des APIs externes peuvent fournir des informations en temps réel, comme des prix de produits, des prévisions météorologiques, ou des informations techniques à jour.

### Défis Techniques et Solutions

Le développement d’un chatbot basé sur l’architecture RAG comporte plusieurs défis techniques. Voici quelques-uns des principaux défis rencontrés et les solutions possibles :

1. **Qualité des Informations Récupérées** :
   * Le chatbot dépend de la qualité des informations qu’il récupère. Si les bases de données sont obsolètes ou inexactes, les réponses seront également erronées. Solution : Mettre à jour régulièrement les bases de connaissances et intégrer des filtres pour vérifier la pertinence des données récupérées.
2. **Latence de Récupération et de Génération** :
   * La récupération d'informations et la génération de réponses peuvent entraîner des temps de latence importants, ce qui dégrade l’expérience utilisateur. Solution : Optimiser l'indexation des données et utiliser des techniques de compression des modèles pour accélérer le temps de réponse.
3. **Gestion du Contexte** :
   * Maintenir le contexte au fil des échanges peut être difficile, surtout dans des conversations longues ou complexes. Solution : Intégrer des mécanismes de gestion du contexte, comme des modèles mémoire capables de suivre les interactions précédentes et de les intégrer dans les réponses futures.
4. **Surapprentissage et Généralisation** :
   * Un chatbot peut être trop spécifique aux données sur lesquelles il a été entraîné, rendant difficile sa généralisation à des requêtes nouvelles. Solution : Utiliser des techniques de régularisation et veiller à ce que le modèle soit suffisamment diversifié dans son entraînement.

### Cas d’Utilisation et Perspectives Futures

Un chatbot intégré à l'architecture RAG peut être déployé dans de nombreux secteurs et cas d’utilisation, notamment :

1. **Service Client et Support Technique** :
   * Le chatbot peut interroger des bases de données de support technique ou des manuels d'utilisation pour fournir des réponses précises aux clients sans avoir à consulter un humain.
2. **Assistants Personnels** :
   * Les chatbots RAG peuvent récupérer des informations en temps réel pour assister les utilisateurs dans leur prise de décision, comme recommander des produits, trouver des lieux, ou obtenir des informations sur les transports en commun.
3. **Recherche Médicale et Juridique** :
   * Ces chatbots peuvent interroger des articles scientifiques ou des bases de données juridiques pour fournir des informations spécialisées à des professionnels.

Dans le futur, l’architecture RAG pourrait être encore améliorée grâce à des innovations dans la gestion des données, l’amélioration des performances des modèles de génération de texte, et l'intégration de modèles capables de maintenir une conversation encore plus fluide et contextuelle.

## Conclusion

L’évolution des chatbots depuis les premières architectures jusqu'à l'intégration de l'architecture RAG met en évidence l'importance croissante du Traitement du Langage Naturel (NLP) pour améliorer l'interaction homme-machine. L'architecture RAG, en combinant récupération d'informations et génération de texte, permet de surmonter les limites des modèles traditionnels, en offrant des réponses plus précises et contextualisées. La conception d'un chatbot RAG exige une approche méthodique, intégrant des technologies avancées et un entraînement rigoureux. Ces avancées font des chatbots de nouvelle génération des outils puissants, adaptés à des usages variés, offrant une meilleure fluidité et pertinence dans les interactions.

# Le Contexte du Projet du Stage

## Contexte et Importance du Projet

Le projet s’inscrit dans une démarche visant à automatiser l’acquisition d’informations et l'apprentissage pour les développeurs Python, ainsi que pour les stagiaires souhaitant effectuer un stage dans le domaine des technologies Python. Avec l’essor rapide du langage Python dans de multiples secteurs, notamment l’intelligence artificielle, l’analyse de données, et le développement web, il est devenu essentiel de mettre en place des outils facilitant l'accès à des ressources pédagogiques et pratiques.

NTT DATA Maroc, étant une entreprise leader dans la transformation digitale, voit dans ce projet une opportunité de former efficacement les stagiaires et les développeurs Python, tout en réduisant les besoins en supervision manuelle. Cette automatisation permettra de faciliter l'intégration des stagiaires et de leur fournir un cadre d'apprentissage structuré, autonome et personnalisable.

### Problématique

La problématique à laquelle nous devions répondre concerne la gestion et la formation des stagiaires et développeurs Python. Actuellement, le processus d’apprentissage et d’acquisition de compétences repose sur des interventions humaines constantes, ce qui ralentit le développement des compétences des stagiaires et peut générer des lacunes dans la formation continue des développeurs. Le défi était donc de développer un système automatisé capable de fournir des ressources d'apprentissage en Python, tout en permettant aux stagiaires d’évoluer de manière autonome.

### Solution Proposée

La solution retenue consiste à mettre en place un chatbot utilisant l’architecture RAG, capable de fournir des informations et des ressources d'apprentissage sur Python de manière automatisée. Ce chatbot offrira des réponses contextuelles aux questions des stagiaires, tout en guidant leur progression dans l'apprentissage de Python. Grâce à ce système, les stagiaires pourront obtenir des réponses précises en temps réel, accéder à des tutoriels adaptés à leur niveau, et bénéficier de ressources d'apprentissage continues, sans intervention humaine directe.

## Objectifs du Projet

L’objectif était de concevoir un système innovant et automatisé pour faciliter l'acquisition de connaissances techniques en Python.

* Automatiser l’acquisition d’informations et l’apprentissage des développeurs Python et des stagiaires souhaitant approfondir leurs compétences en Python.
* Réduire la charge de travail des formateurs en permettant aux stagiaires d’accéder à des ressources d’apprentissage autonomes.
* Offrir un système évolutif qui puisse s’adapter aux différents niveaux d’expertise des utilisateurs (débutant, intermédiaire, avancé).

## **Planification du Projet**

Le projet a été développé en 2 mois, comprenant des phases bien définies pour garantir la qualité et l'efficacité du système. La planification a suivi des étapes rigoureuses pour intégrer des aspects techniques spécifiques à l'architecture RAG et à l'apprentissage automatisé.

### 3.1. Etapes du projet

**Mois 1 : Collection, Prétraitement et Prototypage**

1. **Collection des documents** :
   * Recueillir une grande base de données de documents liés à Python, y compris des fichiers PDF (tutoriels, articles techniques, documentation). Ces documents serviront de base de connaissances pour le chatbot.
2. **Prétraitement des documents** :
   * Convertir les documents recueillis en petits segments de texte pour les rendre plus facilement exploitables par les modèles de langage. Ce découpage garantit que les informations sont formatées pour la création d’embeddings et le processus de récupération.
3. **Création des embeddings** :
   * Transformer les chunks de texte en embeddings (représentations vectorielles) qui capturent le contenu sémantique du texte. Ces embeddings permettront de rechercher efficacement les passages pertinents lors des requêtes utilisateurs.

**Mois 2 : Stockage, Requêtes et Génération de Réponses**

1. **Stockage des embeddings** :
   * Stocker les embeddings générés dans une base de données vectorielle, comme FAISS, pour permettre une récupération rapide des passages pertinents des documents lors des requêtes utilisateurs.
2. **Traitement des requêtes** :
   * Transformer les requêtes utilisateurs en embeddings et les comparer avec les embeddings stockés dans FAISS. Cette comparaison permet de récupérer les passages de texte les plus pertinents par rapport à la question posée.
3. **Génération de réponses** :
   * Une fois les passages pertinents récupérés, un modèle de langage génère une réponse contextuellement appropriée à l'utilisateur, en s'appuyant sur les informations des passages trouvés.
4. **Interaction utilisateur** :
   * Fournir une interface utilisateur simple et intuitive à travers Streamlit, permettant aux utilisateurs de poser des questions sur Python et de recevoir des réponses générées par le chatbot. Cette interface gère également l'upload des documents PDF pour enrichir la base de connaissances.

### Diagramme de Gantt



Le diagramme de Gantt Comme représenté dans la *figure 10*, réalisé à l'aide de l'outil Jira, présente la planification temporelle du projet, mettant en évidence les différentes étapes de développement ainsi que les délais associés. Ce diagramme offre une vue d'ensemble claire et concise de la progression du projet, en permettant de visualiser les phases clés, les tâches spécifiques, et leur interdépendance.

Figure 9 : Logo de Jira

Chaque étape du projet a été soigneusement planifiée pour garantir une exécution ordonnée et efficace. Les barres du diagramme représentent la durée prévue pour chaque tâche, facilitant ainsi le suivi du respect des échéances et l'ajustement des ressources en fonction des besoins.



Figure 10 : Diagramme de GANTT

Cet outil a été essentiel pour assurer la coordination des différentes activités et pour garantir que toutes les phases du projet, de la conception à la mise en œuvre, se déroulent selon le calendrier prévu.

## Conclusion

En résumé, le projet **PYBOT** s'inscrit dans une démarche innovante visant à automatiser l'acquisition de connaissances pour les développeurs Python et les stagiaires. En combinant des technologies avancées comme l'architecture Retrieval-Augmented Generation (RAG) et des outils performants tels que FAISS et les modèles de génération de texte, le projet a permis de créer un système capable de fournir des réponses précises et adaptées aux besoins des utilisateurs.

Grâce à cette solution, l'apprentissage autonome est facilité, réduisant ainsi la dépendance aux formateurs tout en offrant un accompagnement personnalisé en fonction du niveau de compétence de chaque utilisateur. Ce projet constitue une avancée significative dans l'utilisation des technologies de l'intelligence artificielle pour améliorer les processus d'apprentissage et de formation, tout en garantissant une efficacité et une évolutivité à long terme.

Chapitre II :

Conception et Modélisation

## Introduction

La phase de conception est une étape essentielle du développement du chatbot **PYBOT** destiné à l'acquisition et à l'apprentissage automatisés pour les développeurs Python et les stagiaires. Cette phase permet de traduire les besoins identifiés en une architecture technique bien définie, assurant que le système soit performant, évolutif, et capable de s'adapter aux différents niveaux d'expertise des utilisateurs.

Ce chapitre se concentre sur la modélisation du projet à travers différents diagrammes UML, qui fournissent une représentation abstraite et organisée des éléments du système. Ces diagrammes seront utilisés comme base pour la communication entre les membres de l'équipe, facilitant ainsi la compréhension commune des fonctionnalités à développer et des interactions entre les composants du système.

Nous commencerons par la modélisation des cas d'utilisation, définissant les acteurs du système et leurs interactions avec le chatbot. Ensuite, nous passerons à la conception des séquences, qui détaillera l'ordre des interactions nécessaires pour réaliser les fonctionnalités clés, comme le traitement des requêtes et la génération de réponses. Ensuite, nous représenterons la structure statique du système à travers un diagramme de classes, avant de structurer ces classes en groupes logiques dans un diagramme de packages.

Enfin, nous réaliserons un prototypage sous forme de wireframes pour visualiser l'interface utilisateur du chatbot, garantissant une expérience fluide et intuitive pour les développeurs et stagiaires.

Ces outils de modélisation nous permettront de structurer efficacement le développement, en veillant à ce que chaque composant du système soit bien conçu et intégré de manière cohérente, afin de garantir une performance optimale du chatbot **PYBOT**.

# Choix des Outils et Logiciels

## Introduction à L’UML

### Définition et Objectifs de L’UML

#### Définition

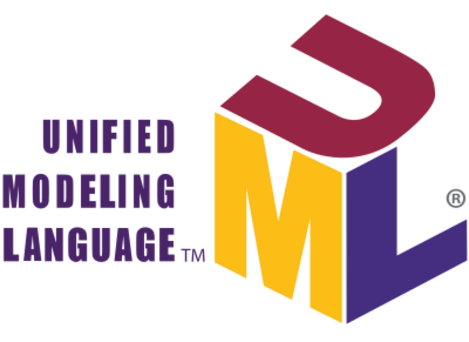
L'Unified Modeling Language (UML) est un langage de modélisation standardisé utilisé pour représenter de manière graphique les aspects structurels et comportementaux d'un système logiciel. Il permet de modéliser différents éléments du système, tels que les acteurs, les cas d’utilisation, les interactions entre objets, les classes, et plus encore.

Figure 11 : Logo de l'UML

#### Objectifs

ML vise à fournir un langage commun pour les développeurs, les architectes et les autres parties prenantes, facilitant ainsi la communication, la documentation, et la compréhension des systèmes complexes. Il aide à capturer les exigences, à concevoir des architectures robustes et à assurer la traçabilité des éléments du système tout au long du cycle de vie du développement logiciel.

### Avantages de l'UML dans le Projet

1. **Normalisation et standardisation** : UML est largement accepté et utilisé dans l'industrie du logiciel, offrant un langage commun aux développeurs, architectes, et autres parties prenantes. Cela assure que les diagrammes et les modèles sont compris de manière uniforme, réduisant les risques de malentendus.
2. **Polyvalence** : UML permet de modéliser divers aspects du système à travers plusieurs types de diagrammes, tels que les diagrammes de cas d’utilisation, de séquence, de classes, et de packages. Cette polyvalence permet de capturer à la fois la structure statique et le comportement dynamique du système.
3. **Documentation claire** : UML aide à créer une documentation visuelle claire et précise, facilitant la communication entre les membres de l'équipe et assurant une compréhension commune du projet. Cela est particulièrement utile pour les revues de code, les discussions sur la conception, et la formation des nouveaux membres de l'équipe

## Présentation de PlantUML

PlantUML est un outil de modélisation UML basé sur du texte. Il permet de créer facilement des diagrammes UML à partir d'une syntaxe simple et accessible. Cet outil est conçu pour faciliter la modélisation, la conception, la gestion et la documentation des systèmes logiciels complexes, tout en permettant une génération rapide de diagrammes UML. PlantUML prend en charge l'ensemble des diagrammes UML et s'intègre bien avec d'autres outils de développement, ce qui en fait un choix populaire pour les projets logiciels.

Figure 12: Logo de PlantUML

### Pourquoi choisir PlantUML ?

* **Support complet des diagrammes UML** : PlantUML prend en charge tous les types de diagrammes UML nécessaires à la modélisation complète d'un système, y compris les diagrammes de cas d'utilisation, de séquence, de classes et de packages. Cela permet une modélisation exhaustive et cohérente.
* **Simplicité et flexibilité** : Le logiciel utilise une syntaxe textuelle simple pour définir les diagrammes UML, ce qui rend la création rapide et flexible. Les utilisateurs peuvent facilement personnaliser les diagrammes et ajuster les modèles selon les besoins spécifiques du projet.
* **Collaboration et gestion via des fichiers texte** : En utilisant du texte pour la modélisation, PlantUML facilite la collaboration entre les membres de l'équipe en permettant un partage simple des fichiers de diagrammes. De plus, il s'intègre parfaitement avec les systèmes de contrôle de version tels que Git, facilitant ainsi la gestion des modifications et la coordination entre les développeurs, les architectes et les gestionnaires de projet.
* **Intégration et génération de diagrammes dans divers outils** : PlantUML s'intègre à de nombreux environnements de développement (comme les IDE) et plateformes, ce qui permet de générer rapidement des diagrammes UML dans le cadre d'un flux de travail de développement. Cela permet une visualisation rapide et précise des conceptions modélisées.

### Fonctionnalités Clés de PlantUML

* **Modélisation UML complète** : PlantUML prend en charge tous les types de diagrammes UML, ce qui permet une modélisation détaillée et complète du système.
* **Simplicité de génération des diagrammes** : Grâce à la syntaxe textuelle, la création et la modification des diagrammes sont rapides. Il est également possible d'automatiser la génération de diagrammes à partir de fichiers source ou de bases de données.
* **Intégration avec des systèmes de gestion de version** : L'utilisation de fichiers texte facilite la gestion des versions des diagrammes et leur intégration dans les processus de développement existants, comme les pipelines CI/CD.
* **Génération automatique de documentation** : PlantUML peut être utilisé pour générer automatiquement des diagrammes dans des documents, facilitant ainsi la création de documentation technique à jour, sans effort supplémentaire.

En résumé, le choix d'UML et de PlantUML comme outils de modélisation dans ce projet était basé sur leur capacité à fournir une représentation précise, standardisée et compréhensible des différents composants du système, tout en permettant une gestion efficace du projet et une collaboration fluide entre les parties prenantes.

# Conception et Modélisation du projet

## Spécification des besoins

La spécification des besoins vise à identifier et documenter les fonctionnalités ainsi que les contraintes du système à développer. Cette étape garantit que PyBot répondra aux attentes des utilisateurs et des parties prenantes. Cette section détaille les **besoins fonctionnels** et **non-fonctionnels**, constituant la base de la conception et du développement de PyBot.

### Identification des acteurs

Il existe deux principaux acteurs interagiront avec **PyBot** comme nous le démontre la *figure 13* :

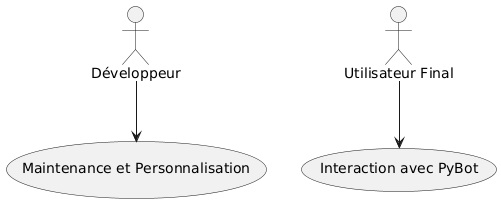
* **Développeur** : Le développeur est responsable de l'amélioration et de la maintenance de PyBot. Il peut personnaliser le comportement du chatbot en modifiant ses fonctionnalités et en testant les nouvelles mises à jour.
* **Utilisateur final** : L'utilisateur final interagit directement avec PyBot pour poser des questions et obtenir des réponses en langage naturel. Cet utilisateur n'a pas accès aux fonctionnalités de développement ou de gestion avancées.

Figure 13 : Les Acteurs du chatbot

### Besoins fonctionnels

Les besoins fonctionnels définissent les fonctionnalités spécifiques que **PyBot** doit offrir pour répondre aux attentes des utilisateurs. Les principaux besoins fonctionnels sont :

* **Interaction via PyBot** : Les utilisateurs doivent pouvoir poser des questions à PyBot en langage naturel. Le chatbot devra comprendre les requêtes et fournir des réponses pertinentes basées sur une base de connaissances ou d'autres ressources externes.
* **Personnalisation des réponses** : Le développeur doit être capable de personnaliser les réponses fournies par PyBot, en fonction des besoins spécifiques du domaine d'application. Cela inclut l'ajout ou la modification de réponses pour s'adapter aux questions les plus courantes ou aux changements de contexte.
* **Accès à des bases de connaissances** : PyBot doit pouvoir accéder à une base de connaissances configurée par le développeur pour enrichir ses réponses. Cela peut inclure des fichiers locaux ou des ressources externes, comme des APIs ou des systèmes de gestion de connaissances.

### Besoins non-fonctionnels

Les besoins non-fonctionnels spécifient les critères de performance, de sécurité, et autres contraintes que le système PyBot doit respecter. Les besoins non-fonctionnels incluent :

* **Sécurité des échanges** : Les interactions entre PyBot et les utilisateurs doivent être sécurisées, notamment lorsque PyBot accède à des ressources externes ou des données sensibles. Les échanges doivent être protégés contre toute fuite d'information.
* **Performance** : PyBot doit être capable de répondre rapidement aux requêtes des utilisateurs, même lorsque le volume de questions ou d'utilisateurs simultanés augmente.
* **Scalabilité** : Le système doit être capable de s'adapter à une augmentation du nombre de requêtes sans affecter la qualité ou la rapidité des réponses. Cela permettra de faire évoluer PyBot en fonction de l'usage.
* **Fiabilité** : PyBot doit être robuste et fonctionner de manière continue. Il doit avoir un temps d'arrêt minimal, et des mécanismes de récupération doivent être mis en place pour assurer la disponibilité du service.
* **Compatibilité avec des services externes** : PyBot doit pouvoir s'intégrer facilement avec des systèmes ou services externes, comme des APIs ou des bases de données, pour élargir ses capacités de réponse. Cela peut inclure des intégrations avec des outils de traitement du langage naturel ou des plateformes de gestion de contenu.

Dans cette version, le focus est mis sur les interactions utilisateur, la personnalisation des réponses, et la gestion des connaissances, tout en tenant compte des besoins non-fonctionnels comme la sécurité et la scalabilité.

## Modélisation des Cas d’Utilisation

Haut du formulaire

La modélisation des cas d'utilisation est une technique clé pour représenter les interactions entre les acteurs et le système. Elle permet de visualiser les différentes fonctionnalités que le système doit offrir et comment elles seront utilisées par les différents acteurs

### Diagramme de cas d’utilisation

Bas du formulaire

Le diagramme de cas d'utilisation est un outil fondamental pour comprendre les interactions entre les utilisateurs et le système. Les diagrammes ci-dessous (*Figures 14 et 15*) représentent les cas d'utilisation pour les deux acteurs principaux : Développeur, Utilisateur.

#### Diagramme de Cas d'Utilisation pour le développeur

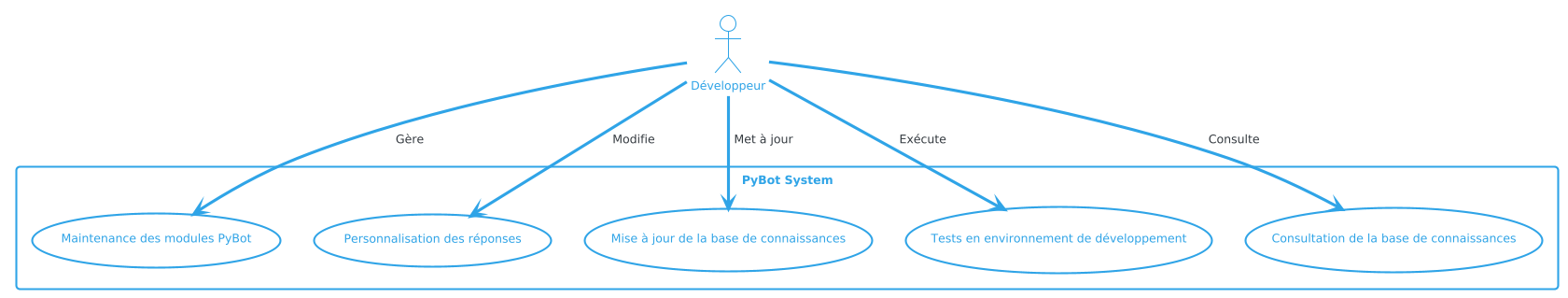


Figure 14 : Diagramme de Cas d'Utilisation pour le développeur

* **Développeur** : L'acteur "Développeur" est celui qui interagit avec les différents cas d'utilisation.
* **Maintenance des modules PyBot** : Le développeur gère la maintenance des modules qui composent PyBot, y compris les mises à jour, les correctifs et les nouvelles fonctionnalités.
* **Personnalisation des réponses** : Le développeur peut modifier les réponses que PyBot fournit aux utilisateurs finaux, en les adaptant aux besoins spécifiques de l'application ou du projet.
* **Mise à jour de la base de connaissances** : Le développeur peut mettre à jour la base de connaissances en ajoutant ou en retirant des documents ou d'autres ressources.
* **Tests en environnement de développement** : Avant la mise en production, le développeur effectue des tests dans un environnement de développement sécurisé pour valider les modifications apportées.
* **Consultation de la base de connaissances** : Le développeur peut également consulter directement la base de connaissances pour s'assurer que les informations qu'elle contient sont correctes et à jour.

#### Diagramme de Cas d'Utilisation pour l’Utilisateur Final



Figure 15 : Diagramme de Cas d'Utilisation pour l'Utilisateur Final

1. **Utilisateur Final** : LUtilisateur Final représente un utilisateur qui interagit directement avec PyBot.
2. **Interaction avec PyBot** : L'Utilisateur Final pose des questions à PyBot en langage naturel et reçoit des réponses basées sur les informations disponibles dans le système. Ce sont des réponses tirées des documents sources et générées par des algorithmes de traitement du langage naturel.

#### Conclusion

L'ensemble des cas d'utilisation souligne la distinction entre les interactions utilisateur final et développeur dans l'architecture RAG de PyBot. D'une part, l'utilisateur final bénéficie d'une interface simple et directe, tandis que, d'autre part, le développeur gère la complexité technique nécessaire au bon fonctionnement de PyBot. Cette architecture modulaire et bien définie permet à PyBot de fournir des réponses contextualisées et précises, tout en permettant une évolutivité et une personnalisation constante par l'équipe de développement.

## Modélisation Dynamique et Statique

### 3.1. Introduction aux Diagrammes UML dans la Conception du Système

La modélisation d'un système ne se limite pas à la simple identification des besoins et des interactions entre les acteurs. Il est essentiel de comprendre les aspects dynamiques et statiques du système pour s'assurer que sa conception est robuste, modulaire et extensible. Les diagrammes UML jouent un rôle crucial en offrant une vision détaillée du fonctionnement interne du système ainsi que de sa structure. Ce chapitre se concentre sur l'utilisation de diagrammes de séquence, de classes, de packages, ainsi que sur le prototypage d'interface utilisateur pour mieux appréhender le système dans son ensemble

### 3.2. Modélisation Statique

#### Diagramme de Classes

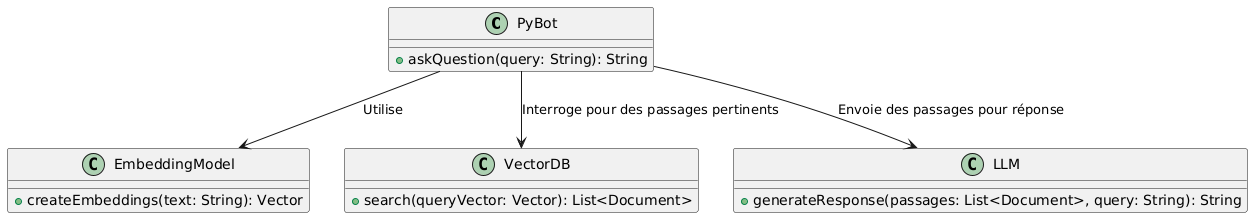


Figure 16 : Diagramme de Classes

Le diagramme de classes comme représenté dans la *figure 16* nous présente la structure du système PyBot, notamment les principales classes utilisées et leurs relations. Il s'agit d'un diagramme statique qui montre comment les objets du système sont structurés, leurs relations et responsabilités.

1. **PyBot** :
   * La classe principale qui orchestre le système.
   * Elle gère la question posée par l'utilisateur avec la méthode askQuestion(query: String).
   * Elle interagit avec les autres composants (modèle d'embeddings, base de données vectorielle, LLM) pour traiter la requête et générer une réponse.
2. **EmbeddingModel** :
   * Cette classe prend du texte (la requête utilisateur) et le transforme en **embedding** ou vecteur numérique.
   * Méthode : createEmbeddings(text: String): Vector.
   * Les embeddings sont ensuite utilisés pour effectuer une recherche dans la base de données vectorielle.
3. **VectorDB** (Base de données vectorielle) :
   * Elle stocke des embeddings de documents et est capable de rechercher les documents les plus pertinents en fonction d'un vecteur d'entrée (embedding de la requête utilisateur).
   * Méthode : search(queryVector: Vector): List<Document.
   * Les documents retournés sont des passages pertinents qui seront utilisés pour générer la réponse.
4. **LLM** (Large Language Model) :
   * Il s'agit du modèle génératif qui prend les documents trouvés par la base de données vectorielle et la requête de l'utilisateur pour générer une réponse en langage naturel.
   * Méthode : generateResponse(passages: List<Document>, query: String): String.
   * Il est responsable de la réponse finale envoyée à l'utilisateur.

*Relations entre les classes :*

* PyBot utilise le EmbeddingModel pour générer des embeddings à partir de la question de l'utilisateur.
* PyBot interroge la VectorDB avec l'embedding pour trouver les passages pertinents.
* PyBot envoie les documents trouvés à LLM pour générer une réponse complète.

#### Diagramme de Composants

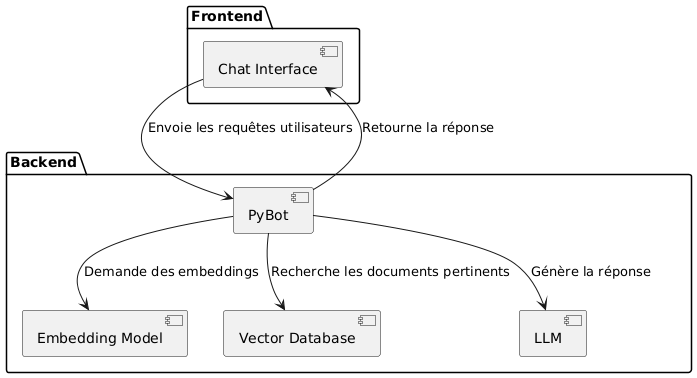


Figure 17 : Diagramme de Composants

Le diagramme de composants illustré dans la *figure 17* est une vue logique qui montre comment les différents composants du système PyBot sont organisés et interagissent entre eux pour réaliser les différentes tâches.

1. **Chat Interface** (Interface utilisateur) :
   * Ce composant représente l'interface que l'utilisateur final utilise pour poser des questions à PyBot.
   * Il envoie les questions posées au composant PyBot et reçoit les réponses.
2. **PyBot** (Backend principal) :
   * C'est le composant central du système qui orchestre tout le flux d'information.
   * Il reçoit les requêtes des utilisateurs, fait appel aux autres composants pour traiter la requête, et retourne une réponse à l'utilisateur.
3. **Embedding Model** :
   * Ce composant est chargé de transformer la question de l'utilisateur en embedding.
   * PyBot lui envoie la requête et reçoit l'embedding qui représente la question sous forme de vecteur.
4. **Vector Database** :
   * Elle stocke des embeddings de documents et est utilisée pour retrouver des passages pertinents en fonction de l'embedding de la requête.
   * PyBot interroge la base de données vectorielle pour obtenir les documents les plus pertinents.
5. **LLM** (Large Language Model) :
   * Ce composant reçoit les passages pertinents trouvés dans la base de données vectorielle et la requête utilisateur pour générer une réponse en langage naturel.
   * PyBot lui envoie les documents trouvés et la requête, et reçoit en retour une réponse à envoyer à l'utilisateur.

*Explication des relations :*

* Chat Interface interagit avec PyBot pour envoyer les questions de l'utilisateur et recevoir les réponses.
* PyBot fait appel au Embedding Model pour générer les embeddings de la requête.
* PyBot interroge la Vector Database pour retrouver des documents pertinents.
* LLM génère la réponse finale basée sur les documents trouvés.

### Modélisation Dynamique

#### Diagramme de Séquence

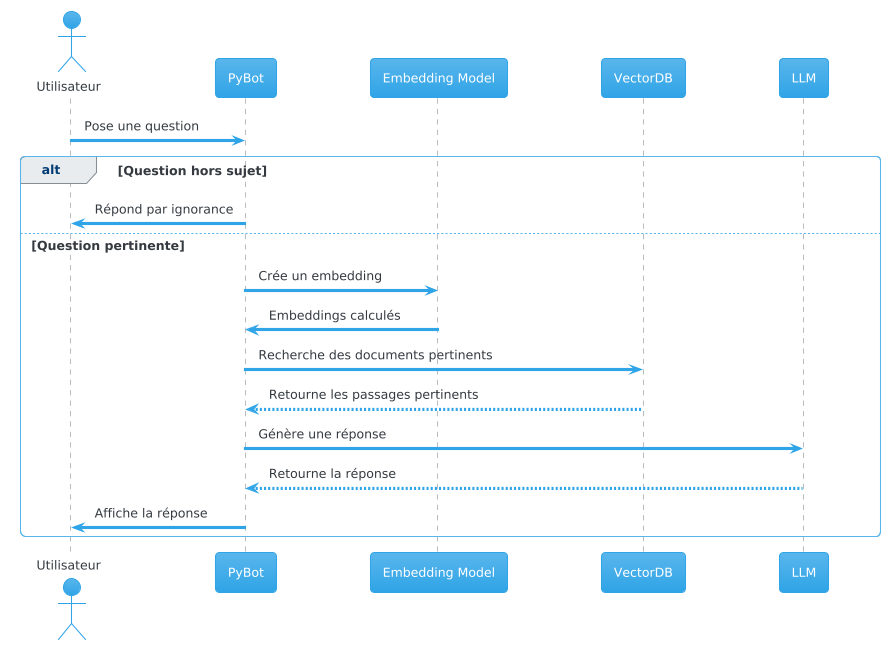


Figure 18 : Diagramme de Séquence

La figure 18 illustre notre diagramme de séquence et qui décrit le déroulement d'une requête utilisateur dans PyBot lorsqu'une question est posée. Il gère également le cas où la question est hors sujet et où PyBot renvoie une réponse générique.

1. **Utilisateur - PyBot** :
   * L'utilisateur soumet une question via l'interface utilisateur. Cette action déclenche l'envoi de la requête au composant PyBot pour traitement.
2. **Bloc alt** (Alternative) :
   * Ce bloc conditionnel gère deux scénarios :
     + Si la question est hors sujet : PyBot détecte que la question ne concerne pas son domaine de compétence (le développement Python). Il renvoie immédiatement une réponse standard du type : **"Je ne suis pas capable de répondre à cette question"**, sans engager de traitement supplémentaire.
     + Si la question est pertinente : Si la question concerne un sujet couvert par PyBot (des questions techniques sur le développement Python), le système continue le traitement normal de la requête.
3. **PyBot - Embedding Model** :
   * Lorsque la question est jugée pertinente, PyBot transmet la requête au composant Embedding Model. Ce modèle convertit la question de l'utilisateur en un embedding (vecteur numérique) qui représente sémantiquement la requête sous une forme exploitable par la base de données vectorielle.
4. **Embedding Model -> PyBot** :
   * Embedding Model renvoie l'embedding généré à PyBot pour les étapes suivantes de traitement.
5. **PyBot - VectorDB** :
   * PyBot interroge ensuite la VectorDB (base de données vectorielle) avec l'embedding généré pour rechercher les **documents pertinents** qui correspondent à la question. Cette étape est cruciale pour récupérer des informations pertinentes à partir de la base de connaissances.
6. **VectorDB - PyBot** :
   * VectorDB renvoie à PyBot une liste de passages pertinents (extraits de documents) basés sur la similarité entre l'embedding de la requête et les documents dans la base de données.
7. **PyBot - LLM** :
   * Une fois les passages pertinents obtenus, PyBot les transmet au LLM (Large Language Model), accompagné de la question initiale, pour générer une réponse complète et contextualisée en langage naturel.
8. **LLM - PyBot** :
   * Le LLM renvoie la réponse générée à PyBot. Cette réponse est basée sur les passages trouvés et la requête initiale de l'utilisateur.
9. **PyBot - Utilisateur** :
   * Enfin, PyBot retourne la réponse générée à l'utilisateur via l'interface utilisateur, concluant ainsi le cycle de traitement de la requête.

#### Diagramme d'Activité

Figure 19 : Diagramme d'Activité

Le diagramme d'activité représenté sur la *figure 19* montre le flux de travail pour le traitement des requêtes dans PyBot. Ce flux comprend le traitement normal d'une question pertinente et le traitement des questions hors sujet.

1. **Début - Utilisateur pose une question** :
   * Le flux commence avec l'utilisateur qui soumet une question via l'interface de PyBot.
2. **PyBot reçoit la question** :
   * PyBot reçoit la requête et la passe à une étape de vérification pour analyser la pertinence de la question.
3. **Condition - Question pertinente ?** :
   * À ce stade, PyBot effectue une vérification pour déterminer si la question relève de son domaine d'expertise (le développement Python).
   * **Cas 1 - Question pertinente** :
     + Si la question est pertinente, PyBot passe à l'étape suivante du traitement. La requête est convertie en un embedding à l'aide du composant EmbeddingModel.
   * **Cas 2 - Question hors sujet** :
     + Si la question ne correspond pas au domaine Python, PyBot renvoie directement une réponse d'ignorance, du type **"Je ne sais pas répondre à cette question"**. Le processus s'arrête ici pour les questions hors sujet.
4. **Création d'un embedding de la requête** :
   * Si la question est pertinente, PyBot utilise l'Embedding Model pour générer un embedding (représentation vectorielle) de la question. Cet embedding capture la sémantique de la requête pour permettre une recherche efficace dans la base de données.
5. **Recherche des documents pertinents** :
   * L'embedding généré est envoyé à la VectorDB pour rechercher des documentspertinents qui correspondent à la requête utilisateur.
6. **Condition - Documents trouvés ?** :
   * PyBot vérifie si des documents pertinents ont été trouvés dans la VectorDB.
   * **Cas 1 - Documents trouvés** :
     + Si des documents pertinents sont trouvés, PyBot envoie ces passages au LLM pour générer une réponse complète.
   * **Cas 2 - Aucun document trouvé** :
     + Si aucun document n'est trouvé, PyBot renvoie une réponse d'erreur (par exemple : **"Je n'ai pas trouvé de réponse à votre question"**).
7. **LLM génère la réponse** :
   * Si des documents pertinents ont été trouvés, le **LLM** génère une réponse en s'appuyant sur les passages extraits de la base de données et la question initiale.
8. **Retour de la réponse à l'utilisateur** :
   * La réponse générée par le **LLM** est envoyée à PyBot, qui la transmet ensuite à l'utilisateur via l'interface utilisateur.
9. **Fin du processus** :
   * Une fois la réponse envoyée, le processus est terminé et PyBot est prêt à traiter de nouvelles requêtes.

## Conclusion

La phase de conception et de modélisation abordée dans ce chapitre a été essentielle pour poser les fondations solides du développement du chatbot PyBot. En utilisant une combinaison d'UML et de diagrammes techniques adaptés à l'architecture RAG (Retrieval-Augmented Generation), nous avons traduit les spécifications fonctionnelles et non-fonctionnelles en une architecture technique complète et détaillée.

Les diagrammes de cas d'utilisation ont permis de clarifier les rôles des différents acteurs, notamment le développeur et l'utilisateur final, ainsi que leurs interactions respectives avec le système. Cela a permis de définir précisément les fonctionnalités principales à développer, comme l’interaction avec le chatbot et la gestion des modules techniques.

La modélisation dynamique à travers les diagrammes de séquence a offert une vision claire de l'ordre des interactions entre les différents composants du système, y compris la gestion des requêtes pertinentes et hors sujet. En parallèle, la modélisation statique avec les diagrammes de classes et de composants a décrit en détail la structure interne du système PyBot, en illustrant les différentes dépendances et relations entre les composants clés comme le Modèle d'Embeddings, la Base de Données Vectorielle, et le LLM.

Ces représentations graphiques ne sont pas seulement des outils de visualisation et de communication, mais elles constituent également une base solide pour le développement. Elles assurent que chaque composant est bien conçu et que les interactions au sein du système se dérouleront de manière cohérente et fluide. La prise en compte de scénarios spécifiques, comme la gestion des questions hors sujet, a également permis de garantir la robustesse et l’efficacité de PyBot dans divers contextes d'utilisation.

En conclusion, cette phase de conception rigoureuse et structurée pour PyBot fournit des bases solides pour le développement à venir, en minimisant les risques d'erreurs et en maximisant les chances de succès. Grâce à cette préparation minutieuse, nous avons une assurance que l'architecture du système pourra évoluer de manière fluide, en s’adaptant aux besoins des utilisateurs tout en offrant une expérience utilisateur optimisée.

Chapitre III : Etude Technologique

## Introduction

L'étude technologique est une étape clé dans la réussite de tout projet complexe, en particulier pour un système intelligent comme PyBot basé sur l'architecture RAG (Retrieval-Augmented Generation). Ce chapitre se concentre sur une analyse approfondie des technologies, outils et frameworks qui seront utilisés pour garantir la performance, la sécurité et la scalabilité du projet.

L'objectif est de choisir les solutions les plus adaptées aux besoins spécifiques, tout en examinant les alternatives possibles et en anticipant les défis techniques. Cette démarche permet non seulement de minimiserles risques associés au développement, mais aussi de s'assurer que l'intégration des technologies dans l'infrastructure existante sera fluide et efficace.

En réalisant cette étude, nous posons les fondations solides pour un développement agile et durable, en sélectionnant les meilleures pratiques pour maximiser les chances de succès du projet.

# Contexte Technologique et Choix des Outils

## Introduction

Le développement de systèmes intelligents, tels que les chatbots, est devenu une composante majeure des technologies modernes. Ces systèmes s'appuient sur des modèles complexes d'intelligence artificielle pour comprendre et répondre aux besoins des utilisateurs, notamment en matière de traitement du langage naturel (NLP). Dans le cadre de ce projet, l'objectif est de concevoir et développer un chatbot intégré capable de fournir des réponses contextuelles optimisées.

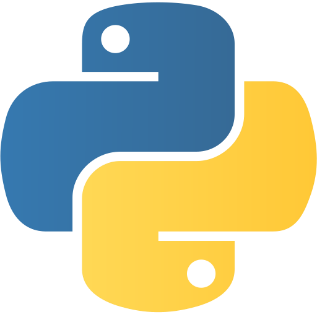
Le succès de ce projet repose non seulement sur une architecture performante, comme l'architecture RAG (Retrieval-Augmented Generation), mais aussi sur le choix judicieux des technologies et outils utilisés pour le développement, la gestion et l'optimisation du chatbot. Une intégration harmonieuse des technologies de traitement des données textuelles, d'apprentissage automatique et de génération de texte est nécessaire pour atteindre des performances élevées, tout en garantissant une expérience utilisateur fluide.

Le contexte technologique dans lequel ce projet s'inscrit nécessite une gestion efficace de documents volumineux, une recherche sémantique rapide, et une génération de texte basée sur des modèles de langage de pointe. À cette fin, des outils spécifiques ont été sélectionnés pour répondre aux différents besoins du système, notamment pour l'extraction de contenu, la gestion des embeddings, l'optimisation des ressources et l'interface utilisateur.

Dans cette section, nous allons détailler les technologies et outils utilisés tout au long du projet, en expliquant les raisons de ces choix, leur utilité dans le cadre du développement et leur rôle dans l'optimisation globale du chatbot.

## Technologies et Outils Utilisés

### 2.1. Langage de Programmation : Python



Python est un langage de programmation polyvalent et interprété, créé à la fin des années 1980 par Guido van Rossum. Connu pour sa syntaxe simple et lisible, Python favorise une écriture de code claire et concise, ce qui en fait un choix privilégié pour les développeurs, qu'ils soient débutants ou experts. La popularité de Python a grandi rapidement, et il est devenu un des langages les plus utilisés dans divers domaines, y compris le développement web, l'analyse de données, l'intelligence artificielle (IA), et la science des données.

Figure 20 : Logo Python

#### Présentation de Python pour le Traitement du Langage Naturel (NLP)

Python est devenu l'un des langages de programmation les plus utilisés pour le développement de solutions d'intelligence artificielle, en particulier dans le domaine du traitement du langage naturel (NLP). Sa popularité dans ce domaine repose sur plusieurs facteurs, notamment sa simplicité syntaxique, son large écosystème de bibliothèques spécialisées, et sa communauté dynamique qui contribue continuellement à l'amélioration et au développement de nouveaux outils.

Le traitement du langage naturel implique plusieurs tâches complexes telles que la tokenisation, le stemming, la lemmatisation, la reconnaissance des entités nommées (NER), la classification de texte, et la génération de texte. Python offre une vaste gamme de bibliothèques et de frameworks conçus pour répondre aux exigences de chaque étape du pipeline NLP.

Voici quelques-unes des raisons pour lesquelles Python est le langage de choix pour le NLP :

1. **Bibliothèques dédiées au NLP** : Python dispose de bibliothèques puissantes et bien maintenues, telles que :
   * **NLTK (Natural Language Toolkit)** : Une bibliothèque complète pour le traitement du langage naturel, couvrant des aspects tels que la tokenisation, le traitement syntaxique, la classification et l’analyse de sentiments.
   * **spaCy** : Un framework NLP moderne, efficace et rapide qui facilite la gestion de grands volumes de données et est adapté pour l'intégration avec des projets industriels.
   * **Transformers (Hugging Face)** : Une bibliothèque axée sur les modèles de langage de grande envergure (LLM) tels que GPT, BERT, et T5, permettant de fine-tuner des modèles pré-entraînés pour des tâches spécifiques de NLP.
   * **TextBlob** : Un outil simple pour le traitement de texte, utile pour des tâches rapides comme l'analyse de sentiments et la correction grammaticale.
2. **Intégration avec des frameworks de Machine Learning** : Python est également largement utilisé pour les modèles d'apprentissage automatique grâce à ses bibliothèques spécialisées comme TensorFlow, PyTorch, et Keras, qui permettent de construire et de former des modèles de deep learning adaptés au traitement des données textuelles.
3. **Simplicité et Rapidité de Développement** : L’approche syntaxiquement simple de Python permet aux développeurs de créer rapidement des prototypes de modèles NLP, d'essayer différentes architectures, et d'intégrer facilement les résultats dans des applications réelles.
4. **Interopérabilité avec des Outils de Gestion de Données** : Python s'intègre parfaitement avec des outils de manipulation de données comme Pandas et NumPy, facilitant ainsi le traitement et la préparation des jeux de données textuels avant leur passage dans des pipelines NLP.
5. **Communauté et Support** : Python bénéficie d'une large communauté de chercheurs et d'ingénieurs en IA, ce qui rend accessible de nombreuses ressources pédagogiques, des forums d'entraide, ainsi que des mises à jour fréquentes et des optimisations constantes des outils dédiés au NLP.

#### Modules Python utilisés

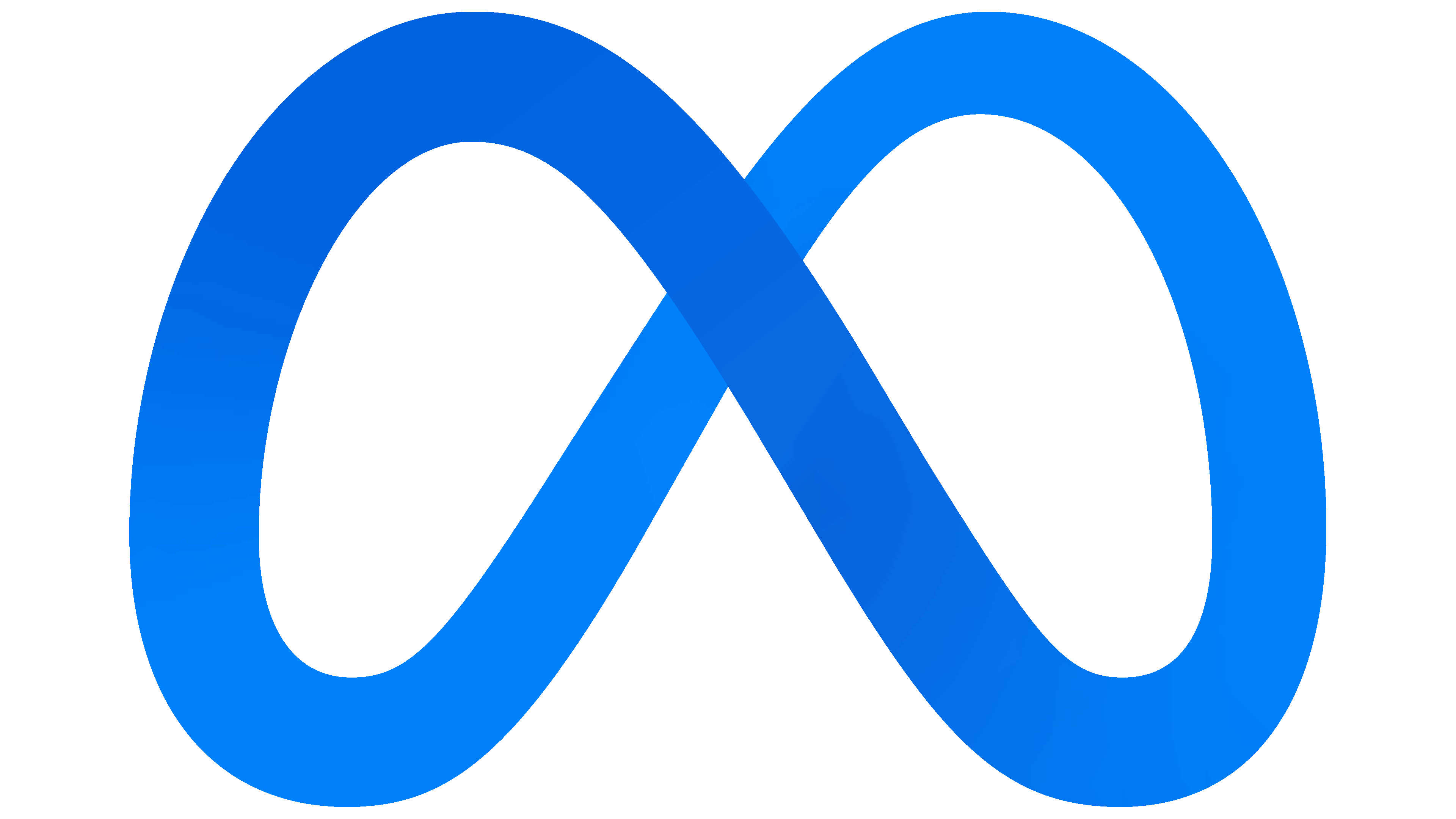
La conception et le développement d’un chatbot performant basé sur l'architecture RAG (Retrieval-Augmented Generation) nécessitent une intégration fine de plusieurs modules Python spécialisés. Ces outils couvrent diverses étapes du pipeline, incluant la gestion des documents, l’encodage des données, la recherche sémantique et la génération de texte. Voici une description technique des principaux modules utilisés dans ce projet :

* **LangChain :**



LangChain est un framework Python moderne conçu pour gérer les interactions entre plusieurs modèles de langage à grande échelle (LLM) et d'autres outils dans des pipelines complexes. Il est au cœur de la gestion de l'architecture RAG, facilitant l’intégration des composants de recherche d’informations et de génération de texte.

Figure 21 : Logo de LangChain

* **Fonctionnalité dans le projet** : LangChain permet d’orchestrer la récupération d’informations (retrieval) via FAISS et l’encodage des requêtes en embeddings à l’aide de modèles comme celles de Hugging Face. Il gère également le passage fluide des résultats entre ces modules et l’API OpenAI pour la génération des réponses.
* **Avantages techniques** : LangChain simplifie l’intégration des modèles de récupération et de génération, tout en automatisant le flux de traitement des données, ce qui permet une optimisation des performances du système tout en réduisant la complexité de l'implémentation.
* **Optimisation du flux RAG** : Grâce à LangChain, il est facile de chaîner les processus d'encodage et de génération en minimisant la latence et en améliorant l’efficacité globale.
* **FAISS (Facebook AI Similarity Search):**

**FAISS** est une bibliothèque spécialisée développée par **Facebook AI** pour gérer des recherches sémantiques rapides dans de larges ensembles de données vectorielles. FAISS est indispensable dans l'architecture RAG pour la récupération d'informations pertinentes en se basant sur des similarités sémantiques entre les embeddings des documents et les requêtes utilisateur.

Figure 22 : Logo FAISS

**FAISS**

* **Rôle dans l’architecture RAG** : FAISS stocke les embeddings générés à partir des documents textuels. Lorsqu’un utilisateur pose une question, FAISS retrouve les passages les plus pertinents en comparant l'embedding de la question aux embeddings stockés, facilitant ainsi la recherche sémantique.
* **Détails techniques** : FAISS utilise des structures d'index optimisées telles que **IVF (**Inverted File System**)** et **HNSW (**Hierarchical Navigable Small World**)** pour effectuer des recherches k-nearest neighbors **(KNN)** très rapidement, même sur des volumes de données importants. L’utilisation de CUDA permet d'accélérer les recherches via des calculs sur GPU.
* **Le choix de FAISS** : Il est particulièrement efficace pour traiter de grands volumes d'embeddings, garantissant une recherche quasi-instantanée dans une base de données contenant des millions de documents, ce qui est essentiel pour l’optimisation de l'architecture RAG.
* **OpenAI API :**

**OpenAI API** permet d’accéder aux modèles de génération de texte **GPT-3** et **GPT-4**, qui sont parmi les plus puissants pour les tâches de génération contextuelle. Dans le cadre de l’architecture RAG, l’API OpenAI est utilisée pour transformer les informations récupérées par FAISS en réponses textuelles cohérentes et précises.

Figure 23 : Logo OpenAI

* **Fonctionnement dans le projet** : Une fois que FAISS a identifié les passages pertinents, ces derniers sont envoyés au modèle GPT-3 ou GPT-4 via l’API OpenAI pour générer une réponse. Le modèle se base à la fois sur le contenu récupéré et sur la requête initiale de l’utilisateur pour produire une réponse contextuelle.
* **Intégration** : **UNIFY.AI** a été utilisé pour sélectionner et benchmarker le modèle LLM optimal (via des tests sur la qualité des réponses et le temps de traitement), garantissant que les modèles utilisés dans ce projet sont les mieux adaptés à la tâche de génération de texte.
* **Avantages techniques** : OpenAI fournit un accès à des modèles LLM de pointe, réduisant ainsi le besoin de ressources internes pour entraîner ou fine-tuner des modèles complexes. De plus, il permet une génération rapide et une scalabilité sans nécessiter d'infrastructure lourde.
* **Hugging Face (Transformers) :**



Hugging Face est une plateforme leader dans le domaine du traitement du langage naturel, offrant un accès à une large collection de modèles pré-entraînés via sa bibliothèque Transformers.

Figure 24 : Logo Hugging Face

Dans ce projet, Hugging Face joue un rôle clé dans la génération des embeddings et la préparation des modèles pour la phase de récupération dans l'architecture RAG.

* **Encodage des documents en embeddings** : Des modèles comme Sentence-BERT (disponibles via Hugging Face) sont utilisés pour transformer les documents et les requêtes utilisateur en embeddings vectoriels. Ces embeddings sont ensuite utilisés par FAISS pour évaluer la similarité sémantique.
* **Flexibilité et optimisation** : Hugging Face permet d'utiliser des modèles comme BERT, RoBERTa, ou DistilBERT pour l’encodage, garantissant une personnalisation en fonction des besoins du projet. Si nécessaire, ces modèles peuvent également être fine-tunés pour améliorer la précision des embeddings.
* **Pourquoi Hugging Face ?** : La bibliothèque « Transformers » offre des modèles puissants pour l'encodage et la génération, avec une facilité d'intégration et des performances élevées. L’utilisation de Sentence-BERT améliore considérablement la précision des embeddings pour la recherche sémantique.

****

* **PyMuPDF (fitz) :**

PyMuPDF est un module Python efficace pour l'extraction et le traitement des documents PDF. Dans ce projet, PyMuPDF est utilisé pour extraire le contenu textuel des documents PDF et les transformer en chunks (segments de phrases), facilitant ainsi leur encodage en embeddings.

Figure 25 : Logo PYMuPDF

* **Rôle dans le pipeline RAG** : PyMuPDF est responsable de l'extraction de texte à partir des documents PDF, qui sont ensuite segmentés en parties logiques. Ces segments de texte sont ensuite encodés en embeddings via des modèles Hugging Face et stockés dans FAISS pour la recherche sémantique.
* **Avantages techniques** : PyMuPDF permet une extraction précise et rapide des documents volumineux tout en garantissant la qualité du texte extrait, ce qui est crucial pour assurer la pertinence des données utilisées lors des phases ultérieures du pipeline.
* **Pourquoi PyMuPDF ?** : Il offre une fiabilité élevée dans l'extraction de données textuelles complexes, tout en permettant une gestion facile des fichiers PDF de grande taille, souvent utilisés dans les projets de recherche documentaire.



* **Streamlit :**

Streamlit est un framework open-source conçu pour créer rapidement des interfaces web interactives, en particulier pour les applications de machine learning. Dans ce projet, Streamlit est utilisé pour fournir une interface simple et conviviale où les utilisateurs peuvent poser des questions et recevoir des réponses en temps réel.

Figure 26 : Logo Streamlit

* **Rôle dans le projet** : L’utilisateur interagit via une interface web développée avec Streamlit, posant des questions qui sont ensuite envoyées au pipeline backend (LangChain, FAISS, OpenAI). Les réponses générées par le modèle LLM sont ensuite affichées à l’utilisateur via l’interface.
* **Avantages techniques** : Streamlit permet de développer une interface utilisateur rapidement, sans avoir besoin de compétences avancées en développement web. Il s'intègre facilement aux solutions Python existantes, facilitant la communication en temps réel entre l'interface et le pipeline de génération de réponses.
* **Pourquoi Streamlit ?** : Streamlit simplifie l’implémentation d'une interface utilisateur réactive et facilement déployable. Son intégration transparente avec Python en fait un choix idéal pour des projets de cette envergure.

#### Avantages de Python pour la gestion des modèles d'IA et des API

Python est devenu le langage de prédilection pour le développement de modèles d'intelligence artificielle (IA) et l'intégration avec des API. Voici les principaux avantages de Python dans ce contexte, en particulier pour la gestion des modèles de traitement du langage naturel (NLP) et l'intégration avec des services API comme OpenAI, Hugging Face, ou d'autres solutions AI-as-a-Service :

1. **Écosystème Riche en Bibliothèques d'IA**

L'un des principaux avantages de Python est son écosystème riche en bibliothèques spécialisées pour le développement et l'entraînement de modèles d'IA. Ces bibliothèques simplifient la mise en place de pipelines de machine learning et de deep learning, et permettent de manipuler facilement des données textuelles ou structurées. Voici quelques bibliothèques phares :

* **TensorFlow** et **Keras** : Ces frameworks permettent de construire et d'entraîner des modèles de deep learning, y compris des réseaux de neurones profonds pour la vision par ordinateur, le NLP et d'autres applications IA.
* **PyTorch** : Utilisé notamment dans la recherche et l'industrie, PyTorch offre une approche flexible et dynamique pour la construction de modèles d'IA, facilitant les opérations sur les tensors et l'entraînement de modèles complexes. Il est largement adopté pour des tâches NLP, comme l’entraînement de modèles transformer.
* **Hugging Face Transformers** : Une bibliothèque dédiée aux modèles NLP pré-entraînés, elle offre des modèles comme **BERT**, **GPT**, **T5**, et **DistilBERT** qui peuvent être facilement fine-tunés et déployés pour des tâches spécifiques de traitement du langage.
* **Scikit-learn** : Pour les modèles d'apprentissage automatique plus classiques, Scikit-learn offre des outils robustes pour la classification, la régression, le clustering et d'autres tâches ML.

**2. Simplicité d'Intégration avec les API d'IA**

Python offre une interopérabilité exceptionnelle avec les API, facilitant l'intégration avec des services comme OpenAI et Hugging Face. Cette simplicité d'intégration permet d'exploiter des modèles d'IA avancés via des appels API sans avoir besoin de les entraîner localement.

* **OpenAI API** : Python peut interagir avec des modèles comme GPT-3/4 via l'API OpenAI, permettant ainsi de générer du texte, de traduire des langues, de répondre à des questions et bien plus encore avec quelques lignes de code seulement. Grâce à des bibliothèques comme Requests ou HTTPx, l'intégration des API est simple et rapide.
* **Hugging Face API** : Grâce à l'API Hugging Face, les utilisateurs peuvent accéder à des modèles NLP pré-entraînés pour des tâches variées, comme la classification de texte, la génération de texte, et la traduction. L'intégration de Hugging Face via Python permet de déployer des solutions IA prêtes à l'emploi, augmentant ainsi la productivité.

**3. Simplicité Syntaxique et Rapidité de Développement**

Python est un langage reconnu pour sa syntaxe simple et lisible, facilitant ainsi le développement rapide de solutions IA. Cette simplicité est particulièrement utile lors de la gestion des modèles et de leur déploiement via des API. Les développeurs peuvent se concentrer sur les concepts d’IA sans être submergés par des aspects syntaxiques complexes.

* **Code concis et clair** : Python permet de prototyper rapidement des modèles d'IA, d’expérimenter différentes architectures, et d'intégrer des modèles avec des API externes en quelques lignes de code.
* **Facilité d'apprentissage** : Cette simplicité encourage l’adoption rapide par les nouveaux développeurs IA et facilite la collaboration sur des projets complexes, en particulier dans des environnements de recherche où le partage de code est important.

**4. Portabilité et Flexibilité**

Python est extrêmement portable et flexible, ce qui signifie que les modèles d'IA et les API développés dans ce langage peuvent être facilement transférés entre des environnements de développement, des systèmes locaux, ou des infrastructures cloud. Cela permet d'assurer une compatibilité multi-plateforme et un déploiement facile des modèles sur divers systèmes.

* **Support multi-plateforme** : Python fonctionne sur la majorité des systèmes d'exploitation (Linux, Windows, MacOS), facilitant ainsi le transfert des modèles IA entre différents environnements de développement.

**5. Optimisation des Performances avec GPU et TPU**

Python, en particulier à travers des frameworks comme TensorFlow et PyTorch, permet d'exploiter la puissance des GPU et des TPU pour accélérer les calculs d'apprentissage profond. Cette capacité est cruciale pour l’entraînement et l’inférence sur de grands modèles IA.

* **CUDA et PyTorch** : Python s'intègre avec CUDA pour permettre l'exécution des calculs sur des GPU NVIDIA, ce qui réduit considérablement le temps d’entraînement des modèles de deep learning.
* **TPU et TensorFlow** : Pour les environnements de calcul à grande échelle, Python offre la possibilité d'utiliser des TPU (Tensor Processing Units) via TensorFlow, permettant une accélération supplémentaire dans des environnements de cloud comme Google Cloud.

**6. Communauté Dynamique et Large Base de Documentation**

Python bénéficie d'une communauté mondiale très active, ainsi que d'une abondante documentation et de tutoriels. Cette communauté contribue à l’amélioration continue des outils et des bibliothèques IA, en facilitant l’accès à des ressources pédagogiques, des forums, et des mises à jour fréquentes.

* **Documentation exhaustive** : Les bibliothèques comme TensorFlow, PyTorch, Hugging Face, et Scikit-learn possèdent une documentation exhaustive, facilitant leur utilisation par les développeurs et chercheurs.
* **Forums d'entraide** : Des plateformes comme Stack Overflow, ou Hugging Face Forum offrent des solutions rapides et des échanges d'expertise, contribuant à résoudre les problèmes liés au développement IA.

### Benchmarking des LLM

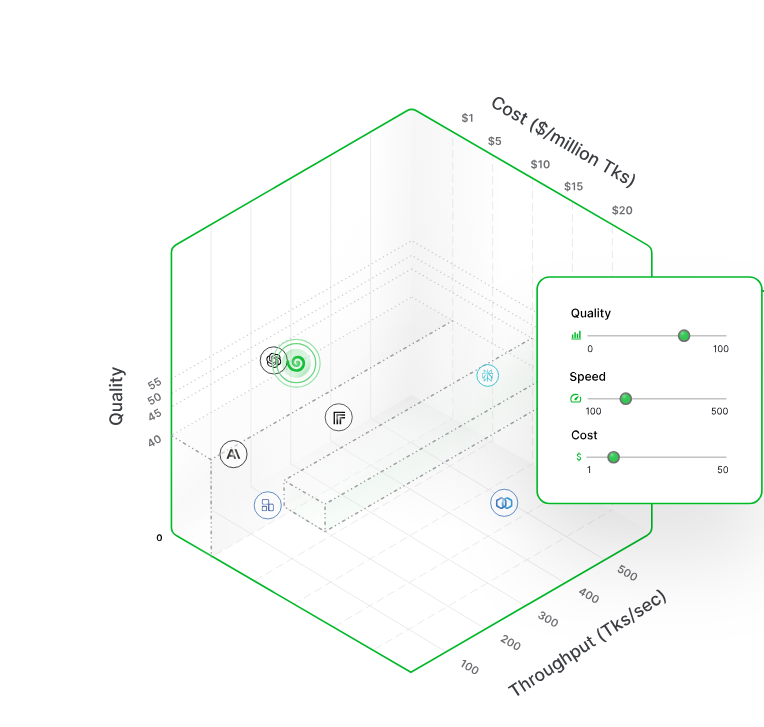
Dans le cadre de notre projet, il est essentiel de sélectionner le modèle de langage large (LLM) le plus performant pour la génération de réponses contextuelles. Le choix du bon modèle a un impact direct sur la qualité des réponses générées, la pertinence du contenu produit, ainsi que sur la rapidité d'exécution du système. Afin de garantir que le modèle LLM sélectionné soit optimisé pour ces critères, nous avons utilisé **UNIFY.AI**, une plateforme spécialisée dans le benchmarking des modèles LLM.

#### Présentation de UNIFY.AI

****

**UNIFY.AI** est une plateforme dédiée au benchmarking des modèles LLM, utilisée pour comparer les performances des modèles de génération de texte en fonction de plusieurs critères : qualité de la génération, latence (temps de réponse), et coût par requête. Elle permet d’évaluer les modèles LLM de différents fournisseurs, tels que OpenAI et Anthropic, et d'obtenir une analyse comparative basée sur des scénarios de tests réels.

Figure 27 : Logo Unify.ai



* **Utilité de UNIFY.AI :** UNIFY.AI a été utilisé pour évaluer et comparer plusieurs modèles LLM afin de sélectionner celui offrant le meilleur compromis entre qualité, performances et coût. Les modèles testés incluent ceux de OpenAI (GPT-3.5 Turbo, GPT-4, GPT-4 Turbo, GPT-4o) et Anthropic (Claude-3, dans plusieurs versions comme Haiku, Sonnet, et Opus).
* **Objectifs du benchmarking** : Le but était de trouver un modèle capable de générer des réponses contextuelles de haute qualité à partir des passages récupérés via FAISS, tout en maintenant une latence faible et un coût raisonnable pour une utilisation à grande échelle.

Figure 28 : Les principaux Axes de performances chez Unify.ai

#### Processus de Benchmarking des LLM

Le processus de benchmarking réalisé avec **UNIFY.AI** a suivi plusieurs étapes cruciales pour évaluer la pertinence des modèles et optimiser les performances du chatbot :

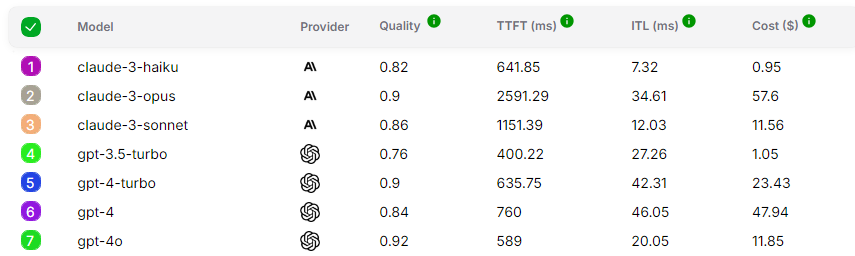
1. **Sélection des Modèles à Tester** :
   * Les modèles de génération les plus courants, tels que GPT-3.5-Turbo, GPT-4, GPT-4o, Claude-3 (versions Haiku, Sonnet, Opus), ont été sélectionnés pour être évalués. Chaque modèle présente des caractéristiques différentes en termes de qualité et de performance.
   * Nous avons utilisé des données textuelles récupérées par FAISS et soumis des requêtes utilisateur à chaque modèle pour mesurer la qualité et la cohérence des réponses générées.
2. **Métriques Utilisées pour le Benchmarking** :
   * **Qualité de la Réponse** : Mesurée sur une échelle de 0 à 1, cette métrique évalue la capacité du modèle à produire une réponse pertinente, cohérente et précise.
   * **TTFT (Time to First Token)** : Le temps nécessaire pour que le modèle génère le premier token de la réponse. Un temps plus court indique une meilleure réactivité du modèle.
   * **ITL (Inference Latency)** : Temps total pour générer la réponse complète, mesuré en millisecondes (ms). Une latence plus faible garantit une meilleure expérience utilisateur dans des scénarios en temps réel.
   * **Coût par Requête** : Coût en dollars (USD) /1M de tokens générée par le modèle. Ce paramètre est essentiel pour évaluer la viabilité du modèle pour une utilisation à grande échelle, surtout dans les cas où de nombreuses requêtes sont effectuées quotidiennement.
3. **Analyse des Résultats** :
   * Après avoir testé les modèles sur des scénarios variés, UNIFY.AI a fourni un tableau comparatif des performances des modèles. Cela inclut une vue d’ensemble de la qualité des réponses, la latence, et le coût par requête.

Table 3 : Tableau comparatif des performances des modèles

**GPT-4o** offre un excellent compromis entre coût, qualité et latence comme représenté sur la *table 3*. Un modèle très performant avec un coût modéré. Il est donc idéal pour des applications où la réduction des coûts est une priorité.

#### Intégration du modèle LLM via OpenAI API

Après avoir effectué une évaluation approfondie des modèles de langage large (LLM) via UNIFY.AI, le modèle GPT-4o a été retenu pour son excellent compromis entre qualité de génération, latence raisonnable et coût faible. Ce modèle, fourni par OpenAI, s'intègre parfaitement au pipeline d'architecture RAG (Retrieval-Augmented Generation) via l'API OpenAI, qui permet d'accéder aux puissants modèles GPT pour la génération de réponses contextuelles.

L'intégration de GPT-4o via l'API OpenAI dans le pipeline de notre chatbot nécessite une gestion efficace de la récupération d'informations, du traitement des embeddings, et de la génération de texte final. Voici une description détaillée des étapes d’intégration et des considérations techniques.

##### Fonctionnement de l'API OpenAI pour GPT-4o



Figure 29 : Fonctionnement de l'API OpenAI

La figure 29 nous démontre le fonctionnement de L’API OpenAI où cette dernière permet d’accéder au modèle GPT-4o via une interface RESTful, où les données (requêtes utilisateur et contexte récupéré) sont envoyées sous forme de requêtes HTTP, et les réponses générées sont renvoyées en format JSON. Cette architecture API permet une communication en temps réel entre le modèle et le pipeline d’architecture RAG.

* **Accès aux Modèles LLM** : L'API OpenAI propose plusieurs versions des modèles GPT (telles que GPT-3, GPT-3.5-Turbo, GPT-4, GPT-4 Turbo, GPT-4o), permettant de choisir la version la plus adaptée aux besoins en termes de performance et de coût.
* **Clé API Sécurisée** : Chaque interaction avec l'API OpenAI est sécurisée via une clé d'API personnelle, garantissant que seuls les utilisateurs autorisés peuvent accéder au modèle.

##### Étapes d’Intégration de GPT-4o dans le Pipeline RAG

L'intégration du modèle GPT-4o dans le pipeline RAG commence par la récupération d'informations jusqu'à la génération de réponses textuelles comme illustré dans la *figure 30*.

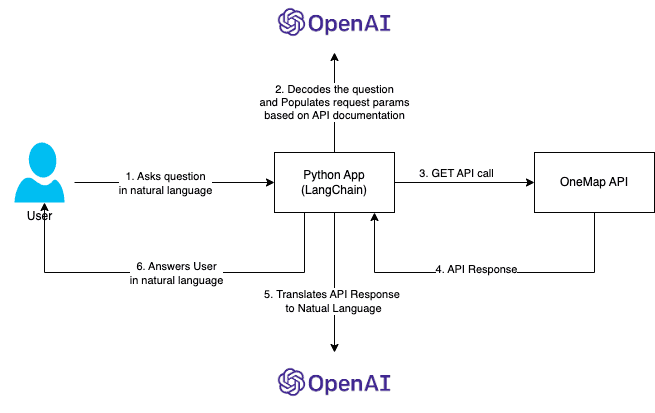


Figure 30 : RAG Pipeline

1. **Récupération des Passages Pertinents avec FAISS** :
   * Après avoir encodé les documents avec un modèle d’embeddings (via Hugging Face), les vecteurs résultants sont stockés dans FAISS pour permettre des recherches rapides.
   * Lorsqu'un utilisateur pose une question, elle est encodée et comparée aux documents stockés dans FAISS. Les passages les plus pertinents sont récupérés sous forme de texte pour être utilisés dans l'étape de génération.
2. **Préparation des Données pour GPT-4o** :
   * Les passages récupérés via FAISS sont combinés avec la requête utilisateur pour former une entrée contextuelle qui est transmise au modèle GPT-4o
   * Le contexte est structuré sous forme de prompt contenant la question de l'utilisateur et les passages sélectionnés. Cette méthode est essentielle pour maximiser la qualité de la réponse générée, car elle fournit à GPT-4o les informations nécessaires pour formuler une réponse basée sur des données externes.
3. **Appel de l'API OpenAI** :
   * Une requête POST est envoyée à l’API OpenAI, contenant le prompt sous forme d’un JSON structuré. Ce prompt inclut la question, les passages récupérés, et les paramètres de génération, tels que le nombre maximum de tokens (longueur de la réponse) et le niveau de température (contrôle de la créativité de la réponse).
4. **Réception et Gestion des Réponses** :

* L'API retourne une réponse sous forme de JSON, qui contient le texte généré par GPT-4o. Cette réponse est ensuite affichée à l'utilisateur via l'interface utilisateur (via Streamlit).

1. **Optimisation des Paramètres de Génération** :

* **Température** : Un paramètre qui contrôle le degré de "créativité" de la réponse générée. Une température basse (proche de 0) force le modèle à produire des réponses plus déterministes et précises, tandis qu'une température plus élevée (proche de 1) permet des réponses plus diversifiées.
* **Max Tokens** : Définit la longueur maximale de la réponse générée. En fonction des besoins, ce paramètre est ajusté pour garantir que les réponses soient suffisamment longues pour traiter correctement la question posée.
* **Frequency et Presence Penalties** : Ces paramètres permettent de contrôler la répétition des informations dans la réponse générée. Ils sont ajustés pour éviter que GPT-4o ne réitère les mêmes concepts ou phrases.

### Environnement de Développement avec Lightning AI Studio



Le développement d’un projet complexe comme le nôtre nécessite une infrastructure puissante pour gérer des modèles LLM, l’encodage d’embeddings, et la recherche sémantique. Pour répondre à ces exigences, Lightning AI Studio a été utilisé comme plateforme de développement, offrant des ressources GPU de haute performance ainsi qu'un environnement intégré pour faciliter la synchronisation des modules et optimiser l’ensemble du pipeline.

Figure 31 : Logo Lightning AI

#### Gestion Simplifiée des Ressources GPU et des Modules

Lightning AI Studio est une plateforme de développement avancée qui fournit un environnement complet et optimisé pour les projets d’intelligence artificielle. Grâce à cette plateforme, les développeurs bénéficient d’un accès facilité à des ressources GPU puissantes et d’une gestion automatisée des ressources nécessaires à l’exécution des modèles, tout en centralisant le travail dans un studio dédié comme représenté sur la *figure 32*.

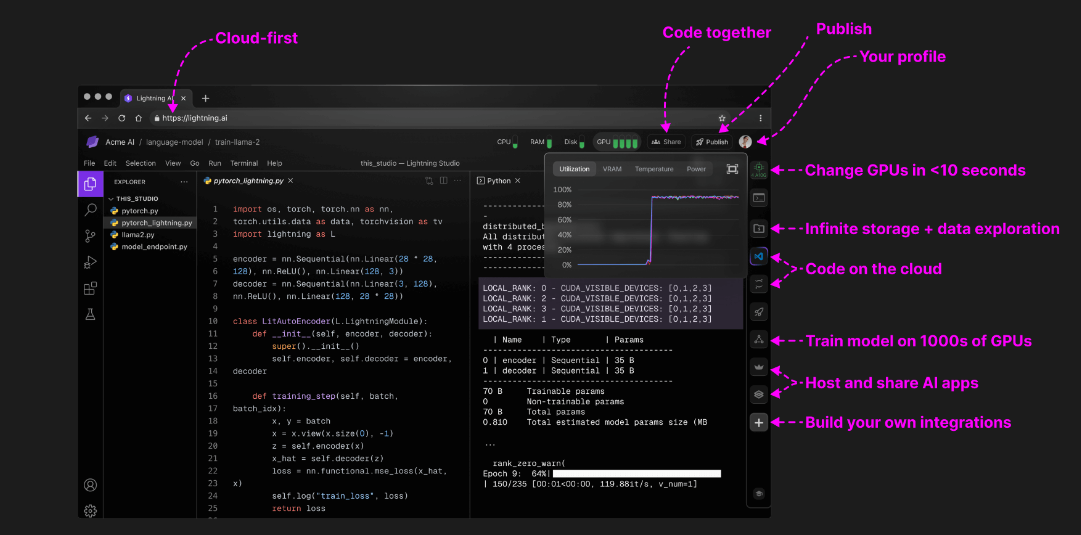


Figure 32 : Les différentes ressources de Lightning AI Studio

**Principales caractéristiques de Lightning AI Studio :**

1. **Environnement de Studio Collaboratif** :

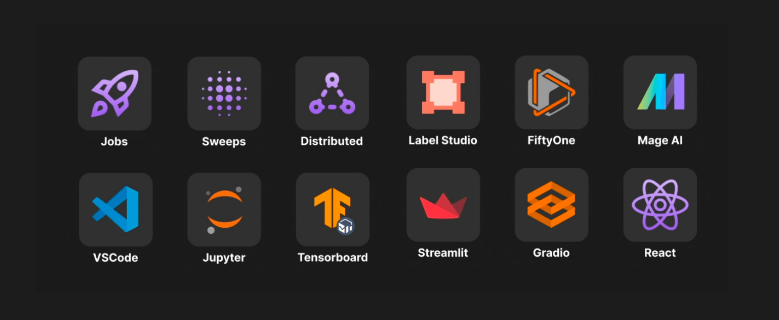


Figure 33 : Les différents Plugins de Lightning AI

* + Lightning AI Studio offre un espace de développement centralisé, où les différents modules de votre projet peuvent être synchronisés facilement, garantissant que toutes les parties prenantes disposent de la même version du projet. Le studio prend également en charge des outils de collaboration intégrés, permettant une meilleure coordination entre les développeurs et les chercheurs.
  + Cela permet d’accélérer les cycles de développement, de tester les modèles rapidement et d’effectuer des ajustements en temps réel, tout en bénéficiant d’un environnement stable et bien équipé.

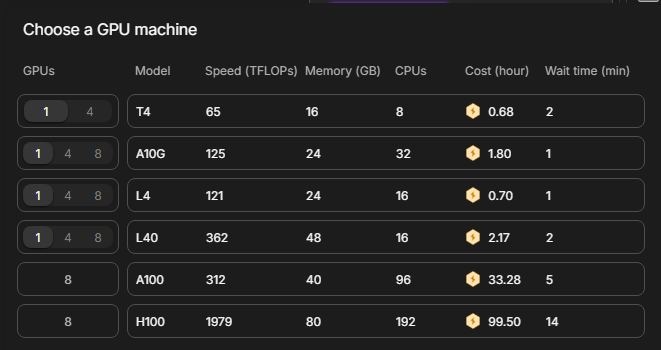
1. **Accès Automatisé à des GPUs Hautement Performants** :

Figure 34 : Les différentes Ressources GPU que propose Lightning AI

* + L'une des fonctionnalités clés de Lightning AI est son accès simplifié à des GPUspuissants. Pour les projets nécessitant une grande puissance de calcul, tels que l'entraînement de modèles de deep learning ou l’inférence rapide des modèles LLM via des API comme OpenAI, Lightning AI alloue automatiquement des ressources GPU adaptées à la charge de travail.
  + Cela permet de réduire la latence et d’optimiser les performances lors de l'entraînement et de l'inférence, tout en supprimant la nécessité de gérer manuellement l'infrastructure. Cette allocation automatique assure une utilisation efficace des ressources sans avoir à configurer ou surveiller les serveurs de manière continue.

1. **Déploiement et Tests Locaux Simplifiés** :
   * Lightning AI Studio permet également un déploiement rapide des versions intermédiaires du projet, facilitant les tests et les itérations dans un environnement de production simulé. Grâce à l’accès à des GPUs puissants, vous pouvez tester les performances des modèles d'embeddings, ajuster les paramètres des modèles LLM, et monitorer le comportement du pipeline en temps réel.
   * Ce processus de test et de déploiement local réduit les délais entre chaque phase du développement, permettant de s'assurer que les modèles fonctionnent comme prévu avant de passer à un déploiement à plus grande échelle.

#### Optimisation de l’Utilisation des GPUs avec Lightning AI

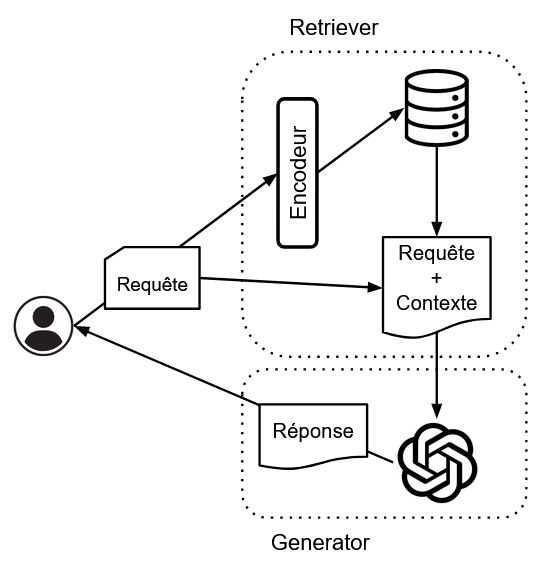
L’optimisation des ressources GPU est essentielle pour garantir des performances maximales, notamment dans les tâches lourdes comme l’entraînement de modèles, l'encodage de documents, ou l’inférence en temps réel pour la génération de texte. Lightning AI Studio assure une gestion optimisée des GPUs, permettant d’accélérer les calculs sans surcharger les ressources disponibles.

1. **Allocation Dynamique des Ressources** :
   * Lightning AI adapte dynamiquement l’allocation des GPUs en fonction des besoins de chaque étape du pipeline. Que ce soit pour la génération de réponses via GPT-4o ou l’encodage de grands volumes de documents avec Hugging Face Transformers, Lightning AI ajuste automatiquement l’utilisation des GPUs pour maximiser les performances tout en minimisant les temps d’inactivité des ressources.
   * Cette allocation dynamique permet d’utiliser les GPUs uniquement quand c’est nécessaire, réduisant ainsi les coûts d’utilisation et améliorant l’efficacité énergétique.
2. **Exécution Parallèle sur Plusieurs GPUs** :
   * Lightning AI Studio prend en charge l’exécution sur plusieurs GPUs en parallèle, ce qui est particulièrement avantageux pour des tâches massivement parallélisables, comme l’encodage de textes avec FAISS ou l’entraînement de modèles d'embeddings complexes. La plateforme distribue automatiquement les calculs entre les différents GPUs, réduisant ainsi les temps d’entraînement et de traitement.
   * Cette exécution multi-GPU est cruciale pour accélérer les tâches lourdes, en particulier lors du traitement d’énormes volumes de données textuelles nécessitant une mise à l’échelle rapide.
3. **Synchronisation des Modules et Pipelines** :
   * L’une des forces de Lightning AI réside dans sa capacité à synchroniser les différents modules de manière fluide et continue. Dans un projet comme celui-ci, où plusieurs modèles (LLM, embeddings, FAISS) et API sont utilisés simultanément, la synchronisation est primordiale pour garantir une cohérence entre les étapes du pipeline et éviter les conflits de version.
   * Lightning AI facilite cette synchronisation entre les modules tout en optimisant l’allocation des ressources pour chaque étape, de la récupération d’informations à la génération de texte.
4. **Suivi en Temps Réel des Performances** :
   * Lightning AI Studio offre également des outils intégrés de monitoring pour suivre en temps réel l'utilisation des GPUs, la consommation des ressources, et les performances globales du pipeline. Cela permet d’ajuster les paramètres en cours d’exécution pour éviter les goulots d’étranglement et garantir que le pipeline reste fluide et performant.
   * Les métriques de suivi permettent d'identifier les tâches qui consomment le plus de ressources et d'ajuster l'allocation des GPUs pour améliorer l’efficacité globale du système.

# II. Architecture RAG : Détails Techniques et Implémentation

## Présentation de l'Architecture RAG (Retrieval-Augmented Generation)

L'architecture **RAG** (Retrieval-Augmented Generation) repose sur une combinaison puissante de deux composantes principales : la récupération d'information (Retriever) et la génération de texte (Generator). Ces deux étapes permettent de répondre à des questions en recherchant des documents pertinents et en générant des réponses basées sur ces documents. Le processus se décompose en plusieurs phases, depuis l'encodage de la question jusqu'à la génération de la réponse, en passant par la recherche dans une base de données vectorielle.



L'architecture RAG comme illustré dans la *figure 35* combine la recherche de passages pertinents et la génération de texte pour fournir des réponses enrichies de contexte. Voici une vue d'ensemble des différentes phases de cette architecture :

1. **Encodage de la Question** : La question posée par l'utilisateur est encodée en un vecteur numérique à l'aide d'un modèle d'embedding (BERT).
2. **Recherche des Documents** : Le vecteur encodé est comparé aux vecteurs de documents déjà encodés dans une base de données vectorielle (via FAISS) pour trouver les documents les plus similaires.

Figure 35 : RAG Architecture

1. **Génération de la Réponse** : Les documents pertinents sont envoyés à un modèle de génération de texte (GPT-4), qui produit une réponse linguistiquement correcte en se basant sur ces documents et la question initiale.

## Étape de Récupération d'Information (Retrieval)

L'étape de récupération d'information (retrieval) est une composante critique dans l'architecture RAG (Retrieval-Augmented Generation), car elle vise à identifier les passages ou documents les plus pertinents à partir d'une vaste base de données, qui seront ensuite utilisés pour la génération de réponses. L’efficacité et la rapidité de cette étape déterminent en grande partie la performance globale du modèle RAG.

La récupération d'information repose sur plusieurs étapes interdépendantes, qui incluent l’extraction des documents sources, leur transformation en embeddings vectoriels via un modèle comme Sentence-Transformers, et leur stockage dans une base de données vectorielle optimisée pour la recherche, telle que FAISS. Ce processus garantit que les passages les plus pertinents sont identifiés et renvoyés rapidement en fonction de la requête de l’utilisateur. Nous allons détailler chaque sous-étape.

### Extraction des documents

L'extraction de texte à partir de documents PDF est l'étape initiale du processus de récupération d'information. PyMuPDF, une bibliothèque Python, est utilisée pour lire et traiter des documents PDF afin de convertir leur contenu en texte brut. Cette conversion est nécessaire pour préparer les documents à l'encodage et à la recherche vectorielle.

#### Utilisation de PyMuPDF pour la lecture et la segmentation des documents

PyMuPDF permet de manipuler des fichiers PDF en accédant à leurs pages et au texte brut contenu dans ces dernières. Voici un résumé des principales opérations effectuées avec PyMuPDF :

1. **Extraction de texte** : Le texte est extrait du PDF tout en conservant la structure du document, ce qui est important pour la segmentation et la contextualisation. Le texte est segmenté par page, ce qui permet de traiter chaque partie du document indépendamment.
2. **Manipulation des métadonnées** : PyMuPDF permet également d’accéder aux informations structurelles des PDF, comme les titres de chapitres, les sous-titres, ou les annotations. Cela peut être utilisé pour une meilleure organisation et segmentation du contenu.
   * 1. Prétraitement des documents et conversion en chunks de texte

Le prétraitement des documents est une étape cruciale dans le pipeline RAG, car il conditionne l'efficacité de l'encodage et la pertinence des passages sélectionnés lors de la recherche. Le processus de **conversion en chunks de texte** permet de découper les documents en unités de taille adaptée pour les modèles d'encodage, tout en assurant que chaque chunk contient suffisamment d'informations contextuelles.

##### Segmentation des documents en chunks

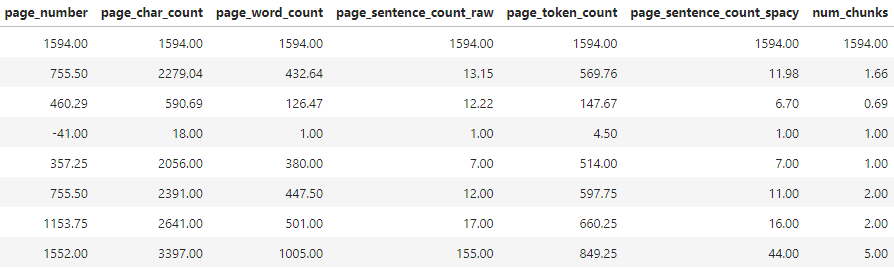
0

Table 4 : Macro Analyse des Données exploitées

La *table 4* nous démontre une marco-analyse des données exploitées dans nos documents PDF, une fois convertis en texte brut, les documents sont segmentés en **chunks** d’environ **12 phrases chacun**. Ce nombre de phrases par chunk est une moyenne empirique qui permet de conserver une bonne granularité tout en préservant la cohérence sémantique du contenu. Le but est de s'assurer que chaque chunk représente un extrait autonome du document, tout en étant suffisamment court pour être traité efficacement par les modèles LLM.

Le processus de **segmentation** peut être personnalisé en fonction de la longueur des documents, de leur complexité, et de la capacité du modèle utilisé (par exemple, la limite de tokens pouvant être traitée par GPT-4).

##### Tokenization des chunks

Une fois les chunks créés, la **tokenization** est effectuée. La tokenization consiste à transformer chaque chunk de texte en une séquence de **tokens**, c'est-à-dire de représentations

numériques des mots ou sous-mots. Ces tokens sont ensuite utilisés comme entrée pour les modèles de transformation et d'encodage, comme **all-mpnet-base-v2**.

La tokenization est une étape clé pour la préparation des données, car elle permet au modèle de comprendre et de manipuler le texte sous forme numérique. De plus, la longueur des chunks en termes de **tokens** est surveillée afin d'éviter d'excéder la capacité du modèle lors de l'encodage ou de la génération. Par exemple, GPT-4 a une limite maximale de tokens par prompt, et il est donc important de veiller à ce que les chunks soient de taille appropriée.

##### Analyse des données sur les chunks

Une analyse des données a été réalisée sur les documents segmentés afin d'optimiser la taille et le format des chunks. Voici un aperçu des statistiques tirées des documents traités :

Table 5 : Statistiques des données des documents

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of pages** | **page\_word\_count** | **page\_token\_count** | **page\_chunk\_count** |
| **1594** | **432.64 (mean)** | **569.76 (mean)** | **1.66 (mean)** |

Ces résultats affichés dans la *table 5* montrent que chaque page de notre document contient environ **432 mots**, transformés en **569 tokens** après la tokenization. En moyenne, chaque page est segmentée en **1.66 chunks**. Ces valeurs sont cruciales pour l'optimisation du processus de récupération d'information, car elles influencent la taille des données à encoder et à stocker dans FAISS.

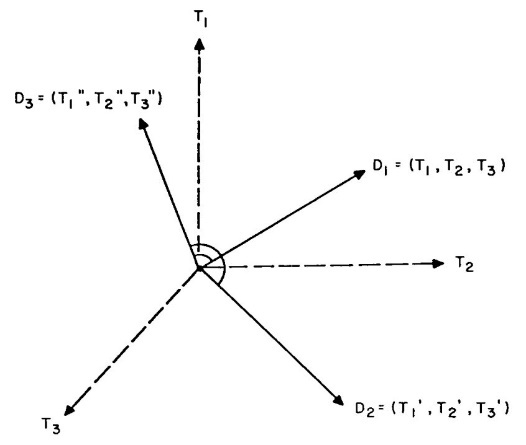
L’objectif de la segmentation est de maximiser l'efficacité du traitement tout en assurant que chaque chunk est suffisamment petit pour être manipulé rapidement par le modèle d’encodage, mais aussi suffisamment informatif pour fournir des résultats pertinents lors de la recherche.

##### Optimisation du chunking

Les **mean values** ont été utilisées pour affiner les paramètres de segmentation des documents. Par exemple, si un chunk contient trop de tokens, cela pourrait ralentir le modèle et augmenter la latence du pipeline. À l'inverse, des chunks trop courts pourraient entraîner une perte de contexte. Par conséquent, une attention particulière est portée à la création de chunks de taille optimale.

En ajustant le **page\_chunk\_count** (nombre moyen de chunks par page), nous veillons à équilibrer la quantité d'informations encodées par chunk et la performance globale du système. De plus, le **page\_token\_count** (nombre moyen de tokens par chunk) est surveillé afin de s'assurer que chaque chunk reste en dessous de la limite de tokens imposée par les modèles d'encodage et de génération.

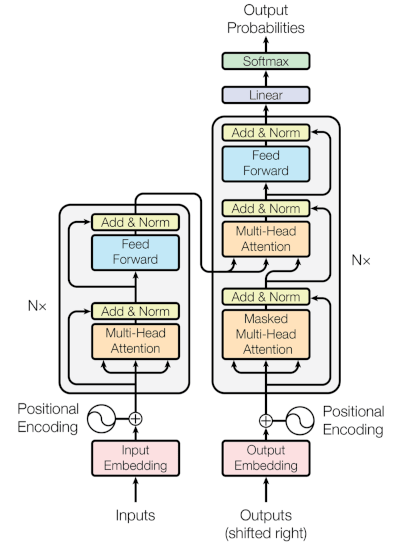
### Encodage des Documents avec Sentence-Transformers

Une fois les documents transformés en chunks, chaque chunk est encodé en un vecteur d'embedding à l'aide d'un modèle de type Sentence-Transformer. Dans notre architecture, nous utilisons le modèle all-mpnet-base-v2, qui fait partie de la famille des modèles d'encodage textuel capables de transformer des phrases ou des passages en vecteurs de haute dimension.

Dans un premier temps, nos chunks doivent être transformés en vecteurs numériques comme illustré dans la figure *36*. Cette phase d'encodage est cruciale car elle permet de représenter nos documents dans un espace vectoriel où il sera possible de calculer des similarités sémantiques avec la question de l’utilisateur.

Figure 36 : Exemple de représentation de vecteurs dans un repère orthonormé

#### Fonctionnement de l'Encodage dans BERT et Génération des embeddings

L'encodage des documents comme représenté dans la *figure 37* se fait via un modèle de type BERT (Bidirectional Encoder Representations from Transformers) qui repose sur un modèle de type transformer, et permet de comprendre le contexte d'un mot non seulement en regardant les mots qui le précèdent, mais également ceux qui le suivent. Cela s'appelle un modèle bidirectionnel. Contrairement à des modèles plus anciens comme GPT, qui sont unidirectionnels, BERT analyse les phrases dans leur globalité, ce qui permet de mieux capturer les relations entre les mots. Voici les étapes de l’encodage BERT

* **Tokenization** : Le chunk est d'abord tokenisé, c'est-à-dire que chaque mot ou sous-mot est transformé en un token. BERT utilise un tokenizer spécifique qui divise les mots en sous-unités afin de pouvoir gérer des mots rares ou inconnus.
* **Embedding des Tokens** : Chaque token est ensuite transformé en un vecteur d'embedding de dimension fixe, souvent de taille 768. Chaque mot est ainsi représenté dans un espace de haute dimension.
* **Couches de Transformations** : Les vecteurs passent ensuite par plusieurs couches de transformeurs où s'appliquent des mécanismes d'attention. Cela permet de peser l'importance relative de chaque token par rapport aux autres dans la phrase, renforçant ou atténuant certains aspects de la relation sémantique.

Figure 37 : BERT Architecture

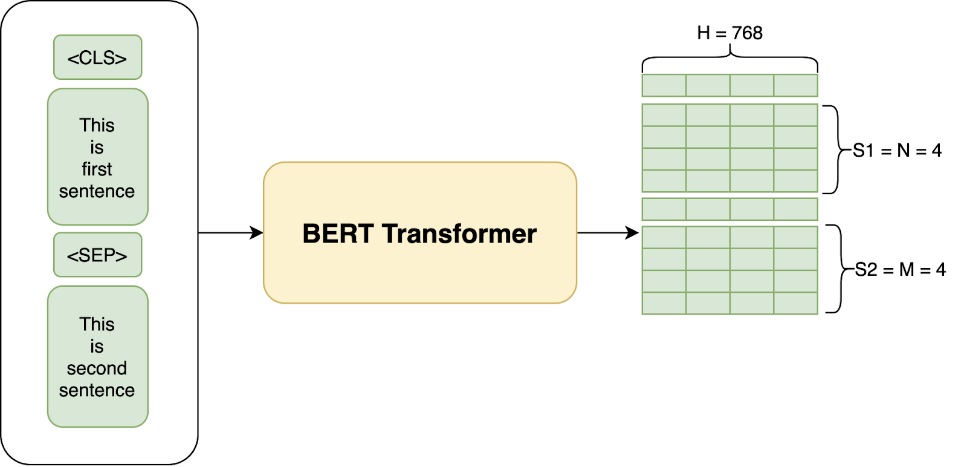
BERT prend une ou plusieurs phrases comme entrée, utilise des tokens spéciaux [CLS] pour la classification de la phrase et [SEP] pour séparer les phrases. Après l'étape d'attention multi-têtes à travers les couches du transformeur, des vecteurs d'*embedding* contextuels sont générés pour chaque token, capturant les relations entre les mots dans leur contexte, tout ce processus est bien illustré dans la *figure 38*.

Figure 38 : Étapes de Création des Embeddings avec BERT Transformer

#### Formule d’Encodage.

Pour une séquence de tokens **T = [t1, t2, ..., tn]**, l'embedding résultant E(T) est obtenu après avoir appliqué des transformations basées sur des mécanismes d'auto-attention multi-tête et des couches denses.

La sortie finale E(T), souvent de dimension 768, peut être utilisée pour des tâches de calcul vectoriel, comme la recherche de similarité sémantique.

Equation 1 : BERT Function

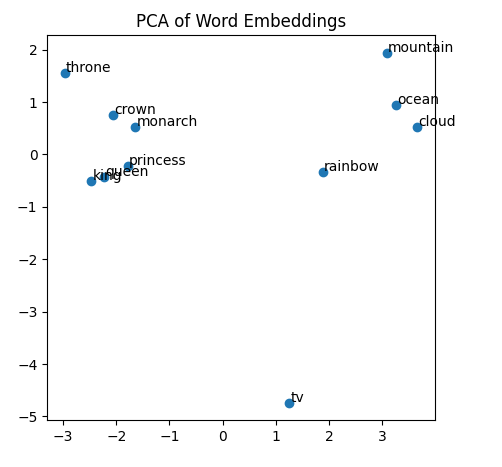
**E(T)=BERT(T)**

Les embeddings sont des représentations vectorielles qui capturent la signification sémantique des chunks. Cela signifie que des chunks ayant des significations similaires auront des vecteurs proches dans l’espace vectoriel. Le modèle optimise également ces embeddings pour qu’ils soient riches en informations sémantiques, garantissant une bonne performance lors de la recherche de documents pertinents.

#### Optimisation des Embeddings pour la Recherche Vectorielle

Après avoir généré les embeddings, il est nécessaire de les optimiser pour une recherche efficace. Cela inclut :

1. **Normalisation des embeddings** : Les embeddings sont souvent normalisés afin d'améliorer la précision des calculs de similarité. Cette normalisation permet d’obtenir des résultats plus cohérents lors des comparaisons de vecteurs dans la base de données.



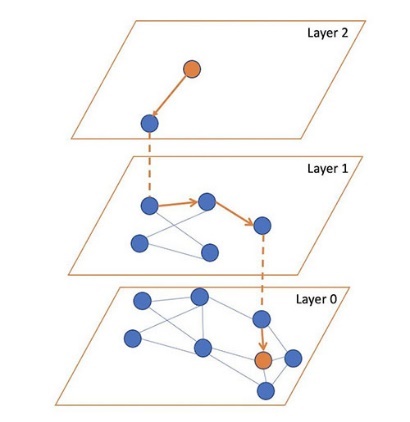
1. **Réduction de la dimensionnalité** : Si nécessaire, des techniques comme **PCA (Principal Component Analysis)** peuvent être utilisées pour réduire la dimensionnalité des embeddings, améliorant ainsi la vitesse de recherche, en particulier lorsqu’on manipule des bases de données contenant des millions de vecteurs.

Figure 39 : Exemple de Représentation des embeddings à travers PCA

### Stockage et Indexation dans FAISS

Une fois les embeddings des chunks générés, ils sont stockés dans une base de données vectorielle afin de permettre une recherche rapide. Dans notre architecture, nous utilisons **FAISS (Facebook AI Similarity Search)** pour réaliser cette tâche. FAISS est une bibliothèque optimisée pour la recherche rapide de similarité entre des vecteurs de grande dimension

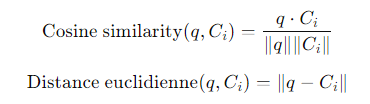
FAISS est conçu pour effectuer des recherches de **k-plus proches voisins** (k-NN) sur des vecteurs d'embeddings à haute dimension. Voici comment fonctionne FAISS :



* **Stockage des vecteurs** : Les vecteurs d'embeddings générés à partir des chunks sont stockés dans une structure d'index optimisée. FAISS offre plusieurs structures d'index, comme **IVF** (Inverted File Index) et **HNSW** (Hierarchical Navigable Small World) qui est illustrée dans la *figure 40*, qui permettent d’accélérer la recherche dans de grands ensembles de données.
* **Recherche de similarité** : Lorsqu’une requête est encodée en un vecteur d'embedding, FAISS cherche les k vecteurs les plus proches

en utilisant des mesures de distance comme la distance cosinus ou la distance euclidienne :

Figure 40 : Représentation des Différentes Couches HNSW



Equation 2 : Fonctions Cosinus Similarité et Distance Euclidienne

* Cosinus similarité

La cosinus similarité mesure l'angle entre deux vecteurs dans un espace à plusieurs dimensions. Elle est définie comme le cosinus de l'angle entre ces deux vecteurs. En termes simples, cette mesure indique à quel point deux vecteurs sont orientés de manière similaire.

La cosinus similarité donne une valeur entre -1 et 1 :

* **1** signifie que les vecteurs sont parfaitement alignés (très similaires).
* **0** signifie qu’ils sont orthogonaux (pas de similarité).
* **-1** signifie qu'ils sont opposés (complètement dissimilaires).

En FAISS, la cosinus similarité est souvent utilisée dans les tâches où l'orientation des vecteurs est plus importante que leur magnitude pour la recherche de proximité sémantique.

* Distance euclidienne

La distance euclidienne mesure la distance « géométrique » entre deux points dans un espace à plusieurs dimensions. Elle correspond à la longueur du segment de droite qui relie les deux points.

C’est une mesure de la différence absolue entre les vecteurs, plus les vecteurs sont éloignés l’un de l’autre, plus la distance est grande.

Les vecteurs qui minimisent la distance avec la requête sont considérés comme les passages les plus pertinents et sont renvoyés à l'étape suivante du pipeline.

## Étape de Génération

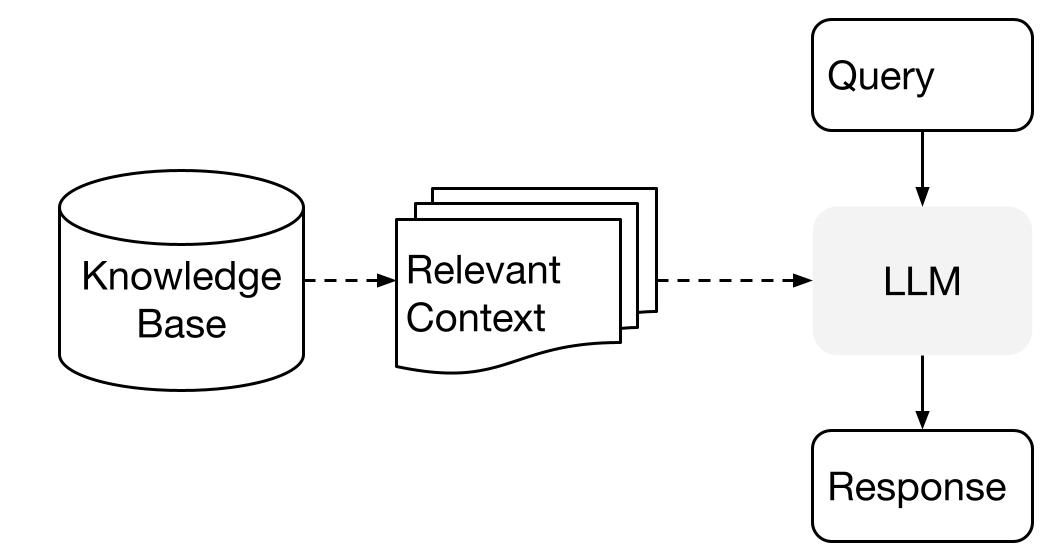
L'étape de génération comme illustré dans la *figure 41* proprement dite repose sur un modèle **LLM** capable de traiter la requête enrichie du contexte et de produire une réponse en langage naturel. Le modèle sélectionné pour la génération peut varier en fonction des spécificités de l'architecture. Dans notre projet **UNIFY.AI** est utilisé pour sélectionner le modèle optimal, en fonction de critères tels que la complexité de la tâche, la longueur de la réponse attendue ou les performances antérieures du modèle.

Figure 41 : Architecture de l'étape de Génération

### Sélection de LLM optimale pour la génération de texte

**UNIFY.AI** a fait le choix du modèle LLM le plus adapté à notre projet. Les modèles peuvent varier en termes de taille, de temps de latence, et de performance en génération de texte. Le système évalue plusieurs LLMs et sélectionne celui qui offre la meilleure balance entre rapidité d'exécution et qualité des réponses est qui est GPT-4. Ce processus se base sur des indicateurs de performance précédemment mesurés et stockés dans une base de données.

### Optimisation des paramètres de génération pour une meilleure qualité de réponse

Pour améliorer la qualité des réponses générées, plusieurs paramètres du LLM sont ajustés :

* **Température** : Ce paramètre contrôle le degré de créativité dans la génération. Une température plus basse (0.2) produira des réponses plus déterministes, tandis qu’une température plus élevée (0.7) encouragera la diversité et la créativité des réponses.
* **Top-k sampling** : Le modèle sélectionne les (k) tokens les plus probables à chaque étape de génération. Un plus grand (k) permet plus de variabilité dans les réponses.
* **Top-p sampling** : Cette technique ajuste la sélection des tokens en fonction de leur probabilité cumulative, permettant d'assurer que les réponses sont à la fois cohérentes et pertinentes.

Ces optimisations permettent de générer des réponses à la fois informatives et adaptées à la question initiale, tout en tenant compte des passages pertinents récupérés.

### Gestion des erreurs et optimisation du flux RAG

Dans un pipeline RAG, il est essentiel de prévoir des mécanismes de gestion d’erreurs et d’optimisation pour garantir la fiabilité des réponses. Cela inclut :

* **Détection des réponses incohérentes** : Si les passages récupérés sont peu pertinents ou incohérents, le modèle peut générer une réponse incorrecte. Il est possible d'intégrer des heuristiques pour évaluer la qualité de la réponse avant de la retourner à l’utilisateur.
* **Répétition des étapes de retrieval** : Dans certains cas, il peut être nécessaire de relancer une recherche avec des paramètres ajustés pour trouver des documents plus pertinents.
* **Amélioration continue** : Les modèles peuvent être ajustés au fil du temps en fonction des performances mesurées, en optimisant la taille des chunks, la méthode de sélection des passages, et les paramètres de génération.

### Analyse de la latence et performances du pipeline RAG

Le pipeline RAG, bien que puissant, peut introduire une certaine **latence** en raison du nombre d'étapes nécessaires à la récupération et à la génération. Cette latence est mesurée et optimisée en tenant compte des éléments suivants :

* **Latence de retrieval** : Le temps nécessaire pour rechercher les passages pertinents dans la base de données vectorielle (FAISS) peut être significatif, notamment pour des corpus de grande taille.
* **Latence de génération** : La génération de texte par le LLM peut varier en fonction de la complexité de la requête et du modèle utilisé. Les modèles plus grands, comme GPT-4, tendent à avoir une latence plus élevée que les modèles plus petits.

Des techniques d'optimisation comme l'indexation hiérarchique dans FAISS ou l'utilisation de modèles plus légers pour des requêtes simples peuvent être utilisées pour réduire la latence tout en conservant une bonne qualité de réponse.

## Conclusion

L'architecture RAG (Retrieval-Augmented Generation) combine efficacement la recherche d'informations et la génération de texte pour offrir des réponses plus précises et contextualisées. En exploitant des outils tels que FAISS pour la recherche vectorielle, LangChain pour l'orchestration des flux, et GPT-4 pour la génération de texte via l'API OpenAI, ce système permet d’allier récupération et génération de manière fluide.

L'optimisation des ressources via Lightning AI garantit des performances élevées et une latence réduite, notamment grâce à l’utilisation de GPUs. Le traitement des documents, l'encodage avec Sentence-Transformers et la gestion des passages pertinents sont des étapes clés qui assurent la pertinence des réponses générées.

Enfin, l'interface utilisateur conçue avec Streamlit facilite l'interaction fluide avec le système, offrant une expérience optimisée. En somme, cette architecture RAG est une solution puissante et flexible, capable de fournir des réponses contextuelles précises tout en restant scalable et performante.

CHAPITRE IV : Réalisation et   
Interaction

# Introduction à l'Interface Utilisateur de l'Application

* + 1. Objectifs et Description de l'Application

L'objectif principal de cette application est de fournir une interface utilisateur conviviale et interactive permettant d'interagir avec un chatbot spécialisé dans le développement Python. Cette interface permet à l'utilisateur de poser des questions sur des sujets variés liés à Python et de recevoir des réponses générées par un modèle d'intelligence artificielle, ici GPT-4, intégré via une API.

L'application se positionne comme un assistant virtuel qui peut aider à :

* Résoudre des problèmes techniques liés au développement Python.
* Donner des conseils et des astuces sur les meilleures pratiques en Python.
* Répondre à des questions sur des concepts spécifiques du langage.

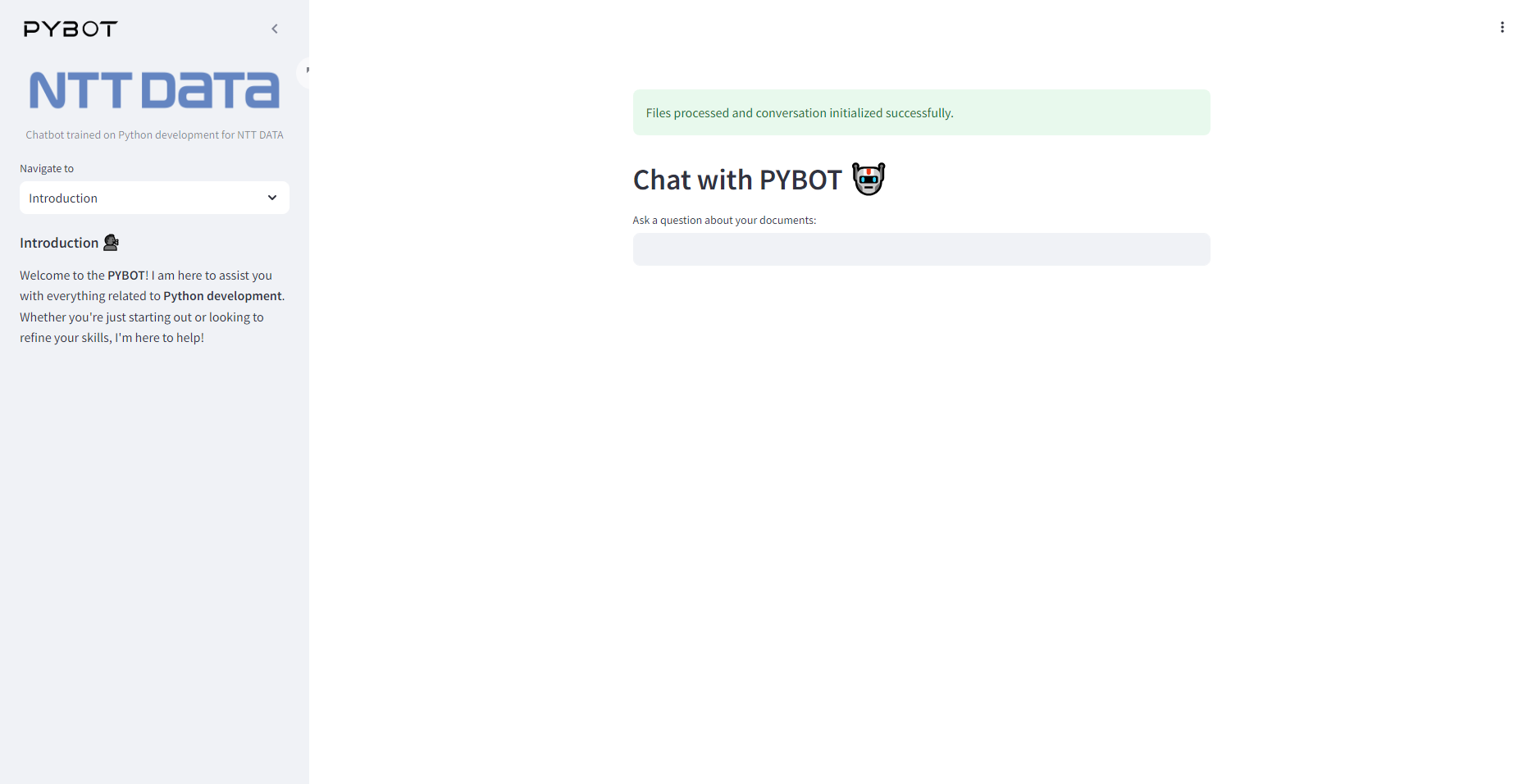


Figure 42 : Interface de PYBOT

La *figure 42* nous prévisualise l'interface qui est conçue pour être simple et intuitive, avec un espace de saisie pour poser des questions et une zone dédiée à l'affichage des réponses du chatbot. Elle offre également une navigation minimaliste avec des options pour accéder à des sections comme l’introduction ou l’aide.

Les objectifs spécifiques sont donc :

* **Faciliter l'accès aux informations techniques sur Python** à travers une interface intuitive.
* **Améliorer l'expérience utilisateur** en réduisant les efforts nécessaires pour obtenir des réponses à des questions complexes.
* **Minimiser le temps de développement frontend** en utilisant des outils comme Streamlit pour la mise en place de l'interface.

## Fonctionnalités de Streamlit utilisées dans l'application

L'application utilise plusieurs fonctionnalités clés de Streamlit pour garantir une interactivité optimale et une expérience utilisateur fluide :

**Mise à Jour en Temps Réel** : Une fonctionnalité essentielle de Streamlit est la possibilité d'afficher et de mettre à jour les informations en temps réel. Chaque fois qu'une nouvelle question est posée par l'utilisateur, l'interface est rafraîchie automatiquement avec la réponse du chatbot, sans rechargement manuel.

1. **Navigation Simplifiée** : Streamlit permet également d'ajouter des éléments de **navigation** simplifiés. Dans l'application, un menu déroulant permet à l'utilisateur de naviguer entre différentes sections, telles que l'**introduction** ou des explications plus détaillées sur le fonctionnement du chatbot.
2. **Personnalisation Visuelle** : Bien que Streamlit se concentre principalement sur la fonctionnalité, il offre une certaine flexibilité dans la personnalisation visuelle, comme l'ajout d'un logo. Cette capacité à adapter le style visuel permet de rendre l'interface plus professionnelle et engageante pour l'utilisateur final.

## Présentation des éléments de l'interface

L'interface utilisateur de l'application a été conçue pour être simple, intuitive, et réactive afin de garantir une utilisation fluide du chatbot. L'application est centrée sur l'interaction avec un chatbot développé pour répondre aux questions liées au développement Python, dans le cadre de NTT DATA.

Les éléments principaux de l'interface incluent :

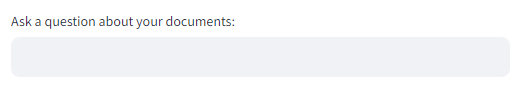
* **Champ de saisie de texte** : Ce champ permet à l'utilisateur d'entrer une question qu'il souhaite poser au chatbot. Il est placé de manière centrale pour faciliter l'accès.

Figure 43 : Figure de champ de saisie de texte

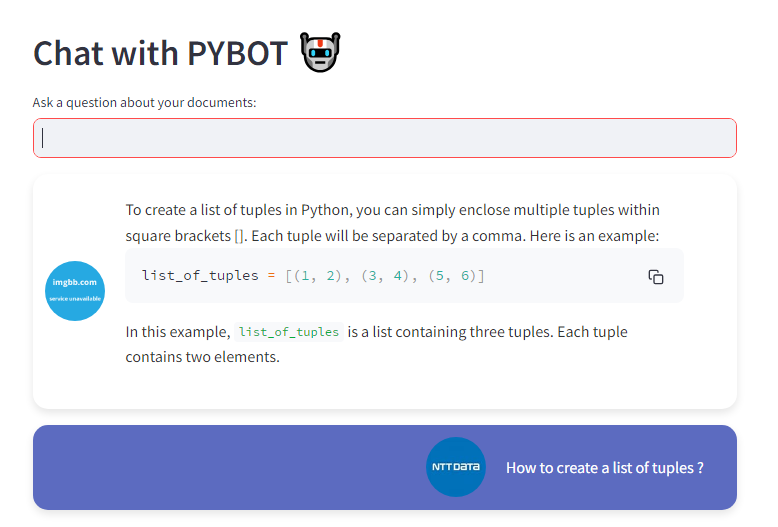
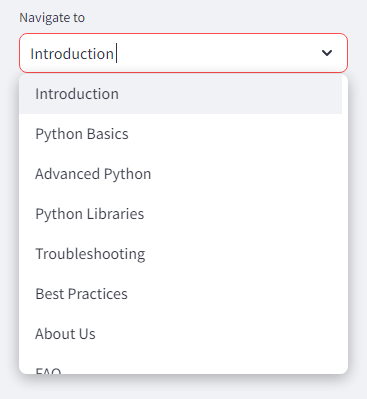
* **Zone d'affichage des réponses** (st.write()) : Cette section affiche la réponse générée par le chatbot, après traitement de la requête via l'API OpenAI. La présentation des réponses est claire, avec une typographie lisible comme représenté dans la *figure 44*.

Figure 44 : Exemple de conversation entre le PYBOT et l’Utilisateur



* **Menu de navigation** : Un menu déroulant est utilisé pour naviguer entre les différentes sections de l'application, comme "Introduction" ou d'autres sections qui enveloppent une documentation.

Figure 45 : Liste de Navigation PYBOT

L'agencement des éléments est conçu pour que l'utilisateur comprenne rapidement l'interface et puisse interagir facilement avec le chatbot, sans avoir besoin de compétences techniques avancées.

## Limitations et Contexte d'Utilisation du Chatbot

Dans cette section, nous allons démontrer que le chatbot, conçu avec Streamlit et OpenAI API, est spécifiquement configuré pour répondre uniquement aux questions liées au langage Python. Cela a été mis en place pour garantir une cohérence et une pertinence des réponses fournies par le chatbot dans le cadre du développement Python chez NTT DATA.

### Focalisation sur le Langage Python

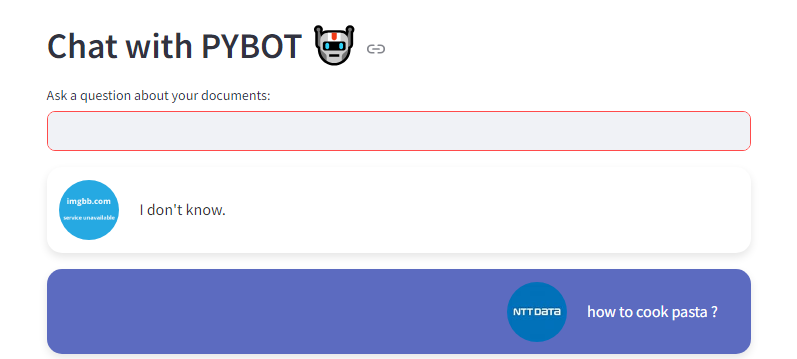
Le chatbot a été **entraîné** et configuré pour ne répondre qu'aux questions qui concernent le **développement Python**. Lorsque l'utilisateur pose une question hors du contexte Python le chatbot renvoie soit une **réponse générique,** soit un message **indiquant la limitation** du système comme représenté sur la *figure 46.*

Figure 46 : Exemple de Limitation de PYBOT par rapport au contenu du prompt

### Personnalisation et Scalabilité du Chatbot

Le chatbot peut être personnalisé pour d'autres langages ou domaines, mais dans sa configuration actuelle, il est optimisé pour Python. Cela permet une spécialisation et une efficacité accrues dans les réponses, car le modèle se concentre exclusivement sur un domaine spécifique.

Cette limitation est un choix stratégique qui permet de garantir que les réponses sont fiables, pertinentes et directement applicables aux problèmes des développeurs Python. Toutefois, cette limitation pourrait être levée à l'avenir si des besoins supplémentaires émergeaient dans d'autres domaines de programmation ou technologies.

## Adaptation du Chatbot aux Différents Niveaux de Complexité des Questions

L'une des caractéristiques principales du chatbot est sa capacité à s'adapter aux différents niveaux de difficulté des questions posées, que ce soit pour les utilisateurs débutants, intermédiaires ou avancés. Cela permet une meilleure expérience utilisateur, quelle que soit l'expertise de la personne interagissant avec le cha-tbot.

Le chatbot a été configuré pour traiter trois niveaux de questions distincts :



Figure 47 : Exemple de conversation débutante avec PYBOT

* **Débutant** : Cette réponse démontre que PYBOT sait interpréter et répondre à des questions simples tout en fournissant des exemples pratiques pour aider les débutants.

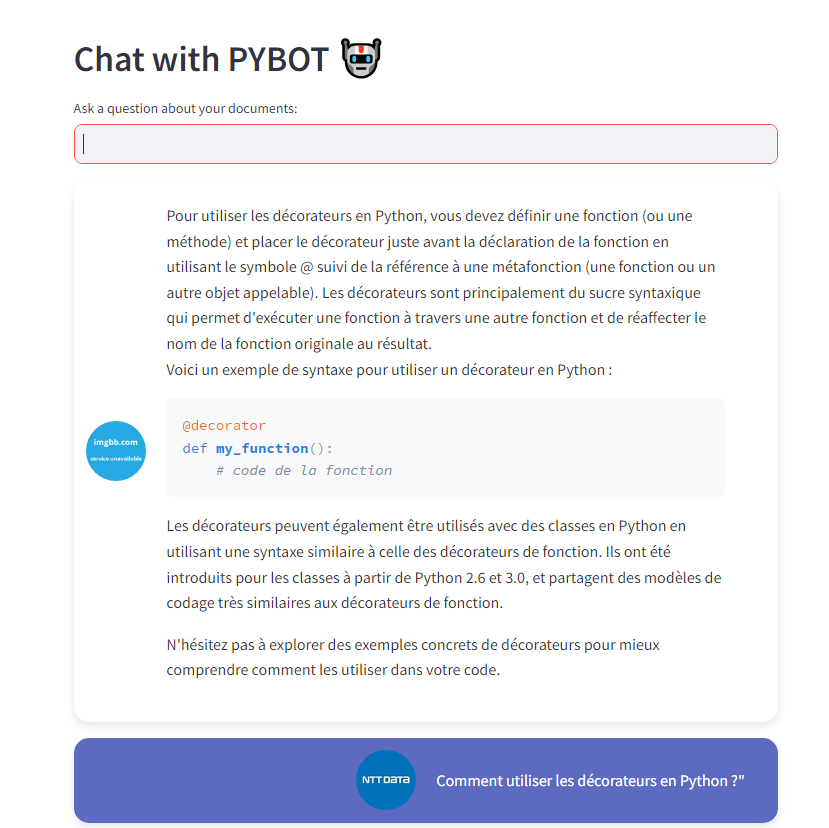


Figure 48 : Exemple de conversation Intermédiaire avec PYBOT

* **Intermédiaire** : Ce type de question pourrait inclure des concepts comme :

-L'utilisation des décorateurs

-La gestion des classes et de l'héritage

-Les fonctions anonymes (lambda)

En offrant ce niveau de complexité, PYBOT devient un outil précieux pour les développeurs de Python ayant un niveau intermédiaire.



* **Avancé** : Ce niveau de questions pourrait inclure des sujets tels que :

-La gestion de la mémoire avec le garbage collector

-Les optimisations liées à la gestion des objets Python

-Le threading et la concurrence en Python

En offrant ce niveau de support, PYBOT se positionne comme un outil capable d’accompagner les utilisateurs à tous les niveaux d'expertise.

Figure 49 : Exemple de conversation Avancé avec PYBOT

## Conclusion

L'utilisation de Streamlit a permis de développer rapidement une interface simple et fonctionnelle pour PYBOT, optimisée pour répondre aux questions liées à Python. L'intégration du backend avec LangChain et OpenAI API a rendu l'expérience utilisateur fluide et interactive, tout en gérant efficacement les questions de différents niveaux de complexité (basique, intermédiaire et avancé).

L'application a ainsi pu éviter les contraintes du développement frontend classique tout en offrant une interface performante. Enfin, l'optimisation via Lightning AI a permis d'améliorer la gestion des requêtes et les performances globales, garantissant une réponse adaptée et rapide aux utilisateurs.

CHAPITRE V : Conclusion Générale

En somme, Notre projet de développement du chatbot PYBOT, spécialisé dans l'assistance au développement Python, permet d'améliorer l'expérience des utilisateurs en répondant de manière contextuelle et adaptée à des questions de différents niveaux de complexité. Ce projet a débuté par une analyse des besoins en termes d'assistance au codage Python, suivie d'une étude approfondie des outils et technologies les plus adaptés, tels que Streamlit pour l'interface utilisateur et LangChain pour la récupération et la création des embeddings et OpenAI API pour la génération de réponses intelligentes.

On a conçu et implémenté le système de récupération d'information avec FAISS pour la recherche vectorielle et les embeddings textuels via Sentence-Transformers, afin d'optimiser la pertinence des réponses générées. L'intégration de Lightning AI a permis de garantir une utilisation efficace des ressources matérielles et d'assurer une performance optimale lors de la génération des réponses.

Durant la réalisation de ce projet, on a pu répondre de manière satisfaisante aux exigences techniques, tout en identifiant des opportunités d'amélioration future, notamment en matière de gestion des requêtes plus complexes et d'optimisation de l'interface utilisateur. Ce projet a été pour nous une occasion précieuse d'approfondir nos compétences techniques, de maîtriser de nouvelles technologies et principes en machine learning, et spécialement pour le développement d’une assistance intelligente, et de nous immerger dans un environnement de développement concret.

On espère ainsi avoir répondu aux attentes de nos encadrants, tout en créant une solution fonctionnelle et évolutive pour répondre aux besoins des utilisateurs de PYBOT dans le cadre de leur développement Python.

# Webologie

[sentence-transformers/all-mpnet-base-v2 · Hugging Face](https://huggingface.co/sentence-transformers/all-mpnet-base-v2)

<https://arxiv.org/abs/2005.11401>

<https://lilianweng.github.io/posts/2019-01-31-lm/>

<https://www.pinecone.io/learn/retrieval-augmented-generation/>

<https://medium.com/gitconnected/building-a-multi-modal-rag-system-for-visual-question-answering-52b5b0979acb>

<https://unify.ai/benchmarks/gpt-4o>

<https://www.vantagemarketresearch.com/industry-report/chatbot-market-2312>

<https://arxiv.org/abs/2005.11401>

<https://help.openai.com/en/articles/8868588-retrieval-augmented-generation-rag-and-semantic-search-for-gpts>

<https://arxiv.org/abs/1810.04805>

<https://faiss.ai/>

<https://github.com/facebookresearch/faiss>

<https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00530/114590/Improving-the-Domain-Adaptation-of-Retrieval?searchresult=1>

<https://www.turingpost.com/p/rag-tools>

<https://ar5iv.labs.arxiv.org/html/2407.01219>

[https://www.deeplearning.ai/short-courses/embedding-models-from-architecture-to-implementation](https://www.deeplearning.ai/short-courses/embedding-models-from-architecture-to-implementation/)

<https://developer-blogs.nvidia.com/wp-content/uploads/2024/05/rag-architecture.png>