Rapport du Projet :

ChatBot de recommandation de films

Matière : NLP

Encadrant : *Hamza ASSOUANE*

Etudiants : *Es-Sadany Yassine*

Groupe : *ING3 IA 2*

06 Décembre 2023

Table des matières

1 Introduction

Ce projet a pour but de créer un chatbot de recommandation de films selon un film référence ou selon une description. Dans ce projet, on a utiliser 2 LLM open source pour la partie conversationnelle **(Llama-2-13B-chat et GPT-TURBO-3.5)** : ainsi qu'on a comparé entre 2 approches pour le même LLM Llama-2 : **HuggingFacePipeline** et **CTransformers** .

2 Description de la Solution/Travail

2.1 Modèles et approches Utilisés

la technique utilisée dans ce porjet est appelée le RAG(Retrieval-Augmented Generation), il consiste à chercher des embeedings dans le vectorestore (base de données des embeedings) les plus similaires. elle est bien expliquer dans le diagramme fonctionnel en suite. Dans un premier temps, j'ai choisi de travailler avec un d'OpenAi (GPT-TURBO-3.5), et un llm de Llama2 (Llama-2-13B-chat) avec

2.2 Dataset Utilisé

Le dataset utilisé est extrait (toutes les données ou la colonnes split = train) d'un fichier csv télechargé sur via le lien `https://www.kaggle.com/datasets/cryptexcode/mpst-movie-plot-synopses-with-tags` qui contient 6 colonnes :

- **imdb_id** : L'id IMDB du film
- **title** : Titre du film
- **plot_synopsis** : Description du film
- **tags** : les catégories du films (comedy, action ...)
- **split** : (train, test, val)
- **synopsis_source** : (imdb, wikipedea)

le dataset a était convertit un nouveau dataset qui contient 2 colones : l'imdb_id du film et un colonne movie qui fusion les 5 auters colonnes décrits dessus. La taille du fichier final est de 46MB (ceci a affecté la rapidité du chatbot)

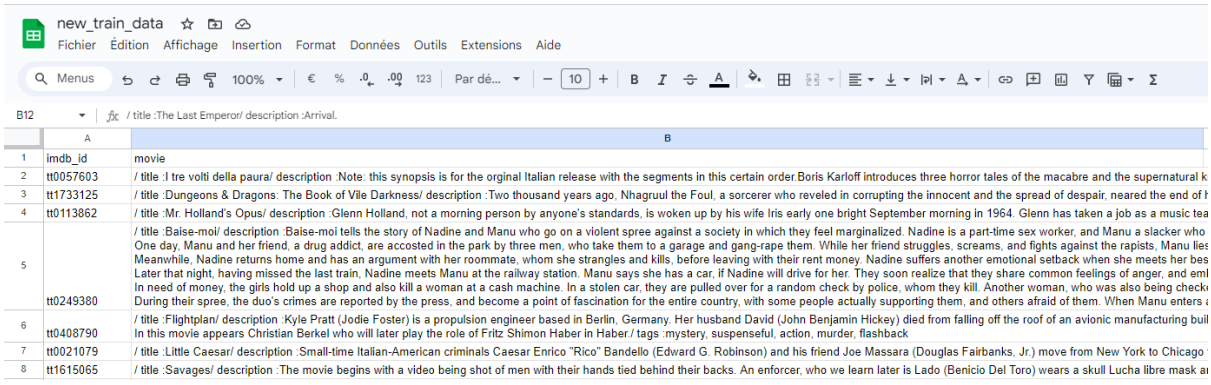

FIGURE 1 – Dataset d'entrainnement

2.3 Diagramme Fonctionnel

Voici le diagramme fonctionnel du chatbot, réalisé à l'aide de l'outil drawio :

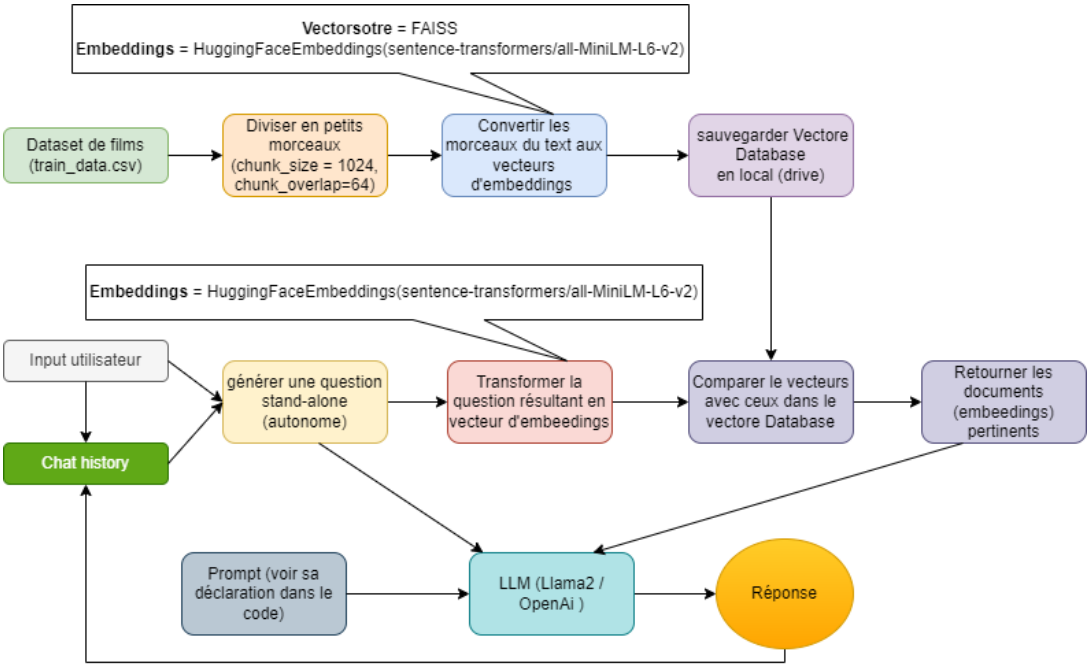

FIGURE 2 – Diagramme fonctionnel

2.4 Méthodologie de Travail

la première étape était du choisir un bon dataset, ensuite après la recommendation du professeur, j'ai consulté la documentation de la technique du **RAG** sur le site du LangChain (`https://python.langchain.com/docs/use_cases/question_answering/code_understanding`), puis j'ai appliqué cette technique sur mon dataset afin de créer un vectorestore d'embeedings, mais j'ai utilisé la librairie FAISS au lieu de CHROMA, enfin j'ai intégré ce contexte dans les modèles LLM choisis pour créer le chatbot en personalisant la prompt pour qu'elle répond selon le dataset et le contexte de la conversation, et évitant de répondre à n'import quelle question hors contexte du recommendation de films.

3 Approche Méthodologique

3.1 Première LLM

Au début, j'ai choisi de travailler avec la méthode CTransformers pour le LLM LLama2 :

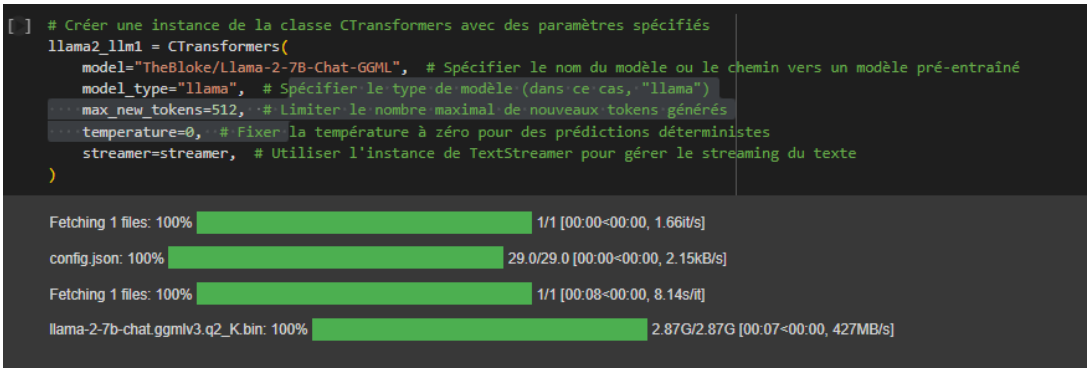

FIGURE 3 – llama2 avec CTransformers

Après pour le tester j'ai posé au chatbot une question hors contexte pour tester au même temps la fonctionalité de prompt et avoir le temps d'execution dans les meilleurs des cas, car dans ce cas le modèle va pas aller chercher dans le vectorstore des embeeding qui est volumineux, mais malgré ça le temps de réponse obtenu (131.56 seconds) est très grand commme vous voyer ci-dessous :

(a) Teste de llama2 avec CTransformers

(b) suite

FIGURE 4 – Teste de llama2 avec CTransformers

Pour cela j'ai optimisé le modèle llama2 de la manière suivante :

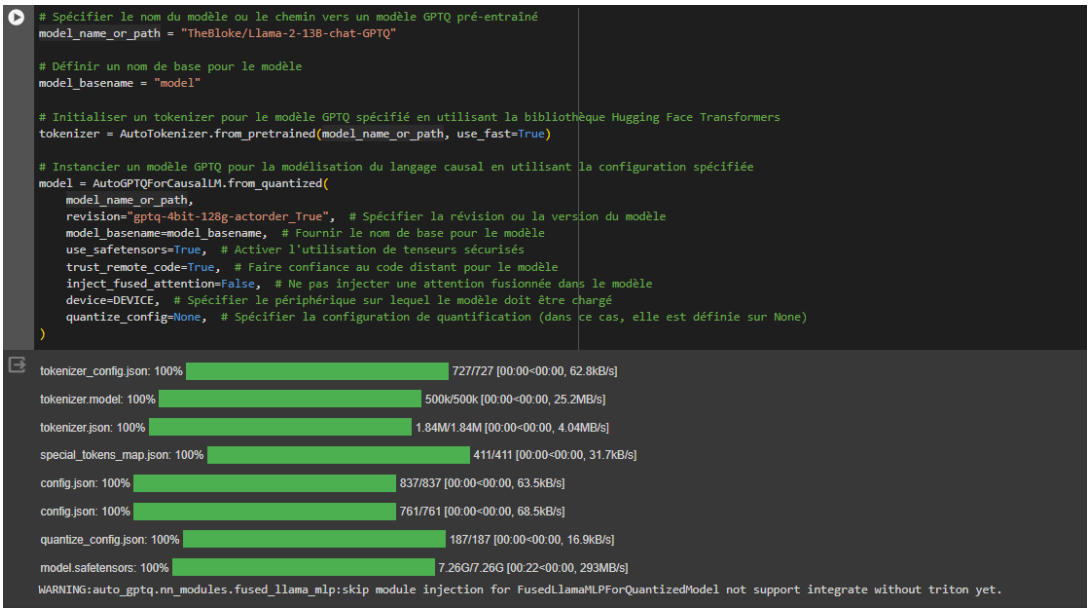

FIGURE 5 – llama2 quantisation de llama2 avec AutoGPTQForCausalLM

ensuite on crée le modèle avec HuggingFacePipeline :

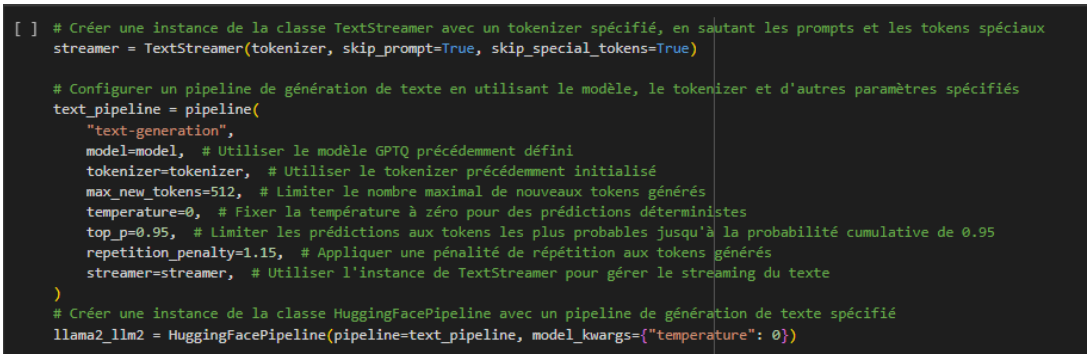

FIGURE 6 – Llama2 HuggingFacePipeline

Nous utilisons la bibliothèque Hugging Face et son module `transformers` pour charger le tokenizer et le modèle. Le modèle est instancié en tant que `AutoGPTQForCausalLM` à partir de la version quantifiée du modèle pré-entraîné. Les paramètres spécifiques incluent la révision du modèle, l'utilisation de tenseurs sécurisés (`use_safetensors`), la confiance dans le code distant (`trust_remote_code`), et d'autres paramètres tels que l'appareil (`device`) et la configuration de quantification (`quantize_config`).

- **revision :** La révision spécifique "gptq-4bit-128g-actorder_True" indique que le modèle est quantifié avec une précision de 4 bits, une taille de 128 Go, et un ordre d'activation activé. Cette configuration particulière peut être le résultat d'une optimisation pour l'efficacité de la mémoire et des performances spécifiques à la tâche.
- **use_safetensors :** L'activation de SafeTensors peut être cruciale pour la stabilité pendant l'entraînement. Si le modèle est sujet à des instabilités numériques, l'utilisation de SafeTensors peut aider à détecter et à traiter ces problèmes.
- **trust_remote_code :** Si le modèle doit charger du code distant, le paramètre 'trust_remote_code' est défini sur True pour indiquer que le code distant associé au modèle est considéré comme sûr.
- **inject_fused_attention :** Le choix de ne pas injecter une attention fusionnée peut être lié à des considérations de performance ou de préférences spécifiques au modèle. L'injection d'une attention fusionnée peut affecter la vitesse d'inférence et d'autres aspects du modèle.
- **device :** Le choix du dispositif (probablement spécifié comme "DEVICE") est crucial pour le déploiement. Il peut s'agir d'un GPU spécifique pour accélérer les calculs ou d'un autre dispositif en fonction des ressources disponibles.
- **quantize_config :** Le choix de laisser 'quantize_config' à None indique probablement que la quantification est gérée automatiquement ou que des paramètres par défaut sont utilisés.

En testans ce nouveau modèle dans le chatbot on obtien un temps de réponse (8.86 secondes) beaucoup plus mieux par rapport à la même question posé au premier modèle :

FIGURE 7 – Test de Llama2 quantisé avec HuggingFacePipeline

3.2 Configuration du Pipeline de Génération de Texte

Nous utilisons le module `pipeline` de la bibliothèque Hugging Face pour configurer le pipeline de génération de texte. Les paramètres incluent le modèle, le tokenizer, le nombre maximal de nouveaux tokens générés (`max_new_tokens`), la température (`temperature`), *top-p* (`top_p`), et la pénalité de répétition (`repetition_penalty`).

3.3 Configuration de la Mémoire de Conversation

Nous utilisons une mémoire de résumé de conversation (`ConversationSummaryMemory`) pour stocker les échanges précédents. Cette mémoire est liée au modèle de langage (`llm`) et permet de retourner les messages stockés.

```python
# Créer une instance de la classe ConversationSummaryMemory avec un autre modèle Llama (llama2_llm2),
# une clé de mémoire ("chat_history") et la spécification de retourner les messages
llama2_memory2 = ConversationSummaryMemory(
    llm=llama2_llm2,  # Utiliser l'autre modèle Llama spécifié (llama2_llm2)
    memory_key="chat_history",  # Utiliser la clé de mémoire "chat_history"
    return_messages=True  # Spécifier de retourner les messages
)
```

FIGURE 8 – Historique du chatbot

3.4 Chaîne de Récupération Conversationnelle

Enfin, nous utilisons une chaîne de récupération conversationnelle (`ConversationalRetrievalChain`) qui utilise le modèle de langage comme LLM (`llama2_llm1`) et un récupérateur de documents (`docsearch.as_retriever()`). Des paramètres spécifiques, tels que la chaîne de combinaison de documents (`combine_docs_chain_kwargs`) et la mémoire (`memory`), sont également configurés.

```python
# Créer une instance de la classe ConversationalRetrievalChain en utilisant un autre modèle Llama (llama2_llm2),
# un récupérateur spécifié (docsearch.as_retriever()), des paramètres pour la combinaison de documents (combine_docs_chain_kwargs),
# et une mémoire spécifiée (llama2_memory2)
llama2_qa2 = ConversationalRetrievalChain.from_llm(
    llama2_llm2,  # Utiliser l'autre modèle Llama spécifié (llama2_llm2)
    retriever=docsearch.as_retriever(search_kwargs={'k': 2}),  # retoruner les 2 meilleurs recommendations
    combine_docs_chain_kwargs={"prompt": prompt},  # Utiliser les paramètres pour la combinaison de documents
    memory=llama2_memory2  # Utiliser la mémoire spécifiée (llama2_memory2)
)
```

FIGURE 9 – Fonction de réponse

Cette configuration complète constitue l'approche adoptée pour définir le modèle de langage utilisé dans le chatbot, intégrant des éléments tels que la quantification, la génération de texte, la mémoire de conversation et la récupération conversationnelle.

3.5 Deuxième LLM

Le deuxième LLM utilisé pour la comparaison et GPT-TURBO-3.5 d'OPENAI, celui ci nécessite un clé api d'un compte openai pour pouvoir l'utilisé, mais après le test il montre qu'il réponds dans un délai (2.87) mieux que le premier LLM par rapport à la même question :

```python
# Définir la clé d'API OpenAI en tant que variable d'environnement
os.environ['OPENAI_API_KEY'] = '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'

# Créer une instance de la classe ChatOpenAI avec des paramètres spécifiés
gpt_llm = ChatOpenAI(
    model_name="gpt-3.5-turbo",  # Spécifier le nom du modèle (dans ce cas, "gpt-3.5-turbo")
    temperature=0,  # Fixer la température à zéro pour des prédictions déterministes
    max_tokens=512,  # Limiter le nombre maximal de tokens générés
    top_p=1,  # Limiter les prédictions aux tokens les plus probables jusqu'à la probabilité cumulative de 1
    frequency_penalty=0  # Appliquer une pénalité de fréquence de token de zéro
)
```

FIGURE 10 – LLM : GPTTURBO3.5

```
Question 1 : Hi, what is the capital fo Morocco

Réponse 1 : I'm sorry, but I'm specialized in providing movie recommendations and don't have information about the capital of Morocco. Is there anything else I can assist you with regarding movies?
(Temps de réponse : 2.87 secondes)


Question 2 : stop
Sortie du programme
```

FIGURE 11 – Test de GPT-TURBO-3.5

Remarque *: Pour ne pas avoir d'interupption dans le cas du GPT-TURBO-3.5 il faut attendre au moins 20 secondes avant de poser une nouvelle question au chatbot car c'est la limite posé par OPENAI.*

4 Test des recommendations du ChatBot pour LLama 2 optimisé et GPT-TURBO-3.5

4.1 LLAMA 2

FIGURE 12 – CHATBOT (llama 2)

4.2 GPT-TURBO-3.5

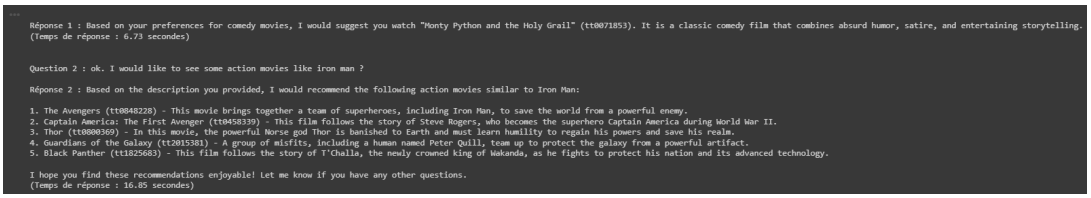

FIGURE 13 – CHATBOT (GPT)

5 Limitations et Perspectives d'Améliorations

D'après les résultats obtenue, on constate que mon travail peut être amélioré dans 2 sens : premièrement la rapidité du temps d'exécution, car le chat prend quelques secondes avant de répondre mais je pense que cela est dû essentiellement à la taille du vectorstore à cause du dataset volumineux (46MB), ainsi à l'environnement d'exécution Colab, qui utilise le GPU gratuit qui n'est pas puissant. Deuxièment dans l'interface du chatbot, en utilisant par exemple streamlit, mais ici en fait j'ai pensé à le faire mais après avoir testé l'exécution du script sur mon local, il était très lent vu qu'il n'utilise pas le gpu afin de répondre à la question et donc ça prend beaucoup plus du temps à répondre à l'utilisateur, pour cela j'ai décidé de travailler sur Colab qui ne support pas l'utilisation du streamlit chat.

6 Contribution des Membres du Groupe

Dans ce projet je suis seul dans le groupe, donc il s'agit d'un travail individuel dont j'ai réalisé toutes les étapes et le travail du projet.

7 Conclusion

Ce project m'a permet de découvrir certains LLMs open source les plus populaire dans le domaine NLP et aussi de savoir que les LLMs en géneral ne font pas le même types des tâches, tel que la génération du texte dans mon cas et d'autres llm comme BERT pour la classification du texte etc. Et aussi que les résultats obtenus par un LLm dépends des valeurs des paramètres qu'on lui attribue comme la température et la personalisaton de la prompt.