Fiche Documentaire du Code : *Search and Recovery*

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

Auteurs : ***Elmzouri Fatimazahra*** – Responsable du développement des outils d'analyse

Description du code : Le notebook *Search and Recovery* met en œuvre un système d'interrogation basé sur des *embeddings* pour extraire des sections d'articles scientifiques en fonction d'une requête utilisateur. Il utilise **ChromaDB**, une base de données vectorielle, pour rechercher les passages les plus similaires dans une collection d'articles scientifiques prétraités. L'objectif est d'intégrer cette extraction d'informations dans une interface utilisateur pour générer des réponses personnalisées basées sur le profil de l'utilisateur (étudiant ou chercheur).

Structure du code :

- get_distilbert_embeddings(text): Fonction générant des *embeddings* à partir de texte en utilisant **DistilBERT**.
- extract_information_from_chroma(query, num_results=5): Interroge la base de données *Chroma* et structure les résultats dans un *DataFrame*.
- collection: L'objet Chroma utilisé pour stocker les embeddings des articles scientifiques.
- query: Exemple de requête complexe sur les techniques d'optimisation des moteurs systoliques pour les *FPGAs*.

Instructions d'installation :

1. Monter Google Drive pour stocker les données localement et persister les résultats.
2. Installation des dépendances nécessaires, incluant **ChromaDB** et **transformers** :

```
!pip install chromadb
!pip install transformers
```

Prérequis :

- **Python 3.x**
- **ChromaDB 0.5.9**
- **Transformers (Hugging Face)** pour générer les embeddings avec DistilBERT.
- **Pandas** pour manipuler les résultats et structurer les données.
- **Google Drive** monté pour stocker les fichiers.

Fonctionnement du code :

1. **Initialisation de ChromaDB** :
   Le client persistant Chroma est initialisé pour accéder aux embeddings stockés dans Google Drive.

```
client = chromadb.PersistentClient(path=CHROMA_PATH)
collection = client.get_or_create_collection("Article_Embedding")
```

2. **Génération des embeddings avec DistilBERT** :
   Une fonction génère des *embeddings* pour la requête utilisateur afin de comparer les passages d'articles scientifiques.

```
def get_distilbert_embeddings(text):
    inputs = tokenizer(text, return_tensors='pt', max_length=512, truncation=True
    outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).detach().numpy().reshape(-1)
```

3. **Interrogation et extraction des données de Chroma** :
   Cette fonction interroge la base de données *Chroma* pour extraire les sections pertinentes d'articles en fonction de la requête.

```
def extract_information_from_chroma(query, num_results=5):
    # Générer l'embedding pour la requête
    query_embedding = get_distilbert_embeddings(query)

    # Rechercher les documents les plus similaires dans Chroma
    results = collection.query(
        query_embeddings=[query_embedding.tolist()],
        n_results=num_results
    )
```

4. **Résultats** :
   Le DataFrame final contient les sections et leurs métadonnées (nom de fichier, paragraphe) correspondant aux passages les plus similaires à la requête.

Entrées et sorties :

- **Entrée** : Une requête utilisateur sous forme de texte détaillé.
- **Sortie** : Un DataFrame contenant les sections pertinentes extraites des articles, accompagnées de leurs métadonnées.

Exemples d'utilisation :
Voici un exemple d'interrogation :

```
query = """
Advanced techniques for optimizing systolic engines in FPGAs...
"""
df = extract_information_from_chroma(query)
print(df)
```

Problèmes connus :

- Le modèle DistilBERT est limité par la taille maximale des *tokens* (512), ce qui peut entraîner une troncature des textes trop longs.
- Les performances peuvent varier en fonction du nombre d'articles dans la base de données vectorielle et de la complexité de la requête.