“Talk To CSV using LLM”

Making a simple Chatbot to interact with CSV files using Llama 2 LLM and Langchain along with FAISS.

by Dhenuvakonda Sai Naveen

This is part of my internship carried out at Experion Technologies where I was tasked with doing a deep dive on Generative AI, LLMs and related technologies and further given a simple use case involving enhancing analytics of data with LLMs. This is a detailed summary of my work done as part of the said so internship.

1. Introduction

Overview

Developing a beginner-level project for conducting analytics on a CSV file using natural language is indeed a challenging endeavour. Several constraints have been encountered during this project's development, which has led to certain limitations. Firstly, the availability of computational resources has been a constraint, as advanced Language Model Models (LLMs) often demand substantial RAM capacity for optimal performance. Secondly, premium LLMs that offer superior quality and capabilities typically require paid subscriptions or API keys, which were not available for this project. As a result, open-source quantized models were employed to ensure the project could run locally on the available system.

In its current state, the project primarily focuses on generating embeddings for the data within a CSV file and responding to basic prompts. However, there is significant room for improvement to enhance its functionality and capabilities. As a beginner in the realm of LLMs, I experienced a very steep learning curve. The project's potential for growth and refinement is well overdue, and future improvements will be undertaken.

Deep Dive into LLMs

Generative AI (Gen AI):

Gen AI is a type of AI that can create text, audio, and even videos. It's like a smart computer that understands patterns in information. We teach it by showing it lots of examples so it can understand how things relate to each other. When you ask it for something, it can give you an answer based on what it learned from all those examples. To do this, it breaks down what you ask into smaller pieces, kind of like breaking a sentence into words. Then, it gives each piece a number to help it understand the order and meaning of the words.

Early Gen AI Models:

Before Gen AI got really good, there were some AI models like Variational Autoencoders and Generative Adversarial Networks (GANs). They could create things, but they weren't great at understanding language and grammar.

Large Language Models (LLMs):

Large Language Models like GPT-3 are designed to create and work with text. They learn by looking at a huge amount of text and figuring out how language works. These models are called "large" because they have a ton of settings, like having billions of tiny switches. They mostly use something called a "Transformer."

Transformers:

Transformers are a special type of computer program, kind of like a super-smart robot. They are good at understanding how words in a sentence are connected to each other. They first learn the basics of language and then get better at specific things through more training. Big models like GPT and Bard use this type of robot-like program.

Why Transformers Are Better:

Transformers are good at understanding words in a sentence, no matter where the words are. They're like super-efficient robots that can look at all the words together. They can see the important parts and how they fit together, even if the words are far apart. This is better than older methods like RNNs that had trouble with long sentences. That's why transformers are used in models like GPT-3.

CHATGPT (GPT-3):

GPT-3 is a special AI model created by OpenAI. It uses a method that's good at understanding words and relationships between words. It has layers that help it calculate which words are important in a sentence and how they go together. It's like having a team of helpers who work together to figure out the best answer. They also use a smart trick to understand far-apart words in a sentence. GPT-3 is kind of like a group of six friends, each with a job to figure out what you're asking.

Difference Between GPT-3 and GPT-4:

GPT-4 is like a newer, smarter version of GPT-3. It learned from a lot more information. It's better at working with longer sentences and text. However, it still can't make pictures.

Bard (Google):

Bard is made by Google, and it's good at pulling in information from different sources. It learns step by step, getting better as it goes along. It's like a student who studies from different books and combines what it learns in a smart way. Bard is also good at understanding math.

Comparing GPT-4 and Bard:

GPT-4 has learned from even more information than Bard. It can understand longer text and more types of information. Bard is a bit like a student who studies from different books, while GPT-4 is like someone who read a whole library of books. But they each have their own strengths and ways of learning.

Why use Vector Embeddings and what's FAISS

1. Vector Databases, Search, and Embeddings:

Vector databases are like smart systems that store and retrieve data in a special way. They keep information in a format called "vectors," which are just numbers that represent the data. This helps to quickly find the information you need, especially when dealing with a lot of data. It's used in things like recommendations, search engines, and content sorting.

2. Semantic Search:

Semantic search is a clever way of looking for information. Instead of just matching words, it understands the meaning of what you're asking for. It looks at the words you use and what you're looking for to give you better results.

3. RAG - Retrieval-Augmented Generation:

RAG is a method that combines finding information and creating answers. First, it looks for the right information and then uses it to make a response. It's like having a helper that finds the right books and then writes an answer for you. It's used in things like answering questions and generating content.

4. What is Vector Search:

Vector search is a way to find things in a big pile of information that are similar to what you're looking for. It's like finding items that are very much like what you want. This is used in recommendations, searching, and sorting information.

5. Filtering:

Filtering is like telling a system to only show you certain things. For example, if you want to see only Nike shoes, you tell the system to only show you those. It's used to make search results better.

6. ANN and Filtering:

ANN is a way to help filter and search for things more quickly. It's like having a helper that first looks for similar things and then narrows down what you want. This makes finding things faster and more accurate.

7. Vector Stores - Databases, Libraries, and Plugins:

Vector stores are systems that manage and organize the special data we talked about earlier. They help with basic things like adding, finding, changing, or removing data. Different tools like libraries and plugins make it easier to use these systems to store and find data.

8. Best Practices to Follow:

- Think about whether you really need a special system like a vector database for your project. For simple tasks, you might not need it, but for complex things like answering questions, it can be very useful.

- Choose the right way to organize and represent your data. You can keep it in one piece or split it into parts, depending on what works best for your project.

- Make a plan for what to do if the system takes too long to find what you want. Think about things like checking for harmful content and setting a time limit to keep things running smoothly.

What is FAISS

FAISS (Facebook AI Similarity Search) is a library that allows developers to quickly search for embeddings of multimedia documents that are similar to each other. It solves limitations of traditional query search engines that are optimized for hash-based searches, and provides more scalable similarity search functions. With FAISS, developers can search multimedia documents in ways that are inefficient or impossible with standard database engines (SQL). It includes nearest-neighbour search implementations for million-to-billion-scale datasets that optimize the memory-speed-accuracy trade-off. FAISS aims to offer state-of-the-art performance for all operating points.

FAISS contains algorithms that search in sets of vectors of any size, and also contains supporting code for evaluation and parameter tuning. Some if its most useful algorithms are implemented on the GPU. FAISS is implemented in C++, with an optional Python interface and GPU support via CUDA.

Some examples of searches are cosine similarity search or finding eucledian distance between the vector points. Vector libraries only store vector embeddings, not the associated objects, and index data is immutable.

What does Langchain do?

Understanding Prompt Templates and Chaining:

Prompt Templates are standardized prompts which have been evaluated and judged to give the best possible response, instead of making a prompt on the fly, having a prompt template can allow it to be part of a workflow and input data can be embedded into the prompt itself. This has given birth to a new line of work and research namely Prompt Engineering

Smartly Combining LLMs and Vector Databases:

To supercharge our tasks, we use both smart programs (LLMs) and data storage tools (vector databases). We employ LangChain to create complex logic flows, and LLM Agents are our guides in making these programs understand each other. Whether it's a single interaction with a smart program (a "task") or multiple interactions forming a series of tasks (a "workflow"), they make these processes smarter.

Crafting Effective Prompts:

The art of prompt engineering involves crafting well-written questions to get the best answers from smart programs. For example, imagine you want to understand how people feel about a topic based on various articles. It is an approach where you first summarize the articles using a summarizing smart program and then pass these summaries to another smart program to analyse sentiment. This kind of "chaining" helps us get more valuable insights.

Building LLM Chains:

LLM Chains are like a string of connected tools and smart programs. They are linked together effectively. For instance, one smart program's output can become the input for another. These chains can not only connect to other smart programs but also to different tools and even the internet itself. It's a versatile way to work on tasks step by step.

Empowering LLM Agents:

LLM Agents act as coordinators, delegating tasks to the right tools and programs. They operate based on a "Reason-Action Loop" that involves observing, thinking, and taking action. This loop helps create a clear plan for solving a problem. The task itself is done using the tools, while the LLM Agents bring everything together and provide a step-by-step plan when a prompt is given. They're like the conductors of a symphony of tasks.

Finetuning a LLM

1. Introduction to Fine-Tuning: Fine-tuning refers to the process of taking a pre-trained model and adjusting it to perform better on a specific task. For example, new generation LLMs are released in multiple sizes and sequence lengths, and each one is fine-tuned to examples like Chat, Base, Instruct, etc. The base mode is mainly used to predict just the next word, while smaller and bigger ones are fine-tuned for various purposes.

2.Applying Foundation LLMs: Foundation LLMs can be applied to various tasks. For example, if your task is to convert news articles into riddles for users to read, you could use a zero-shot, one-shot, few-shot approach, or your own method.

3. Few-Shot Learning: Few-shot learning involves training a model on a small number of examples and then using it to make predictions on new, unseen data. This approach can be combined with APIs to produce results. For instance, you could use a news API and a few examples to get results from a model. Another approach is using a pre-fine-tuned instruction-following model, where the model is trained to follow instructions without needing few-shot learning

4. DIY Fine-Tuning LLM: If you want to fine-tune an LLM on your own, you can either build your own model and then fine-tune it or fine-tune an existing model. While fine-tuning an existing model, task tailoring can make inference cost cheaper and more specific, and you can have control over the data being used. However, you need data and skills to choose the right parameters for fine-tuning. The fine-tuning process involves training the model on high quality data.

5. Evaluating LLMs: Evaluating LLMs can be challenging due to the lack of a consistent definition for evaluation. Generic loss functions often don't suffice. Metrics like Perplexity and Accuracy are commonly used. Each NLP task will have its own metrics, like BLEU for Translation and ROUGE for summarization. Datasets like the Stanford QA Dataset are commonly used for benchmarks.

6.DeepSpeed for Training Speed Boost: DeepSpeed is a performance optimization library from Microsoft that significantly reduces the resources needed for model training. It implements several memory and compute-efficient practices such as model parallelism, pipeline parallelism, and ZeRO (Zero Redundancy Optimizer) to allow for the training of models with billions of parameters. It can be used to fine-tune models more rapidly)

Problems with Inference

Large Language Models (LLMs) have become increasingly influential in various sectors, from customer service to creative writing. However, the use of LLMs is not without risks and limitations.

Risks and Limitations of LLMs: LLMs have the potential to disrupt the job market by replacing roles such as customer service representatives and content creators. Training these models also incurs financial and environmental costs due to the energy-intensive nature of machine learning. Moreover, LLMs are only as good as the data they are trained on. For instance, if the data is biased towards a certain demographic, the model will reflect this bias in its predictions. This is particularly relevant in the case of GPT-2, which was largely trained on Reddit data, a platform known to have a predominantly male user base. This could lead to gender bias in the model's outputs. Furthermore, LLMs can also generate potentially harmful content, such as misinformation or discriminatory language. There are also privacy concerns, as these models could potentially leak sensitive information.

Hallucination in LLMs: Hallucination refers to the phenomenon where an LLM generates content that is nonsensical or unfaithful to the provided source content. There are two types of hallucinations: intrinsic and extrinsic. Intrinsic hallucinations directly contradict the source material, introducing factual inaccuracies or logical inconsistencies. Extrinsic hallucinations do not contradict but also cannot be verified against the source, adding elements that could be considered speculative or unconfirmable. The main causes of hallucination include biased training data and the limitations of the model's architecture. Evaluating hallucination is challenging, but some commonly used metrics include BLEU, ROUGE, and PARENT.

Mitigation Strategies: To mitigate the risks associated with LLMs, it is important to build a faithful dataset that accurately represents the problem space. This can be powered by human interactions to remove irrelevant data. More research and experimentation in model architecture can also help improve the model's performance and reduce hallucination. Advanced prompt engineering methods, such as self-consistency, Reason and Act (React), Chain-of-Verification (COVE), and Guidance, can also be used to control the model's output. These methods involve generating multiple responses from the model and checking their consistency to assess the accuracy of the output. Reinforcement Learning from Human Feedback (RLHF) can also be applied during the later stages of training to optimize for more accurate and grounded outputs.

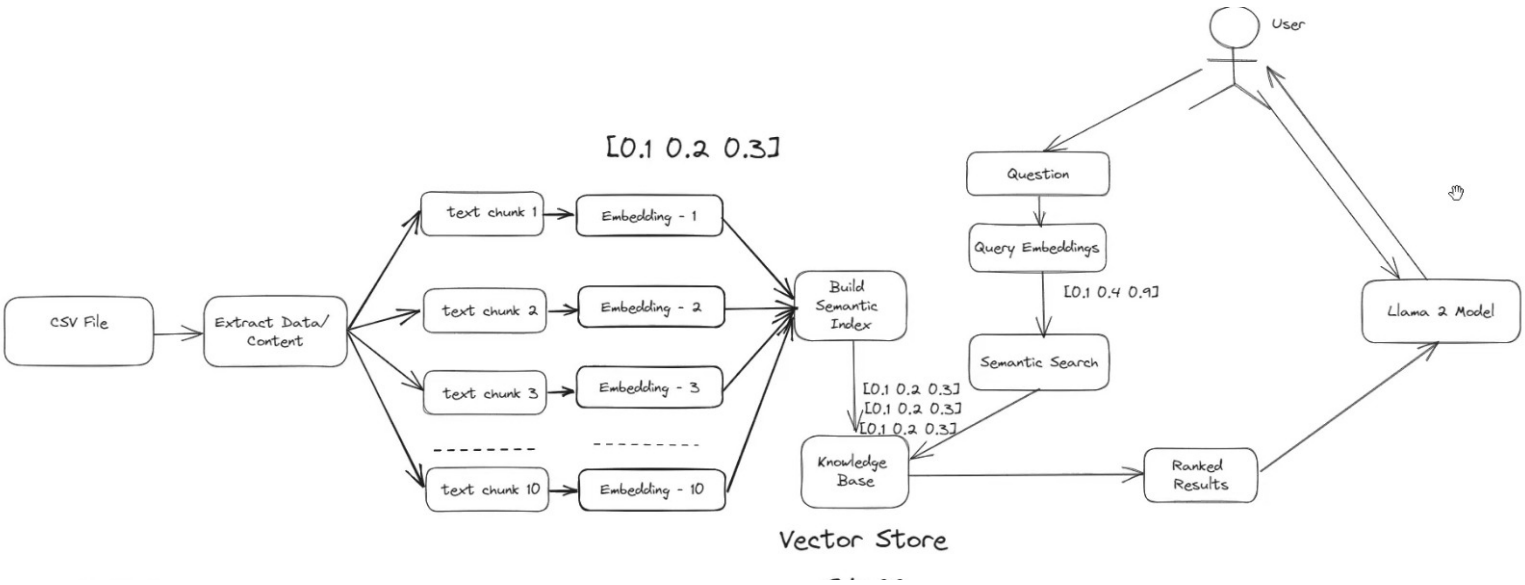
2.Existing Work

There are quite a lot of tools and plugins which pretty much do the job quite well. PandasAI is one hot trending library that has revolutionized analytics. OpenAI has released plugins which can do inference from pdfs, CSVs and even images. Big corporations are investing heavily into Generative AI and there are flourishing startups with amazing AI tools most of which do have a feature to draw insights from Datasets. There are several python plugins which can be infused with Excel or Google sheets.

This work was done as part of a learning project and is not intended to replace any existing wonderful tools. This task has helped me understand how to get started with LLMs and build very cool tools and software to make things easier for the user and clients.

3.Workflow

Chosen Workflow



The project involves processing a CSV file to create vector embeddings, enabling users to extract relevant information through natural language queries. To handle potentially large datasets, the system first breaks down the data into smaller chunks, each containing approximately 500 characters. Vector embeddings, representing compressed versions of these text chunks, are then generated. These embeddings are indexed and stored in a local vector database using FAISS (Facebook AI Similarity Search), forming the project's knowledge base.

User interaction is facilitated through natural language prompts, questions, or instructions. These inputs are converted into embeddings for comparison with the knowledge base. Semantic search techniques, such as cosine similarity and distance calculations, are employed to find the most relevant answers from the knowledge base. The results are ranked, typically presenting the top 5 responses to the user. Finally, these top-ranked results, along with the user's query, are sent to the LLaMa2 model which I downloaded for further processing, leading to the generation of natural language responses as the final outcome.

Alternate Workflow

There were many other ways this project could have been done

1. Use OpenAI API and draw results from GPT 3.5/4 itself. The results would have been way more accurate and it has much more potential to be a good tool.
2. Use Bard or other LLM-as-a-service APIs.
3. Use much more advanced LLMs which are open source like Minstral 70B or falcon 70B. Claude as well, but I personally did not have enough computational resources to use them.
4. Fine tuning model on a custom dataset with entries related to basic analytics tasks. This firstly needed a dataset to be made and vetted by humans and fine tuning is no easy task and is more or less equal to pre training a model when it comes to expense of resources like RAM and VRAM.
5. Use Cloud based notebooks and use higher level LLMs. The problem of resources can be overcome but again, costs are there for getting premium version of Colab Notebooks or Kaggle Notebooks or Databricks notebooks as well.
6. Different workflow where python code Is generated by the LLM and it is applied onto the dataset. This is something viable but as a beginner, it is hard to implement all this in a short interval of time.

4.Code

This script uses the LangChain library to create a conversational agent that can respond to user queries using a large language model. It follows these steps:

➔ **Before running the code, please download Llama2 quantized model from HuggingFace and the dataset from Kaggle**

1. Load the Data: It uses the CSVLoader class to load a dataset from a CSV file.
2. Split the Text: It uses the RecursiveCharacterTextSplitter to split the data into smaller chunks.
3. Generate Embeddings: It uses the HuggingFaceEmbeddings to generate vector embeddings from the text chunks.
4. Store Embeddings: It uses the FAISS class to store the generated embeddings in a FAISS vector store.
5. Load the Model: It loads a CTransformers model, a type of large language model.

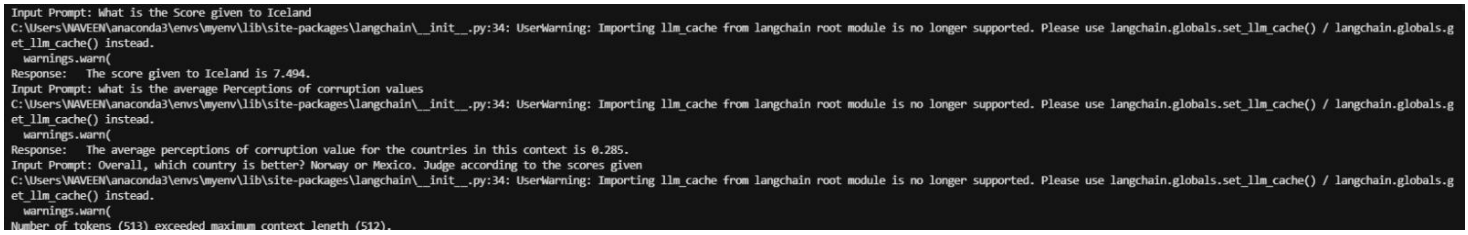
6. Create a Conversational Agent: It uses the ConversationalRetrievalChain class to create a conversational agent that can retrieve responses to user queries.
7. Start a Conversation Loop: It starts a conversation loop where the user can input a query and receive a response from the conversational agent.

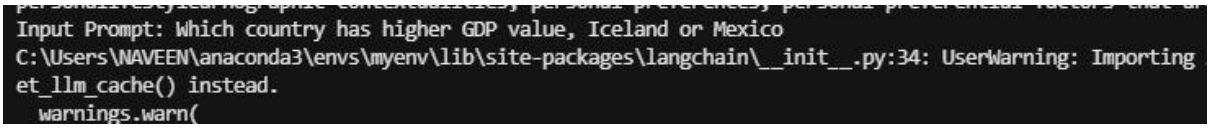
Code is given below:

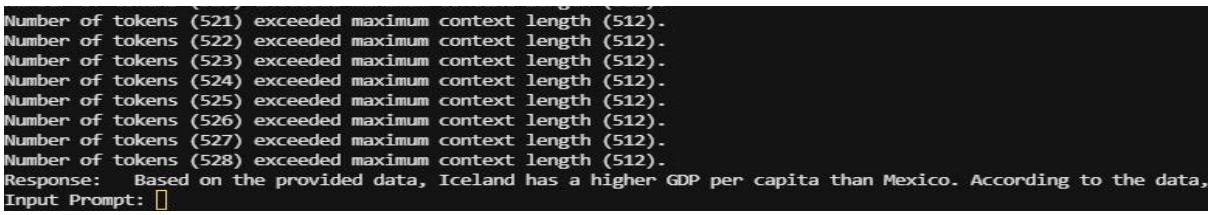
```
# Import necessary libraries from LangChain
from langchain.document_loaders.csv_loader import CSVLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain.llms import CTransformers
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationalRetrievalChain
import sys

# Set the path for the FAISS vector store
DB_FAISS_PATH = "vectorstore/db_faiss"

# Load the dataset using CSVLoader
loader = CSVLoader(file_path="data/2019.csv", encoding="utf-8",
csv_args={'delimiter': ','})
data = loader.load()  # This loads the data from the CSV file
print(data)

# Split the text into chunks using RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=20)
text_chunks = text_splitter.split_documents(data)  # This splits the data into
chunks
print(len(text_chunks))

# Download Sentence Transformers Embedding From Hugging Face
embeddings = HuggingFaceEmbeddings(model_name = 'sentence-transformers/all-
MiniLM-L6-v2')

# Convert the text chunks into embeddings and save the embeddings into FAISS
Knowledge Base
docsearch = FAISS.from_documents(text_chunks, embeddings)
docsearch.save_local(DB_FAISS_PATH)  # This saves the embeddings locally

# Load the CTransformers model
llm = CTransformers(model="llama-2-7b-chat.ggmlv3.q8_0.bin",
                    model_type="llama",
                    max_new_tokens=1024,
                    temperature=0.1)
```

```python
# Use the ConversationalRetrievalChain for conversation retrieval
qa = ConversationalRetrievalChain.from_llm(llm,
retriever=docsearch.as_retriever())

# Start a conversation loop
while True:
    chat_history = []
    query = input(f"Input Prompt: ")
    if query == 'exit':
        print('Exiting')
        sys.exit()
    if query == '':
        continue
    result = qa({"question":query, "chat_history":chat_history})
    print("Response: ", result['answer'])
```

Results







Future Improvements

1. Getting a better dataset would be good as this one is quite old and most of it is very small numerical values.

2. Using a proper vector database like Pinecone or ChromaDB which can have dynamic embeddings so that they can keep getting updated and even the quality of embeddings will be better.

3. Doing some sort of pre processing to improve the quality of the embeddings

4. Using a better LLM or using a LLM-as-a-service. Discusses in Alternate Workflow..

5. Fine tuning the base LLM on instruction tasks and maybe even on code generation which requires generating custom dataset.

6. Proper evaluation metrics like BLEU but for this task and better hyperparameters.

7. Further scope is definitely there, I will work on it in the future.