Retrieval-Augmented Generation Model for Academic Course Assistance

Mustafa Abbas 25100079 **Raahem Nabeel** 25100091 **Saad Haroon** 25100147 **Ibrahim Farrukh** 25100227 **Jawad Saeed** 25100094 **Safiullah Sarfaraz** 25100227

Abstract

This paper presents the development of a Retrieval-Augmented Generation (RAG) model designed to assist students at Lahore University of Management Sciences (LUMS) with course selection based on student feedback. The system integrates advanced document processing, vector storage, large language models (LLMs), and conversational memories to provide accurate and contextually relevant academic advice. The model utilizes historical course reviews that were collected through a Google Form circulated amongst the LUMS community. Additional reviews were amassed through a private group on Facebook called LUMS Discussion Forum. Lastly, the use of course outlines downloaded from the Registrar Office Portal of LUMS, provides additional context to the model in order to give more informed decisions, for study resources, component breakdowns and prerequesite hierarchies.

1 Introduction

The process of course selection at universities can be overwhelming for students due to the plethora of available courses and the significant implications of their choices over the course of their degrees. To assist students in making informed decisions, we developed a sophisticated system employing machine learning techniques. This system integrates components from the LangChain community to provide personalized course recommendations through an user friendly conversational interface.

2 Methodology

2.1 Data Collection

The data was collected through three primary means:

Google Form: A form circulated amongst the LUMS community to gather personalized reviews from students.

LUMS Discussion Forum: Manually scraped reviews posted by students on the private Facebook group.

Registrar Portal: Course outlines were downloaded from the Registrar's portal at LUMS, providing structure and context.

2.2 Data Annotation

We annotated the data corpus with metadata, including course abbreviations and aliases, to ensure relevant feedback retrieval even when queries use course nicknames.

2.3 Document Loading

The initial system design for document loading needed to handle various formats:

Text Documents: Handled through *Docx2txtLoader*, extracting text while preserving formatting and structure.

PDF Documents: *PyPDFLoader* loads PDFs into an array of documents where each document contains the page content and metadata such as page numbers.

3 Implementation

3.1 Data Collection

The data for the model was collected through three different means.

Google Form: A form was made and circulated amongst the LUMS community to get personalized reviews from students that have already taken particular courses. Students were requested to fill the form with the Course Name and its Code, their grade in the aforementioned course, review of the course workload, difficulty and exams and the course difficulty on a scale of 1-5. 1 represents the lowest rating with rudimentary concepts and a low workload while 5 represents the highest rating corresponding to advanced concepts with a high workload.

LUMS Discussion Forum: To expand our dataset we scraped manually through course reviews posted by students on the private LDF group on Facebook. We searched course wise on LDF and entered the reviews manually into our form based on the reviewer's experience.

Registrar Portal: Lastly to obtain the course outlines for all the courses in our dataset we used the Registrar Portal site for LUMS students. From there we manually downloaded the required outlines and added them into the course reviews folder in our directory.

We were able to gather a total of **471** reviews, along with **170** different course outlines.

3.2 Data Annotation:

In addition to the data collection we manually annotated the corpus (all 471 reviews) with metadata. Firstly, we included all combinations of abbreviations or aliases for the course names, including the course code. This was done to ensure that relevant course feedbacks are retrieved, even when the user queries using a course abbreviation or nickname, for example, 'Netcen' instead of CS-382: Network Centric Computing. Moreover, we added a field in each review to include which year the particular course is taken and in which semesters it is normally offered. This was done to cater to queries requesting advice for a particular semester or a year, for example: "What are some good CS courses offered in Fall semester?"

3.3 Document Loading

Document loading is a critical initial step in our system that involves ingesting and transforming raw textual materials into a structured digital format.

Text Documents: To convert the extracted csv file from the form responses into separate word documents for each review, a python script was used. This ensured ease in processing down the pipeline. We utilize the *Docx2txtLoader*, which extracts text content while preserving basic formatting and structure, which is crucial for maintaining the semantic integrity of the document to improve the model performance. Additionally

PDF Documents: The course outline documents were in PDF format, which posed significant challenges due to the presence of mixed content such as images, tables, and multi-column layouts. For this purpose, we employed *PyPDFLoader*. It's designed to handle complex layouts by using advanced techniques to extract text. This loader also includes capabilities to handle encrypted PDF files, ensuring that all publicly available academic content, regardless of format protections, is accessible to our system.

3.4 Vector Storage and Retrieval

To manage and retrieve the processed documents efficiently, we employ Chroma, a vector storage system that indexes documents using semantic embeddings generated by *nomic-embed-text* from OllamaEmbeddings. Chroma DB is a free, open source embedding database. This allows for high precision in retrieving documents closely related to a student's query based on semantic similarity. We compared and contrasted several embedding models, but nomic-embed-text stood out for the following reasons. Firstly, it is free, unlike other embedding models such as OpenAI's *text-embedding-ada-002* which operates as a pay as you go basis. Additionally, in terms of retrieval average scores, the model achieves a state of the art score which is measured in terms of Normalized Discounted Cumulative Gain. Furthermore, it requires the least amount of memory as compared the models we tested (Figure 1). The used search metric is similarity instead of Maximal Marginal Relevance (MMR) to reduce creativity in retriever to prevent irrelevant documents to be fetched.

3.5 Large Language Model

The core of our conversational interface is powered by a state-of-the-art *Mistral-7B-Instruct-v0.2* from Hugging Face. This model is capable of understanding and generating natural language re-

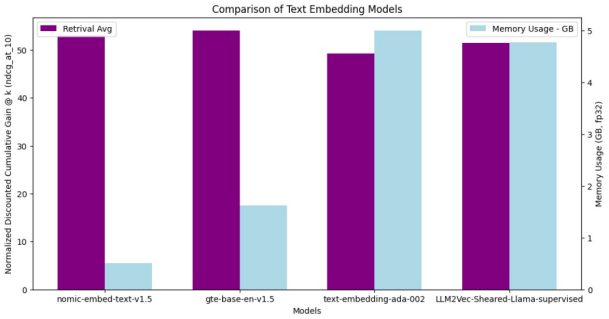

Figure 1: Comparison of Embedding Models

sponses, facilitating an interactive dialogue with the user. It boasts an impressive 32k Context Window with no sliding window attention. (Figure 2)

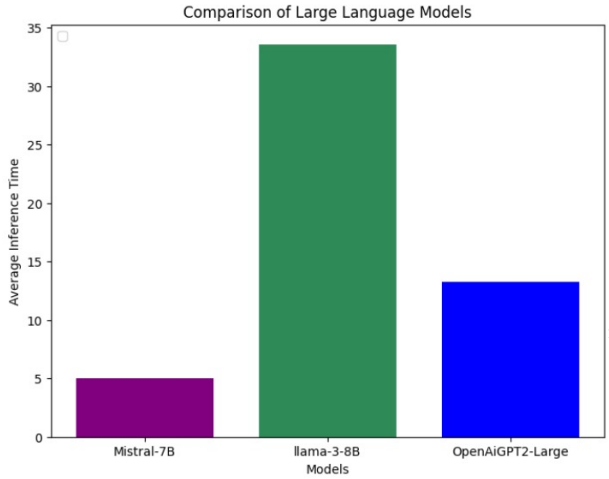

Figure 2: Average Inference Times of LLMs

From the graph above we tested the average inference times for different LLMs to see which one was the most efficient. From our testing we found out that the most efficient was the Mistral-7B model which returned user's queries in around 5 seconds. In comparison the least efficient out of all the tested models was Meta's LLAMA-3 8B model which took more than 30 seconds on average to answer queries.

In addition to the models above we tested the model with multiple LLMs as follows:

1. Google's Gemma-2B

   This is an comparatively smaller LLM that was unable to work since it does not support max output tokens greater than 250 making it not suitable for our use case.

2. NexaAI's Octopus V4

   This LLM was too large to load in the HuggingFace Inference API because of its size.

3. Snowflake's Arctic Instruct

   Similar issue as above as it was not loading in the HuggingFace Inference API.

4. Apple's OpenELM-3B-Instruct

   Facing issues with the remote execution of the code on local machines

5. Mixtral-8x22B

   The model failed to load on local machines due to the model being too large to load locally (281GB $>$ 10GB).

Prompt Template

Perhaps the most important aspect in getting the most out of any LLM is the prompt. For our purpose, we used an instruct fine tuned model. The prompt template we used is as follows:

```
"""
You are a professional chatbot assistant for helping students at LUMS regarding course selection.
Please follow the following rules:
1. Answer the question in your own words from the context given to you.
2. If you don't know the answer, don't try to make up an answer.
3. If you don't have a course's review or outline, just say that you do not know about this course.
4. If a user enters a course code (e.g. ECON100 or CS370), match it with reviews with that course code. If the user enters a course name (e.g. Introduction to Economics or Database Systems), match it with reviews with that course name.
```

```
5. If you do not have information of a
    course, do not make up a course or
    suggest courses from universities
    other than LUMS.
Context: {context}
You are having a conversation with a
    student at LUMS.
Chat History: {history}
Human: {question}
Assistant:
"""
```

The `context` contains the top most similar documents retrieved by the retriever. `history` contains the buffer history of the past three question answer pairs with the user. `question` contains the raw prompt as entered by the user. The first rule is to prevent the LLM from parroting the reviews. This ensures a free flowing conversational style of responses. The second, third and fifth rules are to prevent the LLM from hallucinating about any course that it does not have any information about. LLMs are notorious for making up information and presenting it in a convincing manner, which is why we set up stern rules preventing it from doing so. The fourth rule instructs the model to match course codes to the reviews present in the context.

3.6 Conversational Memory

Our system includes a conversational memory that helps maintain context throughout interactions. This feature enhances the LLM's ability to provide relevant and coherent responses over the course of a conversation. To achieve this we made use of *ConversationBufferMemory* to keep track of the last K interactions of the model with the user. After extensive testing we found a value of K=3 to be optimal for keeping context from the previous prompts in case the user wants to ask follow up questions.

3.7 User Interface

A Gradio-based web interface serves as the point of interaction for students. It provides a user-friendly environment where students can ask questions and receive advice on course selection without needing technical knowledge.
Gradio generates a link for testing on the user's local machine alongside a shareable link that is accessible by anyone across the world provided that the machine that runs it remains accessible. In case Gradio servers go down we have also an alternative hosting solution in the form of ngrok which makes a locally hosted application globally accessible.

3.8 Use Cases

The following use case shows how the model is able to provide assistance to the users. When asked about the reviews for CS331: Intro to AI, the model is successfully able to retrieve the relevant reviews, and is able to formulate a comprehensive and calculated response.

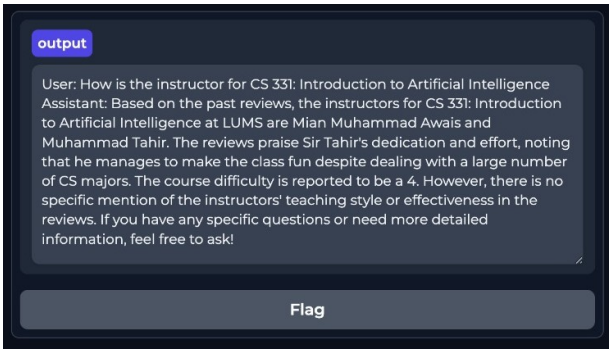

Figure 3: Reviews for CS331

In the following prompt, the user asked for MECO reviews. The model, once again, retrieves relevant reviews and gives a well reasoned response.

Figure 4: Reviews for MECO

4 Conclusion

The developed RAG model significantly enhances the support available to students during the course selection process at LUMS. By integrating advanced machine learning techniques and user-friendly interfaces, the system effectively guides students in making informed decisions.

References

Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar, "Nomic Embed: Training a Reproducible Long Context Text Embedder," 2024. arXiv:2402.01613 [cs.CL].

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed, "Mistral 7B," 2023. arXiv preprint arXiv:2310.06825 [cs.CL].

Zhi Jing, Yongye Su, Yikun Han, Bo Yuan, Haiyun Xu, Chunjiang Liu, Kehai Chen, and Min Zhang, "When Large Language Models Meet Vector Databases: A Survey," 2024. arXiv preprint arXiv:2402.01763 [cs.DB].

Keivalya Pandya and Mehfuza Holia, "Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for organizations," 2023. arXiv preprint arXiv:2310.05421 [cs.CL].

https://www.awesomescreenshot.com/blog/knowledge/chat-gpt-api

https://www.shakudo.io/blog/build-pdf-bot-open-source-llms

https://www.datacamp.com/blog/the-top-5-vector-databases

https://www.youtube.com/watch?v=vXmpThOZiIM

https://tech.smile.eu/your-private-gpt-to-chat-with-your-documents/

https://medium.com/rahasak/creating-custom-chatgpt-with-your-own-dataset-using-openai-gpt-3-5-model-llamaindex-a

https://www.datacamp.com/tutorial/how-to-build-llm-applications-with-langchain

https://www.datacamp.com/blog/what-is-retrieval-augmented-generation-rag

https://docs.llamaindex.ai/en/stable/understanding/using_llms/using_llms.html

https://python.langchain.com/docs/get_started/quickstart#conversation-retrieval-chain

https://awinml.github.io/llm-ggml-python/

https://ollama.com/library

https://medium.aiplanet.com/implementing-rag-using-langchain-ollama-and-ch

https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/5_Levels_Of_Text_Splitting.ipynb

https://medium.com/aimonks/multiple-pdf-chatbot-using-langchain-b3ee2296b

https://python.langchain.com/docs/expression_language/cookbook/retrieval

https://medium.com/@iryna230520/first-steps-in-langchain-the-ultimate-guide-fo

https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

https://www.sbert.net/index.html

https://docs.llamaindex.ai/en/stable/examples/node_parsers/semantic_chunking.html

https://python.langchain.com/docs/modules/data_connection/document_transformers/semantic-chunker

Individual Contributions

- Raahem Nabeel
  - Contributed to Creating the Retrieval Augmented Pipeline architecture from scratch
  - Contributed to debugging and refining the code.
  - Assisted in the development of the user interface on Gradio.
  - Data Annotation and Pre-processing
  - Collecting LDF reviews for summer courses.
  - Comparing and Contrasting different LLM and Embedding Models
- Jawad Saeed
  - Contributed to Creating the Retrieval Augmented Pipeline architecture from scratch
  - Implemented and optimized the vector embedding processes.
  - Contributed in the Project Report
  - Assisted in the development of the user interface on Gradio.
  - Experimentation with different LLMs and Embeddings models to improve performance
- Ibrahim Farrukh
  - Contributed to Creating the Retrieval Augmented Pipeline architecture from scratch
  - Took the lead on the Data Collection, Optimization and Annotation Process
  - Collecting LDF reviews for summer courses.
  - Contributed in the Project Report
  - Wrote Customised Scripts to extract relevant Course Outlines
- Safiullah Sarfaraz
  - Contributed to Creating the Retrieval Augmented Pipeline architecture from scratch
  - Took the lead on the Data Collection, Optimization, and Annotation Process
  - Dataset Gathering
  - Exploratory Data Analysis
  - Data Annotation and Pre-processing
  - Contributed to debugging and refining the code.
- Mustafa Abbas
  - Led the documentation and Report
  - Led the research on different LLM models suited to our pipeline
  - Contributed to Creating the Retrieval Augmented Pipeline architecture from scratch
  - Assisted in the development of the user interface on Gradio.
  - Contributed in the Project Report
- Saad Haroon
  - Contributed to Creating the Retrieval Augmented Pipeline architecture from scratch
  - Designed the Project Poster
  - Project Report
  - Implemented and optimized the vector embedding processes.
  - Comparing and Contrasting different LLM and Embedding Models
  - Led the research on different Vector Embedding models suited to our pipeline