**Rapport développement et Déploiement du Chatbot RAG**

**1. Introduction**

L’objectif de ce projet était de développer un chatbot intelligent capable d’interroger des données provenant de différentes sources externes, notamment **Google Docs**, **Google Sheets** et des conversations **WhatsApp**, puis de le déployer sur le cloud afin de le rendre accessible en ligne.  
Pour optimiser la performance et réduire les coûts d’utilisation des modèles d’IA, une architecture de **Retrieval-Augmented Generation (RAG)** a été mise en place.

**2. Connexion aux sources de données**

**2.1 Création du compte Google Cloud**

* Création d’un compte **Google Cloud Platform**.
* Génération d’une **clé d’authentification JSON** permettant d’accéder aux API Google Docs et Google Sheets.
* Une adresse Gmail associée au compte de service a été automatiquement créée.

**2.2 Attribution des permissions**

* Ajout de l’adresse Gmail générée dans **les permissions en lecture seule** de tous les Google Docs et Google Sheets utilisés.
* Objectif : autoriser le chatbot à lire le contenu sans pouvoir modifier les documents.

**2.3 Test de lecture sur Google Colab**

* Utilisation d’un script Python sur **Google Colab** pour valider l’accès aux documents.
* Les tests se sont révélés concluants : le contenu des documents était bien lu et exploitable.

**3. Première intégration avec un modèle de langage**

**3.1 Modèle initial – LLaMA 3 (gratuit)**

* Connexion du pipeline de lecture au modèle **LLaMA 3** gratuit.
* Résultat : performances insuffisantes (manque de précision et de compréhension contextuelle).

**3.2 Intégration de données WhatsApp**

* Les messages WhatsApp ont été copiés et centralisés dans un **Google Doc** afin qu’ils soient inclus dans les sources.

**3.3 Migration vers OpenAI GPT-4o**

* Passage au modèle **OpenAI GPT-4o**, plus performant pour l’interprétation des données.
* Problème rencontré : **dépassement de tokens** lorsque toutes les données étaient envoyées à chaque question.

**4. Mise en place de l’architecture RAG**

**4.1 Justification**

* Sans RAG : toutes les données devaient être envoyées à chaque requête → **surcharge de tokens** et coût élevé.
* Avec RAG : seules les données pertinentes sont envoyées au modèle → optimisation des performances et réduction des coûts.

**4.2 Développement du RAG**

* Encodage des documents en vecteurs via **SentenceTransformers**.
* Indexation avec **FAISS** pour la recherche rapide des passages pertinents.
* À chaque question :
  1. Encodage de la question.
  2. Recherche des *k* passages les plus pertinents.
  3. Envoi de ces passages au modèle GPT-4o pour générer la réponse.

**5. Problèmes rencontrés et optimisations**

**5.1 Déploiement direct avec connexion en live aux sources**

* Tentatives de connexion directe aux Google Docs depuis le cloud (Render).
* Résultat : erreurs répétées, notamment sur la gestion des **clés API** et **permissions**.

**5.2 Problèmes de mémoire sur Render**

* Lors du premier déploiement, le site ne s’affichait pas et renvoyait uniquement des logs d’erreur.
* Cause : **charge mémoire trop importante** due au chargement en live de toutes les données.

**5.3 Solution – Prétraitement en local**

* Exécution du script de génération des fichiers (.pkl et .faiss) **en local sur Google Colab**.
* Sauvegarde des fichiers générés :
  + all\_texts.pkl
  + all\_sources.pkl
  + all\_embeddings.pkl
  + rag\_index.faiss
* Ajout de ces fichiers directement dans le projet GitHub pour qu’ils soient chargés depuis le disque lors du déploiement.

**6. Procédure pour mettre à jour les données**

1. Lancer le **notebook Google Colab**.
2. Lire les documents Google Docs et Google Sheets.
3. Générer les fichiers .pkl et .faiss.
4. Télécharger les fichiers générés.
5. Les ajouter dans le projet GitHub.
6. Déployer à nouveau sur **Render**.

**7. Perspectives et améliorations**

* **Nettoyage des données** : prétraiter les documents pour retirer les informations inutiles ou redondantes.
* **Optimisation mémoire** : réduire la taille des embeddings et limiter le nombre de passages indexés.
* **Scalabilité** : configurer Render pour supporter plusieurs utilisateurs simultanés sans crash.
* **Interface utilisateur** : améliorer l’affichage et la gestion des sources dans les réponses.

**8. Conclusion**

Ce projet a permis de mettre en place un chatbot basé sur un RAG, capable d’exploiter efficacement des données issues de Google Docs, Google Sheets et WhatsApp.  
Le déploiement sur Render a nécessité des ajustements pour optimiser les performances et éviter les dépassements mémoire, notamment en séparant la phase de collecte/traitement des données (en local) et la phase d’interrogation du modèle (en ligne).  
La prochaine étape consistera à améliorer la robustesse, la scalabilité et la qualité des données pour un usage plus intensif.

Une image contenant texte, capture d’écran, logiciel, nombre

Le contenu généré par l’IA peut être incorrect.

Github link : <https://github.com/Thano02/nobo_chatbot>

Lien pour gérer les dépenses en token de openai : <https://platform.openai.com/usage>

Lien google cloud : Private

Lien du notebook google collab : Private