# GENAI to assist in mental health

In this part we will create a large language model(LLM) QnA bot for medical issue. Since training an LLM is a cumbersome task which requires huge time and compute resource, we will use a technique called retrieval augmented generation. This concept is getting very popular nowadays because using this we don’t need to train or finetune model, but like the name suggests we just pass the reference data which the pretrained LLM model uses as reference and gives answers to the user’s queries.

We use the [PUBMED](https://pubmed.ncbi.nlm.nih.gov/) articles for the reference text input to our model. First we load the articles and then parse them to create a proper structured data including the metadata. For this task we make use of the langchain library. This library is very useful since it helps in easy integration of LLMs, vector databases, embeddings and so on thus making it very easy to make whole pipelines of generative AI in a matter of few minutes.

We begin by splitting the text into token chunks and then use the popular E5 embedding model from huggingface to generate the text embeddings. Text embeddings is nothing but a vector representation of the text which is very popular in NLP and can be used to treat text as numbers thus enabling us the train the models on it and generate statistics.

Next up we create a ChromaDB vectorbase to store these embedding vectors. For this task we are going to use the Falcon LLM model from huggingface. After loading the Falcon model we create our LLM prompt. Prompt is just an initial set of instruction which is used the tell our LLM model how to process the user query, how to analyse it and in what format and how to give the answer required. Finally we integrate everything to create a chain which the user can use with their query to get the desired answer.

## CODE:

## Installation

Pip install langchain chromadb jq tiktoken sentence-transformers biopython accelerate

## Imports

from langchain.document\_loaders import JSONLoader

from langchain.text\_splitter import TokenTextSplitter, CharacterTextSplitter

from langchain.embeddings import HuggingFaceEmbeddings

from langchain\_community.vectorstores import Chroma

import os

from huggingface\_hub import notebook\_login

notebook\_login()

from transformers import AutoTokenizer, AutoModelForCausalLM

from transformers import AutoTokenizer, pipeline

from langchain import HuggingFacePipeline

from langchain.chains import RetrievalQA

from langchain.prompts import PromptTemplate

import time

## Data Loading

# Define the metadata extraction function.

def metadata\_func(record: dict, metadata: dict) -> dict:

metadata["year"] = record.get("pub\_date").get('year')

metadata["month"] = record.get("pub\_date").get('month')

metadata["day"] = record.get("pub\_date").get('day')

metadata["title"] = record.get("article\_title")

return metadata

loader = JSONLoader(

file\_path='./pubmed.json',

jq\_schema='.[]',

content\_key='article\_abstract',

metadata\_func=metadata\_func)

data = loader.load()

## Data processing

text\_splitter = TokenTextSplitter(chunk\_size=128, chunk\_overlap=50)

chunks = text\_splitter.split\_documents(data)

## Embeddings and VectorDB

modelPath = "intfloat/e5-large-unsupervised"

embeddings = HuggingFaceEmbeddings(

model\_name = modelPath,

model\_kwargs = {'device':'cuda'},

encode\_kwargs={'normalize\_embeddings':False})

db = Chroma.from\_documents(chunks, embeddings)

## LLM Model Loading

model\_id = "Rocketknight1/falcon-rw-1b"

tokenizer = AutoTokenizer.from\_pretrained(model\_id)

model = AutoModelForCausalLM.from\_pretrained(model\_id, device\_map='auto')

pipe = pipeline("text-generation", model=model, tokenizer=tokenizer, max\_new\_tokens=128)

llm = HuggingFacePipeline(

pipeline = pipe,

model\_kwargs={"temperature": 0.5, "max\_length": 512}

)

## Prompt Generation

PROMPT\_TEMPLATE = """Answer the question based only on the following context:

{context}

You are allowed to rephrase the answer based on the context.

Question: {question}

"""

PROMPT = PromptTemplate.from\_template(PROMPT\_TEMPLATE)

## Chain Building and prediction

qa\_chain = RetrievalQA.from\_chain\_type(

llm,

retriever=db.as\_retriever(k=2),

chain\_type\_kwargs={"prompt": PROMPT},

return\_source\_documents=True

)

query = "What are the most common mental health issues?"

result = qa\_chain({"query": query})

print(result['result'].strip())