

CTSE Assignment 2

RAG-based PDF question-answering Chatbot

Gunathilaka L. P. N.
IT21286414

Contents

CTSE Assignment 2	1
RAG-based PDF question-answering ChatBot	1
1. Introduction	3
2. Justification of LLM Choice	4
3. Justification of Development Approach	6
4. System Design	8
5. Challenges and Lessons Learned.....	9
6. Use of GenAI Tools (Prompts and Outputs)	10
7. Conclusion	14
8. References.....	15

1. Introduction

The goal of this system is to provide an efficient and user-friendly way to extract information from lecture notes stored in PDF format and answer questions based solely on their content. This is achieved using large language models, embedding techniques, and a question-answering pipeline that integrates various machine learning and natural language processing (NLP) tools. The system uses Retrieval-augmented generation (RAG) techniques for enhancing the accuracy and reliability for the generated answers

The system leverages the power of Groq's LLaMA-4 model for answering questions based on the content of lecture notes, stored in PDFs. The content from the PDFs is first loaded, split into manageable chunks, and then embedded into a vector store to facilitate efficient and accurate retrieval. When a user asks a question, the system uses a search function to find the most relevant content and processes the query with the LLaMA-4 model. The system returns an answer derived from the lecture notes, ensuring that responses are relevant and aligned with the content provided.

This architecture was chosen due to the importance of creating an interactive, scalable, and contextually aware solution for answering academic-related questions. The system not only answers questions based on predefined lecture notes but also demonstrates an efficient use of modern NLP and machine learning techniques to enable intelligent document retrieval.

2. Justification of LLM Choice

The system utilizes Groq's LLaMA-4 model (llama-4-scout-17b-16e-instruct), a state-of-the-art language model developed by Meta, specifically designed for instruction-based tasks. The reasons for selecting Groq's LLaMA-4 model are as follows

1. High-Performance Language Model

LLaMA-4-scout-17b-16e-instruct is based on one of the most powerful large language models (LLMs) currently available. It is capable of understanding and processing a vast amount of context, which makes it particularly useful for question answering. LLaMA-4 was chosen because of its proven ability to process long-form content and understand complex questions.

The llama-4-scout-17b-16e-instruct contains 17B active parameters, 109B total parameters, 16 experts and 10M context length [1].

Category Benchmark	Llama 4 Scout	Llama 3.3 70B	Llama 3.1 405B	Gemma 3 27B	Mistral 3.1 24B	Gemini 2.0 Flash-Lite
Image Reasoning MMMU	69.4	No multimodal support	No multimodal support	64.9	62.8	68.0
MathVista	70.7			67.6	68.9	57.6
Image Understanding ChartQA	88.8			76.3	86.2	73.0
DocVQA (test)	94.4			90.4	94.1	91.2
Coding LiveCodeBench (10/01/2024-02/01/2025)	32.8	33.3	27.7	29.7	—	28.9
Reasoning & Knowledge MMLU Pro	74.3	68.9	73.4	67.5	66.8	71.6
GPQA Diamond	57.2	50.5	49.0	42.4	46.0	51.5
Long Context MTOB (half book) eng → kgv/kgv → eng	42.2/36.6	Context window is 128K	Context window is 128K	Context window is 128K	Context window is 128K	42.3/35.1 ⁵
MTOB (full book) eng → kgv/kgv → eng	39.7/36.3					35.1/30.0 ⁵

Figure 1: Performance benchmark [1]

2. Specialization in Instruction Following

LLaMA-4 has been fine-tuned to follow instructions, which makes it particularly effective for tasks that require understanding specific queries and responding in a structured format. This is crucial when answering questions based on a fixed set of documents, ensuring that the model does not generate out-of-context responses.

3. API Integration and Efficiency

Groq's API allows seamless integration of the LLaMA-4 model into the system. This makes it easy to invoke the model remotely, enabling real-time query processing without the need for intensive local compute resources. This scalability is essential for deployment in a production environment where users can interact with the system concurrently.

4. Flexibility for Customization

Groq's model also provides flexibility for customization, enabling the fine-tuning of behavior, such as modifying temperature, token limits, and response formatting. This adaptability allows the model to be tailored to the specific needs of the system, ensuring optimal performance for academic query answering.

5. Optimized for Handling Large Inputs

LLaMA-4 is optimized to handle large inputs, which is critical when the system processes large PDF files. It has a high capacity for context and can generate relevant responses even when dealing with lengthy documents or complex information structures.

3. Justification of Development Approach

The development approach for this system involves integrating various technologies and components to facilitate the seamless flow of data, from loading documents to responding to queries. Below is the justification for each technology used

1. Langchain

Langchain is an open-source framework for building applications powered by large language models. It was chosen for the following reasons

- **Document Handling:** Langchain provides robust tools for managing document loading, splitting, and embedding, which simplifies the preprocessing steps necessary for efficiently working with large documents like PDFs.
- **Text Splitter:** The RecursiveCharacterTextSplitter is a crucial component because it breaks documents into smaller chunks that can be effectively processed and embedded. Without this, large documents would be challenging to handle within the context of a language model.
- **Vector Store:** Langchain's integration with Chroma for persistent vector storage makes it easy to store and retrieve document embeddings, ensuring fast retrieval of relevant text chunks when answering queries.

2. Sentence Transformers

Sentence transformers, specifically the SentenceTransformer class, were chosen for generating high-quality embeddings of both documents and queries. These embeddings are crucial for efficient similarity-based retrieval. Sentence transformers offer

- **Pre-trained Models:** They come with pre-trained models optimized for generating dense embeddings that capture semantic meaning.
- **Customization:** Sentence transformers allow the use of custom models and fine-tuning to suit specific needs, such as the custom embedding model (NomicEmbedEmbeddings) used in this system.
- **Performance:** The embeddings generated by these models are known for their accuracy in capturing contextual meaning, making them ideal for information retrieval tasks.

3. Chroma

Chroma is used as the vector store to persist the document embeddings. Its selection is based on

- **Efficiency:** Chroma is optimized for storing and retrieving embeddings at scale, which is crucial for handling large datasets like lecture notes in PDF format.
- **Fast Retrieval:** Chroma offers efficient similarity-based search, ensuring that the system can quickly find relevant document chunks for query answering.
- **Persistence:** It allows embeddings to be saved and loaded between sessions, ensuring that the system can handle long-term data storage without needing to regenerate embeddings each time.

4. Groq API

The Groq API provides access to the LLaMA-4 model. It was chosen for the following reasons

- **Ease of Integration:** The Groq API makes it easy to integrate a powerful language model into the system without needing to handle the heavy lifting of training or managing the model locally.
- **Real-time Query Processing:** With the Groq API, queries can be processed in real-time, ensuring that users get immediate responses based on the lecture notes.
- **Cost-Effective:** By using Groq's remote API, the system avoids the computational cost of running LLaMA-4 on local infrastructure.

5. Gradio

Gradio was chosen for building the front-end interface due to

- **Ease of Use:** Gradio makes it easy to create an interactive and user-friendly interface, which is crucial for a system that will be used by a variety of users.
- **Real-Time Interaction:** The Gradio interface is designed to support real-time interaction, allowing users to submit questions and receive answers instantly.

4. System Design

Overview

The system is designed to process lecture notes from PDFs, break them into manageable chunks, embed the content, and use a language model to generate accurate responses to user queries. The architecture of the system includes several key components

1. Document Loading and Preprocessing

- PDFs are loaded from the local directory or Google Drive using the PyPDFDirectoryLoader.
- The text is split into smaller chunks using RecursiveCharacterTextSplitter.

2. Embedding and Storage

- The content is encoded using the NomicEmbedEmbeddings model and stored in the Chroma vector store.
- Chunks are indexed by UUIDs to enable fast retrieval.

3. Query Processing

- When a user submits a query, the system searches the vector store for the most relevant document chunks.
- The relevant chunks are then passed to the Groq API with a structured prompt to generate an answer based on the content.

4. User Interface

- Gradio is used to build the front-end interface, where users can submit questions and receive answers.

5. Challenges and Lessons Learned

1. Handling Large PDFs

- **Challenge:** Processing large PDFs required efficient memory management. Many PDFs contained significant text that needed to be split and embedded in smaller chunks to ensure smooth processing.
- **Solution:** The RecursiveCharacterTextSplitter provided a way to split the text into smaller, manageable chunks that could be processed more easily. Experimentation with chunk size and overlap was essential to ensure optimal balance between performance and relevance.

2. Embedding and Retrieval Optimization

- **Challenge:** Efficiently storing and retrieving embeddings was key, especially as the number of documents increased.
- **Solution:** Using Chroma allowed for efficient vector storage and fast retrieval of relevant chunks. Fine-tuning the chunk size and overlap helped balance performance and memory consumption.

3. API Integration:

- **Challenge:** Integrating with the Groq API required careful management of API keys and handling potential errors like timeouts and rate limits.
- **Solution:** Error handling was added to ensure that the system could gracefully handle API failures, ensuring a smooth user experience.

4. Query-Document Matching:

- **Challenge:** Ensuring that the most relevant content was retrieved for each query.
- **Solution:** Using embeddings allowed the system to capture the semantic meaning of both queries and documents. The NomicEmbedEmbeddings model, fine-tuned with the prefix "search_document:" and "search_query:", helped achieve better matching.

6. Use of GenAI Tools (Prompts and Outputs)

```
AttributeError                                Traceback (most recent call last)
<ipython-input-14-3ccbb0d423d7> in <cell line: 0>()
    78
    79 uuids = [str(uuid4()) for _ in range(len(chunks))]
--> 80 vector_store.add_documents(documents=chunks, ids=uuids)
    81
    82 # Set up retriever

1 frames
/usr/local/lib/python3.11/dist-
packages/langchain_chroma/vectorstores.py in add_texts(self, texts,
metadatas, ids, **kwargs)
    525     texts = list(texts)
    526     if self.embedding_function is not None:
--> 527         embeddings =
self.embedding_function.embed_documents(texts)
    528     if metadatas:
    529         # fill metadatas with empty dicts if somebody

AttributeError: 'function' object has no attribute 'embed_documents'
```

The error you're encountering—`AttributeError: 'function' object has no attribute 'embed_documents'`—stems from passing a plain Python function as the `embedding_function` to the `Chroma` vector store. LangChain's `Chroma` expects an object that adheres to its `Embeddings` interface, which includes methods like `embed_documents`.

✅ Solution: Wrap the Embedding Function in a Custom Class

To integrate the Nomic Embed model properly, you should create a wrapper class that conforms to LangChain's expected interface. Here's how you can do it:

```
python                                                                    Copy Edit

from langchain.embeddings.base import Embeddings
from sentence_transformers import SentenceTransformer

class NomicEmbedEmbeddings(Embeddings):
    def __init__(self, model_name: str = "nomic-ai/nomic-embed-text-v1.5"):
        self.model = SentenceTransformer(model_name, trust_remote_code=True)

    def embed_documents(self, texts: list[str]) -> list[list[float]]:
        # Prefix each document with 'search_document:'
        prefixed_texts = [f"search_document: {text}" for text in texts]
        return self.model.encode(prefixed_texts, normalize_embeddings=True).tolist()



    def embed_query(self, text: str) -> list[float]:
```

the following shows an assignment given by the university. list few potential models that can be used for the LLM. give pros and cons of each model. note that we need to use RAG

"Develop a simple chatbot using an LLM to answer questions based on CTSE lecture notes (in pdf format). The chatbot should be implemented inside a Jupyter Notebook. use Agentic AI if needed. however make sure all the resources are free."

Based on your assignment — building a simple chatbot using an LLM with Retrieval-Augmented Generation (RAG) to answer questions based on CTSE lecture notes in PDF format, and implementing it in a Jupyter Notebook with free resources — here are a few potential LLMs you can use along with pros and cons of each:

♦ 1. Mistral 7B (via Hugging Face Transformers)

- Type: Open-source LLM
- Supports RAG:  Yes (can integrate with RAG pipelines like Haystack or LangChain)
- Free Access:  Yes (Hugging Face Inference or local use)



Pros:

- Lightweight enough to run on consumer GPUs or via Colab.
- High performance compared to earlier open models.
- Easy to integrate into RAG pipelines using LangChain.

Cons:

- Still requires decent hardware (at least 16GB VRAM for local use).
- Requires RAG setup (embedding model + vector store).

♦ 2. OpenAI GPT-3.5 (via API with free-tier)

- Type: Hosted LLM
- Supports RAG:  Yes (via LangChain or custom code)
- Free Access:  Limited (OpenAI offers some free API credits with sign-up)

Pros:

- High-quality responses and contextual understanding.

what other techniques that can be used to improve this?

Here are techniques you can use to improve accuracy, quality, and relevance of your RAG (Retrieval-Augmented Generation) system without significantly increasing response time, especially given your use of Hugging Face's Mistral API:

✓ 1. Prompt Engineering

► Use structured prompts to simulate reasoning

- Example: Add `Thought`, `Action`, `Answer` sections inside the prompt.
- Pros: Higher answer quality, more "agent-like" behavior.
- Cons: Slightly longer prompts.

► Add system-like constraints

text

Copy

Edit

[INST]

Answer only from provided notes. Think step-by-step. Avoid hallucination. Say "Not available" if

[/INST]

✓ 2. Context Optimization

► Use better chunking

- Current: `chunk_size=600`, `overlap=100`



if data directory doesnt exists. mount google drive and get pdf files from /content/drive/MyDrive/mistral . this location contains multiple pdfs. get them all

Here's how you can modify your code to:

1. Check if `data/` exists.
2. If not, automatically mount Google Drive.
3. Then copy all PDFs from `/content/drive/MyDrive/mistral/` to `data/` .

✓ Add This Block at the Start of Your Script:

python

Copy

Edit

```
import os
from pathlib import Path

# Step 1: Check if 'data' directory exists
DATA_PATH = "data"
if not os.path.exists(DATA_PATH):
    from google.colab import drive
    drive.mount('/content/drive')

    # Source and destination paths
    SOURCE_PATH = "/content/drive/MyDrive/mistral"
    DEST_PATH = Path(DATA_PATH)
    DEST_PATH.mkdir(parents=True, exist_ok=True)

    # Copy all PDFs from Google Drive
    for file_name in os.listdir(SOURCE_PATH):
        if file_name.endswith(".pdf"):
            src = os.path.join(SOURCE_PATH, file_name)
            dst = os.path.join(DATA_PATH, file_name)
            !cp "{src}" "{dst}"

    print("PDFs copied from Google Drive to 'data/' directory.")
else:
    print("'data/' directory already exists. Skipping Google Drive copy.")
```

7. Conclusion

This Groq-powered PDF question-answering system offers an effective solution for automatically answering questions based on lecture notes. By leveraging cutting-edge NLP techniques, including document embedding, similarity search, and Groq's LLaMA-4 model, the system provides accurate and relevant answers in real-time. The integration of various technologies like Langchain, Gradio, and Chroma, along with thoughtful handling of large PDFs and embeddings, ensures a robust and scalable solution for academic query handling.

8. References

- [1] “The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation,” Meta.com, 2022. [Online]. Available: <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>.