# Fiche Documentaire du Code : *Response*

**Titre du projet** :
*Analyse personnalisée des articles scientifiques à l'aide des LLM et du paradigme RAG*

**Auteurs** :

- ***Bouaamar Ayman*** - Responsable de l'utilisation d'Ollama pour la génération de réponses.
- ***Baticha Youssef*** - Chargé de l'intégration du backend avec Chroma et Ollama pour la gestion des requêtes et la génération des réponses.
- ***Rahhou Hakim*** - Chargé du développement et de l'intégration de l'interface utilisateur avec Streamlit.

**Description du code :**
Ce notebook utilise le modèle de langage Ollama (LLM) pour générer des réponses personnalisées basées sur des paragraphes similaires, extraits à partir d'une base de données vectorielle Chroma contenant des embeddings. Le système permet à l'utilisateur de sélectionner un profil (étudiant ou chercheur), puis de poser une question à laquelle le système répond en fonction du profil choisi. L'intégration d'Ollama via Docker est utilisée pour générer les réponses en exploitant les informations contextualisées récupérées depuis Chroma.

**Structure du code :**

1. **Imports & Installation des dépendances :**
    - langchain_community.embeddings.ollama, chroma, requests, etc. sont utilisés pour la génération d'embeddings et l'intégration du modèle Ollama via Docker.
2. **Fonctions principales :**
    - get_distilbert_embeddings() : Génère les embeddings à partir des textes à l'aide de DistilBERT.
    - extract_information_from_chroma() : Récupère les documents les plus similaires à la requête depuis Chroma.
    - query_ollama_with_context() : Interroge l'API Ollama via Docker en passant un prompt généré.
    - query_rag_with_context() : Fonction principale qui combine la récupération depuis Chroma et la génération de réponses avec Ollama.
3. **Entrées et sorties :**

- Entrée : Une requête textuelle de l'utilisateur, profil (chercheur ou étudiant).
- Sortie : Une réponse générée par Ollama basée sur les paragraphes les plus similaires extraits de Chroma.

**Instructions d'installation :**

1. **Prérequis :**
   - Python 3.10
   - Installation locale d'Ollama via Docker
   - Streamlit pour l'interface utilisateur
2. **Installation des dépendances :**

```
pip install langchain chromadb langchain_community transformers
```

3. **Exécution :**
   - Lancer Docker et s'assurer que le serveur Ollama fonctionne sur localhost:11434.
   - Utiliser streamlit run response.py pour démarrer l'interface utilisateur.

**Prérequis :**

- Ollama installé via Docker.
- ChromaDB configuré pour stocker et récupérer des embeddings.
- Modules Python : langchain, chroma, requests, transformers.

**Fonctionnement du code :**

1. **Génération d'embeddings :**
   Le modèle DistilBERT est utilisé pour encoder la requête de l'utilisateur en vecteurs d'embeddings. Ces vecteurs sont ensuite comparés aux paragraphes stockés dans Chroma.
2. **Extraction du contexte :**
   La fonction extract_information_from_chroma() interroge la base Chroma en fonction des embeddings et récupère les paragraphes les plus pertinents.
3. **Génération de réponses avec Ollama :**
   Ollama, un modèle de langage installé localement, reçoit le contexte et génère une réponse adaptée au profil de l'utilisateur.

**Entrées et sorties :**

- **Entrée :**

    - query : Requête posée par l'utilisateur.
    - profile : Profil de l'utilisateur (étudiant ou chercheur).
- **Sortie :**
    - Réponse générée par Ollama, adaptée au profil et au contexte récupéré depuis Chroma.

**Exemples d'utilisation :**

```
query = """
Advanced techniques for optimizing systolic engines in FPGAs,
including the use of DSP48E2 blocks in neural network architectures.
"""
response = query_rag_with_context(query, "researcher")
print(response)
```

**Tests et validation :**
Des tests manuels ont été effectués en local en utilisant des requêtes scientifiques sur des architectures FPGA. Les résultats ont montré une correspondance efficace entre les requêtes et les réponses générées.

**Problèmes connus :**

- **Connexion Docker :** L'API Ollama nécessite Docker pour fonctionner, donc le code ne peut pas s'exécuter sans cette configuration.
- **Limite d'intégration de Chroma :** Si les embeddings dans Chroma ne sont pas correctement indexés, le système peut ne pas retourner les paragraphes les plus pertinents.