Rapport de projet – portfolio avec agent IA

Introduction

Ce projet a pour objectif de concevoir une application de chat interactive à l'aide de Streamlit, capable de répondre aux questions d'un utilisateur en s'appuyant exclusivement sur des information contenue dans mon portfolio. Pour atteindre cet objectif, le projet repose sur l'approche RAG (Retrieval-Augmented Generation), qui combine une phase de recherche d'informations pertinentes et une phase de génération de réponse via un agent OpenAI. L'ensemble du système est connecté à une base vectorielle Upstash Vector, qui permet de stocker et d'interroger efficacement les documents.

L'idée principale est la suivante : lorsqu'un utilisateur pose une question, l'application ne génère pas directement une réponse. Elle commence par rechercher les passages les plus pertinents dans les documents disponibles, puis elle utilise ces passages comme contexte afin de produire une réponse cohérente, fiable et directement liée au contenu du portfolio.

Architecture et fonctionnement général

L'architecture du projet s'appuie sur une séparation claire entre l'interface utilisateur, la logique applicative et les données. Le fichier streamlit.py est responsable de l'interface graphique et de la gestion des interactions avec l'utilisateur. Il gère l'affichage du chat, la récupération des messages saisis et l'affichage des réponses retournées par l'agent. De son côté, le fichier chatbot.py contient toute la logique métier, notamment la gestion du RAG, la création de l'agent OpenAI, l'indexation des documents et la communication avec la base vectorielle Upstash.

Le code est conçu de manière modulaire afin de faciliter la compréhension et la maintenance. Par exemple, les fonctions liées à l'indexation des documents sont séparées de celles utilisées pour répondre aux questions. Cette organisation permet de relancer uniquement certaines parties du code, comme l'indexation, lorsque les documents changent, sans impacter le reste de l'application.

Le fonctionnement global débute par une phase d'indexation. Le script lit l'ensemble des fichiers Markdown présents dans le dossier data/. Chaque fichier est traité ligne par ligne afin d'identifier les sections définies par les titres Markdown. Ces sections sont ensuite transformées en chunks exploitables par la base vectorielle. Une fois les chunks générés, ils sont envoyés vers Upstash Vector grâce à une fonction dédiée utilisant une opération d'upsert, ce qui permet d'ajouter ou de mettre à jour les vecteurs existants.

Lorsqu'un utilisateur interagit avec l'application, le fichier streamlit.py appelle une fonction centrale qui transmet la question à l'agent OpenAI. Cette fonction gère également l'historique de la conversation grâce à un identifiant de réponse, ce qui permet au modèle de conserver un contexte conversationnel entre les échanges.

Principe du RAG

Le RAG repose sur la combinaison de deux mécanismes complémentaires. Le premier est la phase de retrieval, qui consiste à retrouver les informations les plus pertinentes dans les documents indexés. Le second est la phase de generation, durant laquelle le modèle OpenAI

génère une réponse en s'appuyant uniquement sur les informations fournies par la phase de recherche.

Cette approche est particulièrement utile car le modèle de langage ne possède pas de connaissance préalable du contenu du portfolio. Sans RAG, les réponses seraient soit imprécises, soit basées sur des suppositions. Grâce à ce mécanisme, les réponses sont directement ancrées dans les données fournies par l'utilisateur, ce qui améliore fortement leur pertinence et leur fiabilité.

Base vectorielle Upstash Vector

La base vectorielle Upstash Vector joue un rôle central dans le projet. Elle permet de stocker les documents sous forme de vecteurs d'embeddings, c'est-à-dire des représentations numériques du sens des textes. Lorsqu'une requête est effectuée, la base compare le vecteur de la question avec ceux des documents afin de retrouver les textes les plus proches sémantiquement.

Dans ce projet, la recherche est dite hybride. Elle combine une recherche dense, basée sur les embeddings, qui capture le sens global des phrases, et une recherche sparse, basée sur un algorithme de type BM25, qui se concentre sur les mots-clés exacts. Cette combinaison permet d'obtenir des résultats plus pertinents qu'une simple recherche textuelle classique.

Découpage des documents

Les documents Markdown ne sont pas stockés tels quels dans la base vectorielle. En effet, un fichier complet peut être trop long et manquer de précision lors des recherches. Pour résoudre ce problème, les documents sont découpés en plusieurs segments appelés chunks.

Le découpage est réalisé en s'appuyant sur la structure Markdown des fichiers, notamment les titres et sous-titres. Chaque section devient ainsi un chunk indépendant. Cette méthode permet d'améliorer la précision des recherches et de fournir à l'agent des extraits plus ciblés, ce qui se traduit par des réponses plus pertinentes.

Indexation des données

L'indexation correspond à l'envoi des chunks dans la base vectorielle. Pour chaque document, les différentes sections découpées sont transformées en vecteurs contenant à la fois le texte et des métadonnées. Ces métadonnées indiquent notamment le fichier d'origine, le numéro du chunk et le titre de la section associée.

Une fois les vecteurs créés, ils sont envoyés dans Upstash Vector. Cette étape permet de rendre les documents interrogeables par la suite. L'indexation n'a pas besoin d'être relancée à chaque utilisation de l'application, sauf si le contenu des documents est modifié.

Recherche et génération de réponse

L'agent OpenAI n'accède pas directement à la base vectorielle. Il utilise une fonction intermédiaire, appelée tool de recherche, qui se charge d'interroger Upstash Vector à partir de la question de l'utilisateur. Cette fonction retourne les passages les plus pertinents sous forme structurée, accompagnés de leurs métadonnées.

Ces passages sont ensuite transmis à l'agent comme contexte. Le modèle OpenAI génère alors une réponse en langage naturel en se basant uniquement sur ces informations. L'agent est

configuré avec une température faible afin de produire des réponses stables, factuelles et peu aléatoires. De plus, le système conserve un identifiant de réponse permettant de chaîner les échanges et de maintenir une continuité dans la conversation.

Interface utilisateur Streamlit

L'interface Streamlit constitue le point d'entrée de l'utilisateur. Lors du premier chargement de l'application, la connexion à la base vectorielle est établie et l'agent OpenAI est initialisé. Ces éléments sont ensuite stockés dans la session Streamlit afin d'éviter de les recréer à chaque interaction.

À chaque message envoyé par l'utilisateur, l'application appelle la logique de question-réponse, affiche la réponse générée et met à jour l'historique de la conversation. Cette organisation permet d'offrir une expérience fluide et interactive.

Déploiement et configuration

Le projet nécessite plusieurs variables d'environnement pour fonctionner correctement, notamment la clé d'API OpenAI et les informations de connexion à Upstash Vector. En environnement local, ces variables sont définies dans un fichier de configuration dédié, tandis qu'en production sur Streamlit Community Cloud, elles sont renseignées dans les paramètres de l'application.

Une fois les dépendances installées et les documents indexés, l'application peut être lancée via Streamlit. Le déploiement sur Streamlit Cloud permet de rendre le chat accessible en ligne sans configuration complexe côté utilisateur.

Conclusion

Ce projet met en œuvre une application de chat intelligente capable de répondre à partir de données issue de mon portfolio grâce à une architecture RAG bien structurée. L'utilisation d'une base vectorielle permet une recherche efficace et pertinente, tandis que l'agent OpenAI assure la génération de réponses claires.