Agent conversationnel et chatbot

Utilisation de Large Language Models pour publier une base de connaissances
Partie 5

Nicolas Debeissat
nicolas.debeissat@mail-formateur.net

Plan du module

Pourquoi faire un agent conversationnel ?

Comment défendre votre projet en entreprise ?

Principes de base des LLMs

Ce qui va faire qu'il va donner des bons résultats, ou non

Entraînement - Fine Tuning

Comment adapter ses réponses

Retrieval Augmented Generation

Répondre à partir de vos données

Intégration dans une application

Créer une API à partir de votre modèle

Evaluation

Entraîner votre IA à faire votre évaluation

Intégration dans une application

Créer une API à partir de votre modèle

TD5-1 : Création API FastAPI

Créer un répertoire td-fast-api-deploiement
poetry init
poetry config virtualenvs.in-project true --local
Créer un fichier README.md, un répertoire app avec à l'intérieur des fichiers main.py et __init__.py
Exécuter poetry install
source .venv/Scripts/activate
poetry add fastapi **uvicorn** pymupdf4llm llama-index python-multipart llama-index-embeddings-huggingface

TD5-2 : Démarrage en debug

Dans main.py :

```
from fastapi import FastAPI
app = FastAPI()
@app.get("/")
def read_root():
  return {"message": "Welcome to the FastAPI application!"}
```

Exécuter le mode debug fastapi (Show Automatic Python configurations) avec un point d’arrêt dans la méthode read_root()
Ouvrir http://localhost:8000

TD5-3 : Les imports

```
import json
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import StreamingResponse
from pymupdf4llm import LlamaMarkdownReader
from llama_index.core import VectorStoreIndex
from llama_index.core.retrievers import VectorIndexRetriever
from llama_index.core import Settings
from llama_index.embeddings.huggingface import
HuggingFaceEmbedding
import shutil
import os
import requests
```

TD5-3 : Indexation du pdf

```
Settings.embed_model = HuggingFaceEmbedding(
    model_name="OrdalieTech/Solon-embeddings-base-0.1"
)

@app.post("/ask_pdf")
async def ask_pdf(query: str = "Pourquoi faire un Chatbot ?", file: UploadFile = File(...)):
    file_location = save_file_to_temp(file)
    retriever = index_pdf(file_location)
    list_nodes = retriever.retrieve(query)
    rag_doc = "\n\n".join([node.text for node in list_nodes])
    return rag_doc

def index_pdf(file_location):
    reader = LlamaMarkdownReader()
    markdown_content = reader.load_data(file_location)
    index = VectorStoreIndex.from_documents(markdown_content)
    retriever = VectorIndexRetriever(
        index=index,
        similarity_top_k=2,
    )
    return retriever

def save_file_to_temp(file):
    file_location = f"temp/{file.filename}"
    os.makedirs(os.path.dirname(file_location), exist_ok=True)
    with open(file_location, "wb") as buffer:
        shutil.copyfileobj(file.file, buffer)
    return file_location
```

TD5-4 : Stream de l’appel Ollama

```
@app.post("/ask_pdf")
async def ask_pdf(query: str = "Pourquoi faire un Chatbot ?", file: UploadFile = File(...)):
  file_location = save_file_to_temp(file)
  retriever = index_pdf(file_location)
  list_nodes = retriever.retrieve(query)
  rag_doc = "\n\n".join([node.text for node in list_nodes])
  call_ollama = call_ollama_stream(f"From that documentation: \n{rag_doc}\nAnswer that query {query}",
"mistral")
  return StreamingResponse(call_ollama, media_type="text/event-stream")

async def call_ollama_stream(prompt, model):
  with requests.post(
      url=f"http://localhost:11434/api/generate/",
      headers={"Content-Type": "application/json"},
      json={"prompt": prompt, "model": model},
      stream=True,
  ) as response:
    response.raise_for_status()
    for content in response.iter_lines(decode_unicode=True):
      if len(content):
        bytes_to_str = content.decode()
        message = json.loads(bytes_to_str)
        yield message["response"]
```