RAPPORT DE PROJET : CHATBOT RAG POUR LE DAKAR INSTITUTE OF TECHNOLOGY (DIT)

Malick Royce LAYINDE

TABLE DES MATIÈRES

PRESENTATION DU PROJET

Contexte et Objectif

Le projet consiste à développer un chatbot intelligent pour le **Dakar Institute of Technology (DIT)**, basé sur l'approche Retrieval-Augmented Generation (RAG). L'objectif principal est de fournir un assistant virtuel capable de répondre avec précision aux questions des visiteurs du site du DIT en s'appuyant sur des documents internes indexés.

Pourquoi ce chatbot est pertinent pour le DIT ?

Le DIT est un établissement spécialisé dans la formation en Intelligence Artificielle et Big Data. Un grand nombre de visiteurs, notamment des futurs étudiants, parents et partenaires, cherchent des informations sur les programmes proposés, les conditions d'admission et les formations disponibles.

Un chatbot permettrait donc :

1. **Un accès rapide aux informations** : Les visiteurs obtiennent des réponses instantanées sans avoir à naviguer sur tout le site.
2. **Une meilleure expérience utilisateur** : Il répond aux questions 24/7, sans intervention humaine.
3. **Une automatisation du support** : Réduction du volume de questions répétitives traitées par l'administration.
4. **Un engagement renforcé** : Il facilite la découverte des formations et améliore l'image technologique du DIT.

DEVELOPPEMENT DU CHATBOT

Collecte et Préparation des Données

Nous avons collecté et préparé les données en combinant plusieurs méthodes afin d'assurer que le chatbot puisse répondre de manière précise et complète aux questions des visiteurs du DIT.

Extraction manuelle et web scraping :

- Nous avons récupéré des informations pertinentes manuellement depuis le site officiel du DIT.
- Pour gagner du temps, nous avons utilisé du web scraping afin d'extraire des sections utiles directement depuis certaines pages web.

Consultation de sources fiables :

- Nous avons complété les informations en nous basant sur des sources sûres telles que Wikipedia, notamment pour obtenir une vue globale et des précisions sur l'établissement.

Utilisation de documents officiels :

- Nous avons rassemblé des documents internes du DIT, tels que :
    - **Plaquettes et brochures** présentant l'école et ses programmes.
    - **Maquettes de formation** détaillant les cursus.
    - **Documents académiques et réglementaires** validés par l'institution.
    - Etc,

Toutes ces sources ont été indexées et transformées en une base de connaissances consultable par le chatbot.

Segmentation des documents (Tokenization & Chunking)

Les modèles de recherche sémantique ne peuvent pas traiter de longs documents en une seule fois. Nous avons donc découpé les textes en petites unités appelées chunks (blocs de texte), afin de les rendre facilement interrogeables.

Nous avons utilisé **RecursiveCharacterTextSplitter** de LangChain, qui segmente les textes en blocs de 500 caractères avec un chevauchement de 50 caractères pour conserver le contexte.

Indexation des documents (Vector Database)

Nous avons utilisé FAISS comme base de données vectorielle. Les textes ont été transformés en vecteurs numériques grâce au modèle GoogleGenerativeAIEmbeddings. Ces vecteurs permettent de comparer la similarité entre une requête et les documents stockés.

Tout cela a été résumer en une fonction que voici :

```python
def preparer_base(dir_path=None, save_path="faiss_index"):
    """Charge les documents, génère les embeddings et sauvegarde la base FAISS."""

    # 1. Chargement des documents
    documents = []

    if dir_path:
        dLoader = DirectoryLoader(dir_path, glob="*.txt", show_progress=True)
        documents += dLoader.load()

    if not documents:
        raise ValueError("Aucun document trouvé pour créer la base.")

    # 2. Découpage en chunks
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
    chunks = text_splitter.split_documents(documents)

    # 3. Conversion en embeddings
    embedding_model = GoogleGenerativeAIEmbeddings(model=EMBEDDING_MODEL, google_api_key=API_KEY)
    vector_store = FAISS.from_documents(chunks, embedding_model)

    # 4. Sauvegarde de FAISS
    vector_store.save_local(save_path)
    print(f" Base FAISS sauvegardée dans {save_path}")

# utilisation
dir_path = "corpus_txt"

preparer_base(dir_path)
```

Implémentation du Module Retriever

Le Retriever est responsable de retrouver les passages les plus pertinents à partir de la base FAISS lorsque l'utilisateur pose une question.

Mécanisme de recherche (Similarity Search)

Lorsqu'un utilisateur envoie une requête, celle-ci est convertie en embedding (vecteur numérique) et comparée aux vecteurs des documents indexés. Nous avons utilisé similarity_search_with_score() qui retourne les 3 meilleurs documents avec un score de pertinence.

```python
@app.post("/chat")
def chatbot_interaction(user_message: UserMessage):

    query = user_message.question

    # Récupérer les documents pertinents
    retrieved_docs = vector_store.similarity_search(query, k=3)
    if not retrieved_docs:
        return {"reponse": "Je suis désolé, je ne dispose pas d'informations pertinentes sur ce sujet. "
        "N'hésiter pas à nous contacter par mail info@dit.sn ."}


    # Sélectionner le meilleur document
    best_doc = retrieved_docs[0]
    document_source = best_doc.metadata.get("source", "Document inconnu")
```

Implémentation du Module de Génération

Une fois les documents trouvés, le module de génération construit une réponse fluide et naturelle en s'appuyant sur le contexte récupéré.

Construction du prompt

Le prompt est structuré pour donner au modèle un contexte clair, éviter les réponses vagues et garantir que le chatbot agit comme un assistant du DIT.

```
51
52      # contexte de réponse
53      context = "\n\n".join([doc.page_content for doc in retrieved_docs])
54      prompt = f"""Tu es Nico, le chatbot officiel du Dakar Institute of Technology (DIT).
55      Ta mission est d'aider les visiteurs du site à trouver des informations sur l'école de manière
56      claire et professionnelle.\n**Document analysé** : {document_source}\n
57      **Règles** :Si l'information est disponible, donne-la directement sans mentionner comment tu l'as
58      obtenue. Ne redirige jamais vers un site web ou une autre source externe.Si une information semble
59      incomplète, donne ce que tu sais mais reste dans ton rôle d'assistant du DIT.Utilise un ton
60      professionnel et institutionnel, comme si tu faisais partie du personnel du DIT.Si tu ne trouves
61      pas la réponse, indique simplement que tu ne disposes pas de cette information actuellement
62      (pas besoin de préciser que ce n'est pas dans les documents que tu as), et invite l'utilisateur à
63      nous contacter.Réponds de manière fluide et naturelle, comme un assistant humain du DIT.\n**
64      Extrait du document** :{context}\n**Question utilisateur** :{query}\n**Réponse détaillée** :"""
65
```

Génération de la réponse avec Google Gemini

Nous avons utilisé Gemini 1.5 Flash pour générer les réponses du chatbot.

```
66      # Génération de la réponse avec Gemini
67      chat_model = ChatGoogleGenerativeAI(model=LLM_MODEL, api_key=API_KEY)
68      messages = [
69          SystemMessage(content="Tu es Nico, le chatbot du DIT. Donne des réponses claires et utiles."),
70          HumanMessage(content=prompt)
71      ]
72
73      response = chat_model(messages)
74      return {"reponse": response.content}
```

Optimisation des réponses

Afin d'optimiser les réponses, nous avons procéder comme suit :

- **Post-traitement** : Suppression des mentions inutiles ("Selon les documents que j'ai analysés…")
- **Formatage** : Ajout d'une structuration claire pour les listes, définitions, etc.

Exemple de réponse générée :

Avant optimisation :

<< *"Selon les documents que j'ai lus, le DIT propose des masters. Voici les détails disponibles sur leur site…"* >>

Après optimisation :

<< "Le Dakar Institute of Technology (DIT) propose plusieurs masters, notamment :

- Master en Intelligence Artificielle
- Master en Finance Digitale

Pour plus d'informations, contactez l'administration à info@dit.sn." >>

INSTRUCTIONS POUR EXECUTER ET TESTER LE CHATBOT

Exécution avec Docker

Le chatbot est disponible sous forme d'une **image Docker**, ce qui facilite son exécution sur n'importe quelle machine.

1 Récupérer l'image Docker depuis Docker Hub

Exécutez cette commande pour la télécharger :

docker pull roy61/chatbotdit-rag

2 Lancer le conteneur Docker

Utilisez cette commande pour exécuter le chatbot en arrière-plan sur le port 8000 :

docker run -d -p 8000:8000 roy61/chatbotdit-rag

✓ Le chatbot est maintenant accessible à l'adresse : **http://127.0.0.1:8000**

Test du Chatbot

Le chatbot peut être testé de plusieurs manières :

1 Test avec un navigateur

Accédez à **http://127.0.0.1:8000** . Vous verrez un message de bienvenue.

2 Test avec l'interface Swagger

FastAPI fournit une interface de test prête à l'emploi : **http://127.0.0.1:8000/docs**

Ici, vous pouvez interagir avec l'endpoint /chat directement.

3 Test avec Postman ou cURL

Si vous utilisez **Postman**, créez une requête POST vers **http://127.0.0.1:8000/chat** avec ce body :

{

"question": "Quels masters propose le DIT ?"

}

Si vous préférez utiliser cURL :

curl -X POST "http://127.0.0.1:8000/chat" -H "Content-Type: application/json" -d '{"question": "Quels masters propose le DIT ?"}'

ANNEXE

- **Lien Github du projet**
- **Nom de l’image Docker dispo sur Docker hub :** *roy61/chatbotdit-rag*