MADS-RAG: A Helpful Chatbot for Master's of Applied Data Science Students at the University of Michigan

SIADS 699 Capstone (Winter 2024) Final Report

Team 28: Patrick Sollars, Aaron Newman, and Tawfiq Zureiq

22 April 2024

Project Statement

New students in the Master's of Applied Data Science (MADS) program at the University of Michigan have a lot of questions as they're trying to navigate a new remote learning environment. We see a lot of these questions on the MADS Slack instance and it is fortunate that there are many helpful people, including faculty, staff, fellow students, and alumni, who frequently will provide answers.

As data scientists, we saw an opportunity to create a solution that takes advantage of modern generative artificial intelligence (AI) techniques. In particular, this course allowed us to experiment with techniques that we haven't had much exposure to in MADS classwork. We built a Large Language Model (LLM)-based chatbot to help answer frequently asked questions about the MADS program and its classes.

Those familiar with related technologies may say that we can have ChatGPT do this for us for a fee. In November 2023, OpenAI, the company developing ChatGPT, introduced the ability to create custom Generative Pre-trained Transformer-based chatbots, or GPTs [1]. A user can choose from a range of existing pre-built GPTs on particular subjects, or provide specific domain information to create a fine-tuned chatbot. For this project, we wanted to do the work ourselves for the following reasons:

1. **Direct Experience:** We deepened our understanding of generative AI by leveraging open source components and building the application ourselves.
2. **Privacy and Control:** Building our own solution allowed us to keep source information (including proprietary course content) as private as possible.
3. **Cost Efficiency:** We designed our solution to operate on minimal resources. It is intended to run on a modern, reasonably powerful laptop.

Our primary focus for this project is on the backend - the technologies that help to answer questions as accurately as possible. We also developed a user interface that allows people to ask questions and receive their answers. In the next section, we'll discuss our approach to solving this problem.

Methodology

We had two basic approaches to consider for this project. We could select an open source base LLM and fine tune it with information about the MADS program and its classes. Alternatively, we could use a Retrieval-Augmented Generation (RAG) approach where we do not make any

changes to the base model, but inject relevant information into each query to obtain a more well-informed answer from the LLM.

Using a fine-tuned LLM would offer a simpler question answering pipeline with fewer components. However, the fine-tuning process can be computationally (and financially) expensive, and also static. Once the model is tuned, its information is fixed. Updating the model to reflect new information requires starting from the beginning and fine-tuning the model once again.

With a RAG approach, we have the additional complexity of preparing and storing source information in a way that is easily accessible to the LLM. However, it is much simpler to add new information and remove information that is determined to be incorrect or out-of-date. Early in our research, we reviewed a *Medium* article by Rubentak [2] that provided a tutorial on building a local RAG pipeline. We used this as inspiration to build a proof of concept and identify the different pipeline components that we would need to implement, from cleaning source data to providing an effective user interface.

Given our constraints and our goals for this project a RAG application was clearly a preferable solution. Our implementation focused on the seven stages described in the following paragraphs.

1. Selecting Relevant Source Information

To answer the types of questions we most frequently see in MADS Slack, the most useful information appears to come from class syllabi, the MADS Student Handbook, and the MADS Student Advising Guide. The most recent class syllabi are available to current students, of course. For this application, we chose to go to the University of Michigan School of Information (UMSI) website, where MADS courses are described, with links to syllabus documents [3]. We recognize that the most recent syllabi are not always made available at this site, but we felt the information would be sufficient to examine the usefulness of the application. (For a "production" version of this chatbot, we assume that the most current syllabi would be made available and would be kept updated as changes are made.) The MADS Student Handbook [4] and Advising Guide [5] are available as Google Docs from the MADS program's Google Drive. These documents are available to the public, although one needs to know the link in order to access them. As shown in Figure 1 below, we performed topic modeling on this set of documents and categorized the results into the following 7 subjects that are relevant to use as prompts for this RAG application.

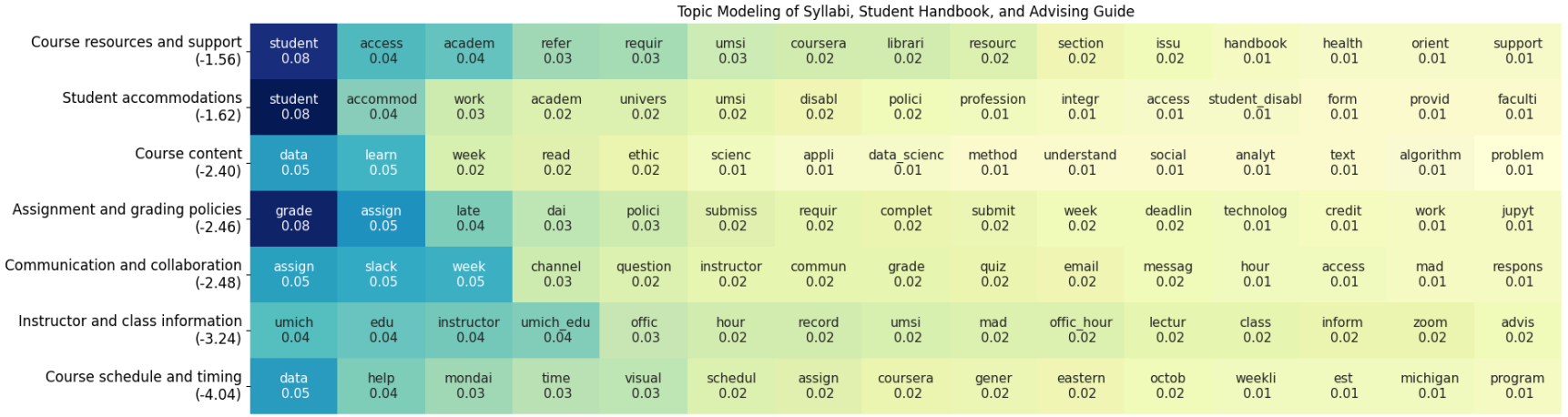

Figure 1: Topic Modeling on Syllabi, Student Handbook, and Advising Guide

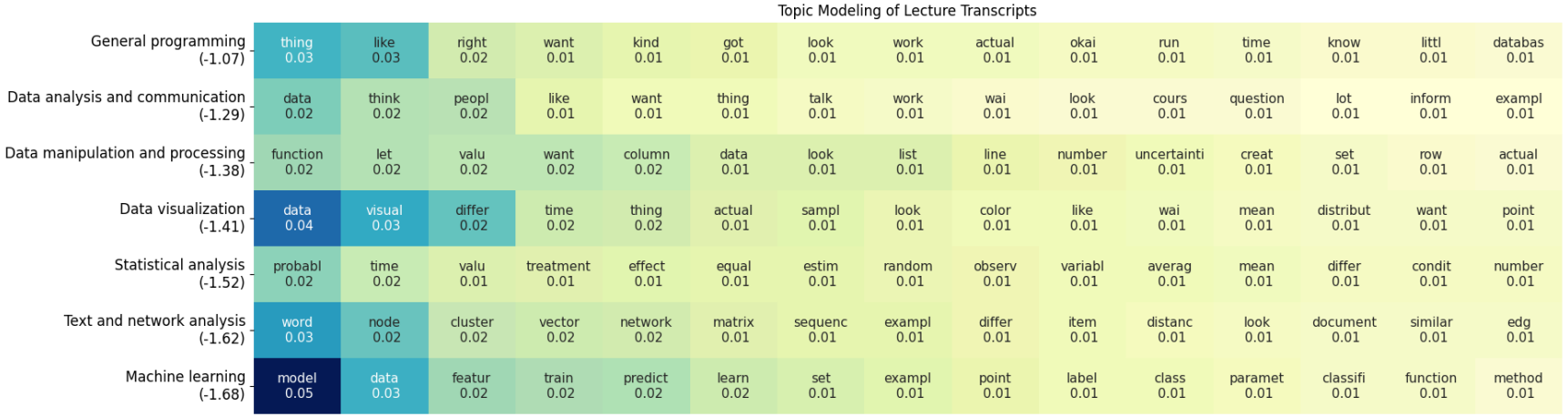

Figure 2: Topic Modeling on Video Lecture Transcripts

As we progressed through the project, we were able to secure permission to use the content from the video lectures we reviewed as students within the Coursera environment. We were cautioned not to use transcripts from any office hours videos to avoid exposure of any student information subject to the Family Education Rights and Privacy Act (FERPA) restrictions. We were able to download these transcripts in bulk from our individual Coursera accounts using the cs-dlp [6] script. We performed the same topic modeling analysis for this set of documents (see Figure 2), which give a vague, but accurate, sense of the MADS program content. As these documents were added later in the project, we focused our evaluations on the syllabi documents, student handbook and advising guide.

2. RAG Pipeline Components

Based on our positive experience with Rubentak's article, we decided to use the LangChain Python package [7] to build and manage our RAG pipeline. LangChain is under active development and appears to be in wide use for development of LLM-based applications. The package allows the developer to create pipelines that integrate various components which may include third-party capabilities. For our application, we chose to use the following:

- Original: Chroma as a vector store. We need a means to index and quickly retrieve the appropriate information to support a given query. Chroma [8] provides that capability, is relatively easy to use, well-integrated with LangChain, and has flexible deployment options, including in-memory, persistent to local disk, client/server, or as a Docker container.
- Later: ColBERT in lieu of a vector store. During our evaluations, we tried different combinations of search strategies using Chroma-based document retrieval. While we found reasonable results, when we attempted to use ColBERT [9] (with its LangChain-compatible wrapper, RAGatouille [10]), we discovered that it retrieved more responsive documents and produced better final answers to our test questions. The only downside is that it is more difficult to make changes to a ColBERT index than it is to a Chroma collection.
- llama-cpp-python for serving LLMs. Our goal is to produce a tool that can be run locally, but we recognize that there may be a need to work with a cloud-based model in a production version of our application. The llama-cpp-python package [11] can wrap various LLMs and provide a unified, OpenAI-compatible application programming interface (API). With that API, we can quickly change out the underlying model (and where it is served from, be it local or remote) with no change to the code that works with it.

We discuss the choice of LangChain-compatible embeddings and retrievers in later paragraphs.

3. Choosing LLMs

In a RAG-based application, information retrieval is probably the most important thing to get right, but the LLM is a foundational part of the pipeline as it ultimately generates the response to the question. The Rubentak article used a version of Mistral AI's Mistral 7B model [12] for its RAG pipeline, and that became our natural starting point. Mistral 7B has documented improved

performance over other seven billion parameter models, and also has an Apache 2.0 license that would allow its use with very few restrictions. It also was trained with a 32K context window. Because it can handle relatively large inputs, we can provide it with more information to use for inference than other models.

For ease of use on modest hardware, we used a four-bit quantized version of the model that was available on the popular Hugging Face repository. We used models in the GPT-Generated Unified Format (GGUF) binary file format, as they are compact and work well with llama-cpp-python.

We did also want to see how performance varied with other LLM choices. We also considered:

- Meta's Llama-2 (both 7B and 13B versions)
- Berkeley-Nest's Starling LM 7B Alpha and/or Beta

While one could use any of these models and produce good results, we found that Mistral provided the most consistent answers to our test questions during evaluation, and so that remained our primary choice.

4. Source Data Preprocessing Strategies

Frameworks like LangChain provide several different means of loading and preprocessing source data so that it is as clean as possible and as easy to retrieve as possible. For course syllabi (usually in PDF format) and the other documents (usually Google Docs), we decided to convert them all into Markdown format. We wrote a custom Markdown document loader to retain the various section headings in document chunks that would help document retrievers to identify the right content. We used a relatively large chunk size target of 1500 characters. Actual chunk sizes varied slightly depending on paragraph and section breaks, and since our logic only broke sections apart on line breaks a few ended up much larger. Our custom loader also included the metadata we needed to be able to identify the document sources, and later, display that relevant context information to the user interface. Transcripts were split by LangChain's built-in SemanticChunker class, based on a document retrieval demo by Greg Kamradt [13]. As shown in the histograms below, while chunking by document section (Figure 3) is much more random, semantic chunking (Figure 4) results close to an exponential distribution.

This strategy worked well when using Chroma as a vector store. However, ColBERT's indexing doesn't work well for text chunks larger than 512 tokens, so we allowed the ColBERT indexer to further split the documents to stay under that limit.

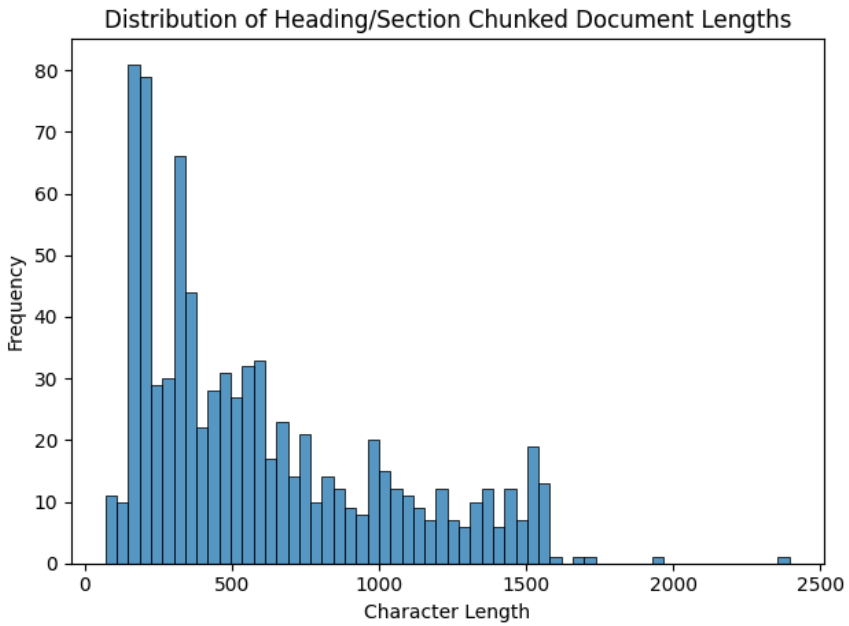

Figure 3: Heading/Section Chunking Size Distribution

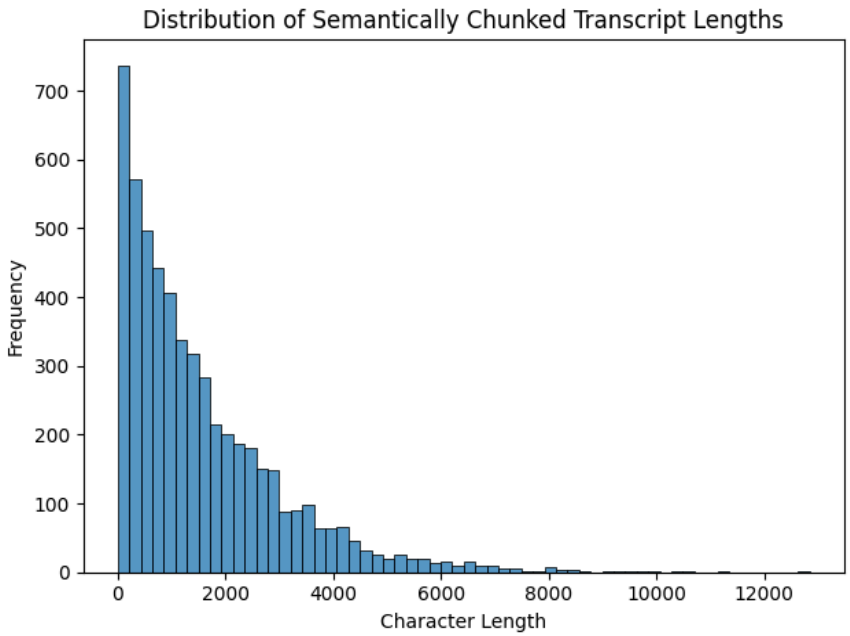

Figure 4: Semantic Chunking Size Distribution

5. Choose the Best Embeddings Strategy

LangChain supports a number of different embedding strategies, including the Sentence-Transformers models that are hosted on Hugging Face. During our initial setup, we focused on the default model, all-MiniLM-L6-v2 [14]. Although there are models that have demonstrated superior performance on the Massive Text Embedding Benchmark (MTEB) Leaderboard, this one offers good performance with modest computing resources.

ColBERT uses a different strategy as a late interaction model that performs embeddings at the token level rather than the document level. According to ColBERT's documentation, it "encodes each passage into a matrix of token-level embeddings... Then at search time, it embeds every query into another matrix… and efficiently finds passages that contextually match the query using scalable vector-similarity… operators." [9]

6. Information Retrieval Techniques

Once our documents are stored and indexed, we next need a capability to find the most relevant documents for a given query. LangChain has several retrievers available, and allows the user to vary the number of documents retrieved, as well as to include arguments to change the search type or to filter documents based on different criteria.

ColBERT leverages a RAGatouille "wrapper" to create a LangChain-compatible retriever. At first, we kept information from the syllabi and the transcripts in different indices, so we used LangChain's EnsembleRetriever to bring back information from both. However, we later realized that this was inefficient, particularly at startup, so we created a new single index with all of the source documents, and then only needed a single retriever.

7. User Interface

The focus of our efforts was on producing the RAG pipeline that offers the best question answering capabilities. Naturally, we still need to produce a means of interacting with the pipeline. Streamlit [15] is a popular Python-based capability to quickly create a user interface for data-based applications. We also created a version using Gradio [16], a Streamlit competitor. Both produce user interfaces of sufficient quality to test our pipeline.

Evaluation Strategy

We first considered questions we would expect typical users to ask based on our experiences as MADS students. We came up with a list of 20 sample questions shown below in Figure 5. We picked questions where the information would likely appear in different source documents as well as a few that might confuse the pipeline, including referring to a class by its class number and then the same question with the class by its name.

- Which class involves time series analysis?
- Who teaches the SQL and Databases class?
- What are the prerequisites for Data Science for Social Good?
- When are the office hours for the Math Methods course?
- Are there any weekly readings for Milestone II?
- What are the outcomes of Qualitative Inquiry?
- What textbook is required for SIADS 505?
- What textbook is required for Data Manipulation?
- Which week of unsupervised learning covers DBSCAN?
- How many credits are required to complete the MADS program?
- How long do students have to complete the MADS program start to finish?
- How many points is the comprehensive oral exam worth in SIADS 593?
- What is the penalty for late submission in SIADS 630?
- How do I get accommodations for a class?
- What is a backpack?
- When is the latest I can drop a course?
- How do I get an override to take a class?
- How do I take a leave of absence from the MADS program?
- What are the prerequisites for Search and Recommender Systems?

Figure 5: Evaluation Questions

We put together a series of Jupyter notebooks to run these 20 questions through the pipeline, while varying:

- The embeddings used, trying several Sentence-Transformers models as well as ColBERT,
- The search strategy used by the document retriever, including Chroma's similarity search, maximum marginal relevance search, and ColBERT as a retriever, and
- The underlying LLM, including Mistral 7B, Starling 7B, and Llama 2 (both 7B and 13B). In all cases, four-bit quantized versions in GGUF format were used.

Our primary measure of effectiveness was what we called the "Eye Check:" simply reviewing the answers provided compared to the ground truth, and using human evaluation to determine whether the answer is correct. It is simply a binary choice.

There are a number of more formal evaluation metrics and methods that one can use with LLMs and RAG pipelines, including: BLEU, ROUGE, METEOR, BERTScore, and various metrics offered by the Ragas Python library. We tried several of these methods, but often found that they did not seem to correlate with whether we considered an answer to be correct.

An example of BERTScore measures versus our Eye Check are shown in Figure 6 below. In the figure, we can see that a normalized F1 score as high as 0.42 corresponded to an incorrect answer, while a score of -0.15 was associated with a correct answer. Figure 7 shows a similar example using Ragas. In that case, we had a correct answer that Ragas assigned an answer relevancy of zero, and an incorrect answer with an answer relevancy of one. The largest frustration with Ragas was the amount of time it took to generate results. Unable to run Ragas locally, we used the commercial OpenAI API to run metrics, but found that we ran into rate limits even with the paid service. In the end, we decided that the Eye Check would be the best performance metric.

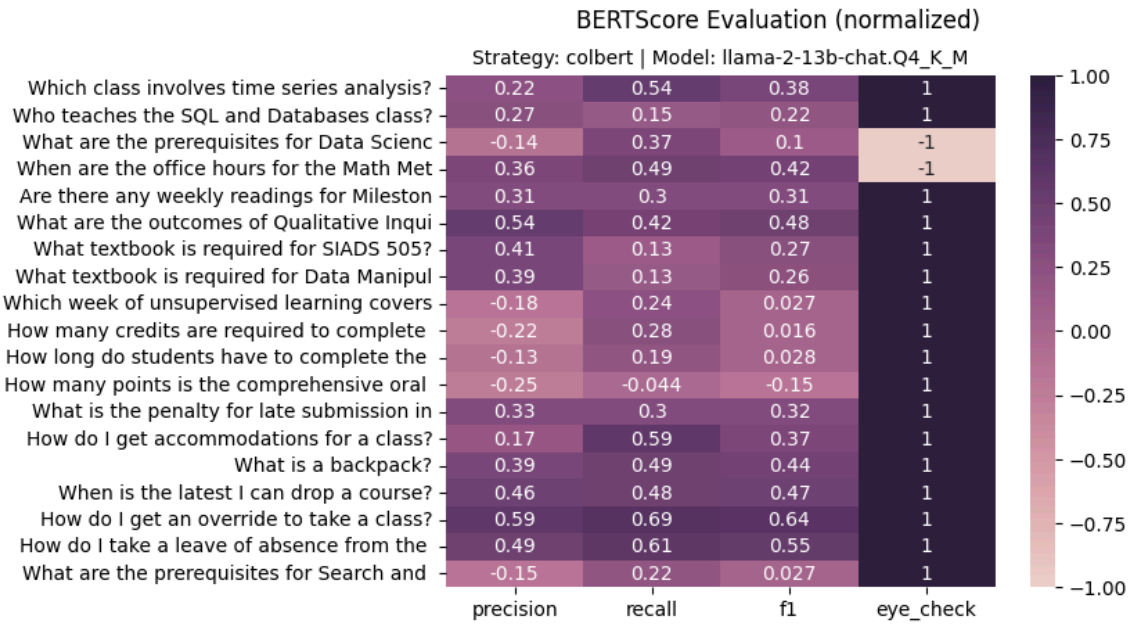

Figure 6: Example Evaluation with BERTScore

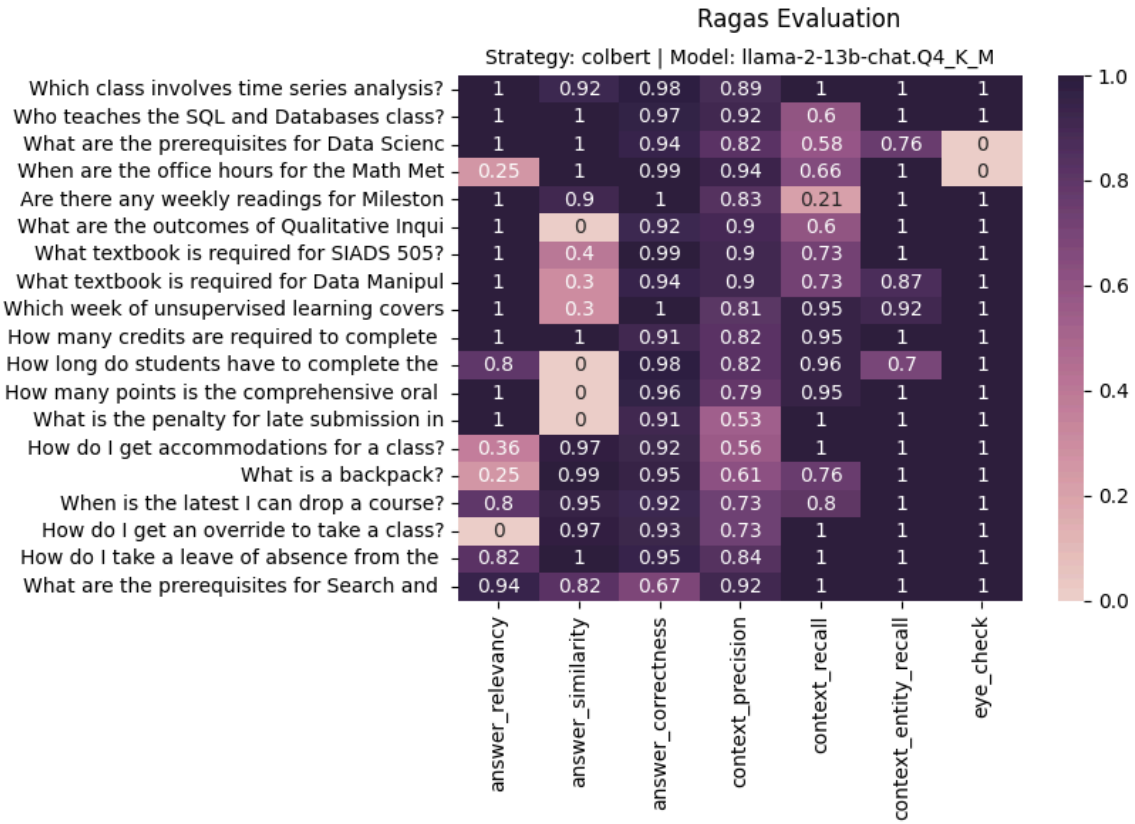

Figure 7: Example Evaluation with Ragas

Results

The only combination of models, document indexing, and retrieval that produced 20 correct answers out of 20 was:

- LLM: Mistral 7B Instruct v0.2 (4-bit quantized, GGUF format)
- Document Indexing: ColBERT
- Retrieval Strategy: Return top five documents

With this baseline, we explored the impacts of using the video lecture transcripts alongside the other documents. We were able to obtain reasonable answers about more subject-related questions, such as “How does PCA work?”, that we could not answer from the syllabi.

We initially created two separate indices for the syllabi and the transcripts but found that this strategy increased initial loading time as well as the chatbot’s response time. So, we merged the information into a single index. Generally, this did not impact the accuracy of answers to our syllabus-focused test questions.

The final enhancement to the pipeline we consider was the inclusion of semantic routing, using Aurelio AI’s open-source Semantic Router package [17]. The tool offers a broad range of capabilities, many of which are beyond the scope of this effort. We focused on training a router to recognize if/when a user query was focused on a particular class (identified by class number and/or name). If we could find a match to a class, we would then limit the document retriever’s scope to only those documents that were marked with metadata about that class. In the event no match was found, the retriever could access all documents in the index.

We saw a slight improvement in response time when the retriever was effectively routed. There were also cases where the router identified the class incorrectly, and therefore was unable to answer the question. Adding more relevant sample questions to each route would likely improve performance. Given the current state of the work, the limited tuning of the routers we attempted, and the relatively small size of the source document corpus, we saw better results by leaving the router out.

Figure 8 below describes the RAG architecture that we used in the final configuration of this application. The Blue pipeline describes pre-processing stages to load the ColBERT indexes. The Maize pipeline shows the information retrieval and generation steps that are orchestrated by LangChain, while the entrypoint and outputs are displayed in the Streamlit UI.

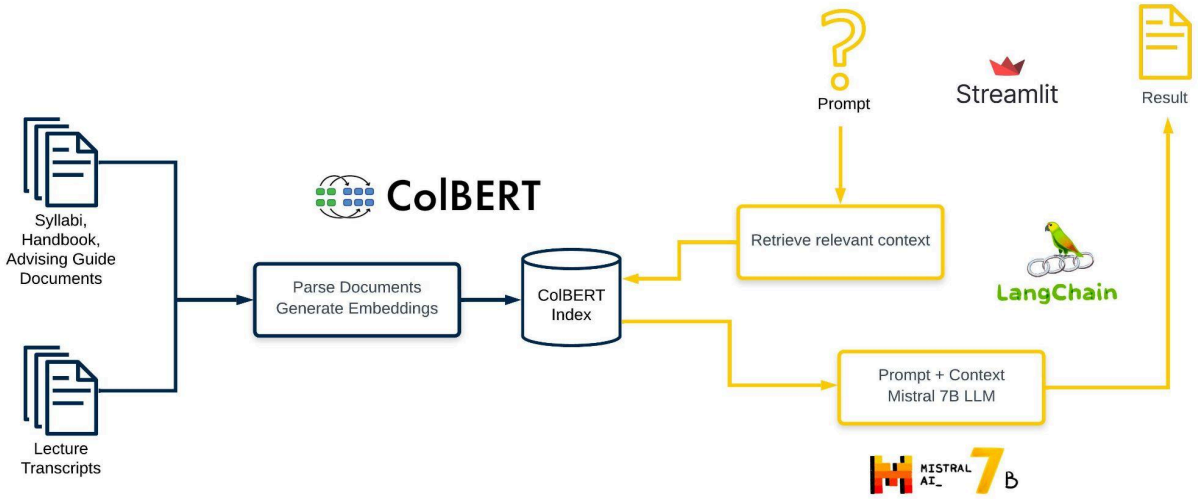

Figure 8: Final RAG Architecture

Finally, Figure 9 below shows a prototype user interface with a few questions answered. Information about source documents is visible at the bottom.

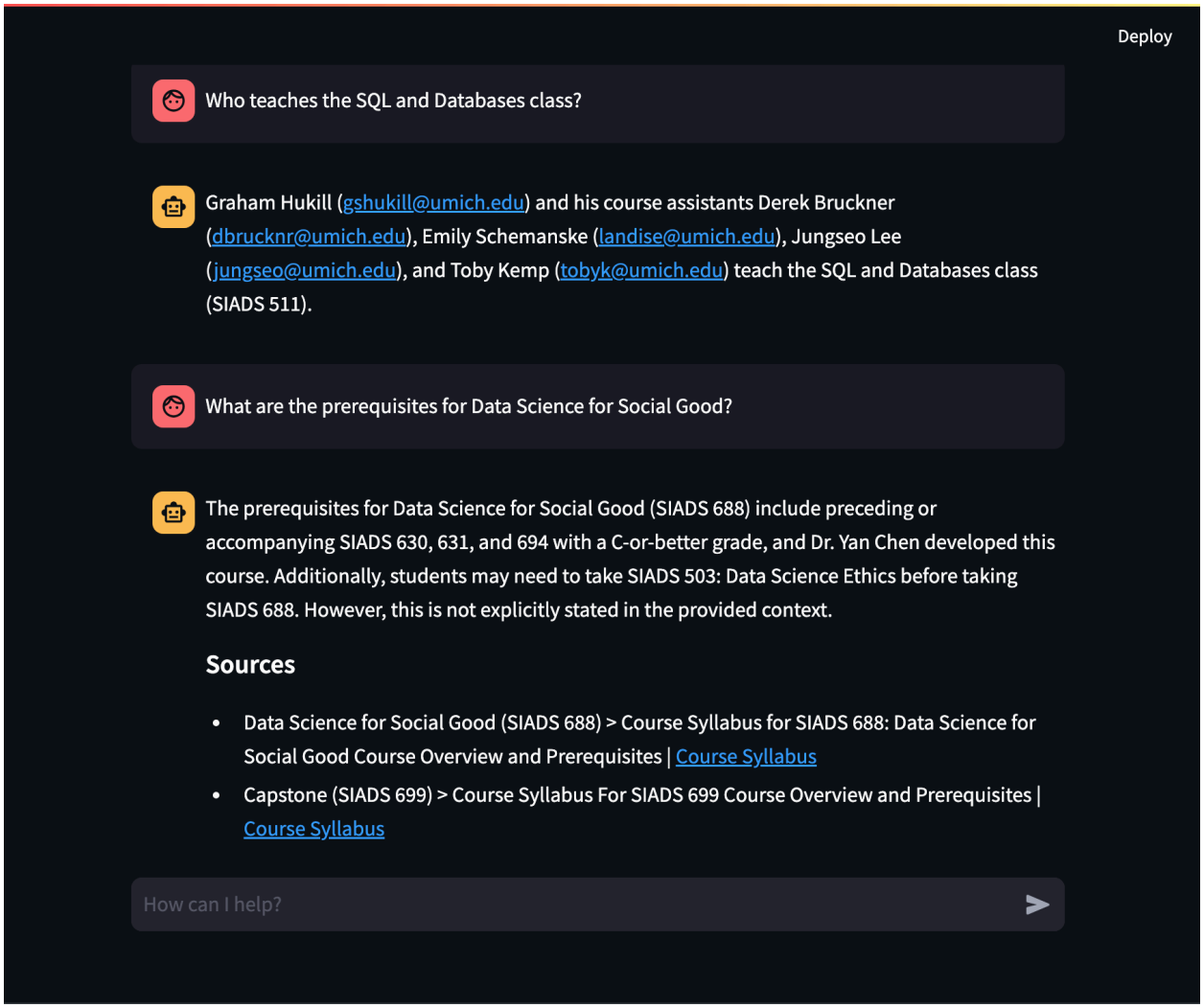

Figure 9: MADS-RAG User Interface

Broader Impacts

The largest issue with the state of the art in LLMs is that they are not always correct in their assertions, and can be misleading, if not outright incorrect. For an application like this, it is very important to communicate to the user (through the user interface and any user documentation) that any advice they receive should be verified, and that the application cannot be considered an authoritative source.

In fact, there is some literature that suggests that the opacity of LLMs makes them unsuitable for any search-related application. Shah and Bender [18] argue that reliance on tools like these have a negative impact on understanding, and cause atrophy in human skills, including "information verification, information literacy, and serendipity." We can agree that over-reliance on LLMs for question answering could cause people to avoid doing research themselves, and/or trust the results they receive without verification. On the other hand, LLMs themselves may provide some serendipity, as they may provide information that the user would not have

discovered through other means. This issue is far larger than our little application. While the broader community continues to debate, it remains important to inform the user clearly on what the tool can and cannot do.

The prompt template used by our RAG pipeline instructs the LLM to use only the contextual information provided by the query to develop its answer. However, we cannot eliminate the possibility that the model will hallucinate and/or use information from its underlying training. There have been many instances in the media and technical literature where bias in model training has been demonstrated, and so it is possible that some of that bias may be exhibited in a given response from the LLM, to include things that may seem inappropriate to the user. Fortunately, we have not seen this behavior in our testing to date.

Finally, many have expressed concerns about LLM-based applications exposing private information. That is one of the reasons we chose to build a local RAG pipeline, so that we controlled what information was used (and how it was used) in the application. Our source data included no personally identifiable information, apart from the name and e-mail address of certain faculty and staff members drawn from publicly available documents. We are confident that the risk of exposing private information is acceptably low.

Conclusion

This has been a very interesting project to conclude our journey in the MADS program. It has allowed us to reflect on what we've learned, make use of some relatively novel artificial intelligence technologies, and recognize that while they are promising, they are not perfect. We hope that future MADS Capstone students will take our work and extend it to make it a useful tool for people in the program.

Statement of Work

- **Patrick Sollars** - Document Processing Strategy and Execution, Evaluation Techniques, GitHub repository management
- **Aaron Newman** - Initial Concept, Pipeline Development, Proposal Drafting, Final Report Drafting
- **Tawfiq Zureiq** - Semantic Router, Streamlit UI
- **All** - Standups, Peer Reviews, Prototyping, Proposal and Final Report Editing

References

1. OpenAI, “Introducing GPTs,” *OpenAI Blog*, 6 November 2023. Accessed on 30 March 2024 at https://openai.com/blog/introducing-gpts.
2. Rubentak, “Talk to your files in a local RAG application using Mistral 7B, LangChain and Chroma DB (No internet needed),” *Medium*, 24 October 2023. Accessed on 30 March 2024 at https://medium.com/@rubentak/talk-to-your-files-in-a-local-rag-application-using-mistral-7b-langchain-and-chroma-db-no-2b4ba77358e0.
3. University of Michigan School of Information (UMSI), “MADS Courses,” *UMSI Website*. Accessed on 30 March 2024 at https://www.si.umich.edu/programs/master-applied-data-science/curriculum/mads-courses.
4. University of Michigan, *MADS Student Handbook*, 1 February 2024. Accessed on 30 March 2024 at https://docs.google.com/document/d/1YEOcpdONdme5kmpNEnZpdbJeVFhEIw1pS0wq16QdH1I/.
5. University of Michigan School of Information, *Academic Advising Frequently Asked Questions, Master of Applied Data Science Program*. Accessed on 30 March 2024 at https://docs.google.com/document/d/1A3zdTF0AYQY_zzD2-OlpSHeDxnWqFVEhXl446SyT_pA/.
6. Raffaele Mancuso (raffaem), “cs-dlp: Script for downloading Coursera.org videos and naming them,” *GitHub Website*. Accessed on 7 April 2024 at https://github.com/raffaem/cs-dlp.
7. LangChain, “Get started,” *LangChain Website*, 2024. Accessed on 30 March 2024 at https://python.langchain.com/docs/get_started.
8. Chroma, “Chroma” (documentation), *Chroma Website*. Accessed on 30 March 2024 at https://docs.trychroma.com/.
9. Future Data Systems (stanford-futuredata), “ColBERT: state-of-the-art neural search,” *GitHub Website*. Accessed on 7 April 2024 at https://github.com/stanford-futuredata/ColBERT.
10. Benjamin Clavié (bclavie), “RAGatouille” (documentation), *RAGatouille Website*. Accessed on 7 April 2024 at https://ben.clavie.eu/ragatouille/.
11. Andrei (abetlen), “Python bindings for llama.cpp,” *llama-cpp-python Documentation Website*. Accessed on 30 March 2024 at https://llama-cpp-python.readthedocs.io/en/latest/.
12. Mistral AI Team, “Mistral 7B,” *Mistral AI Website*, 30 September 2023. Accessed on 30 March 2024 at https://mistral.ai/news/announcing-mistral-7b/.
13. Greg Kamrat (gkamradt), “5 Levels Of Text Splitting,” *GitHub*. Accessed on 13 April 2024 at

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

14. Hugging Face, “all-MiniLM-L6-v2 Model Card,” *Hugging Face Website*. Accessed on 7 April 2024 at https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.
15. Snowflake, Inc, “A faster way to build and share data apps,” *Streamlit Website.* Accessed on 7 April 2024 at https://streamlit.io/.
16. Gradio, “Build & Share Delightful Machine Learning Apps,” *Gradio Website*. Accessed on 10 April 2024 at https://www.gradio.app/.
17. Aurelio AI, “Semantic Router: Deterministic Decision Making for AI,” *Aurelio AI Website.* Accessed on 10 April 2024 at https://www.aurelio.ai/semantic-router.
18. Chirag Shah and Emily M. Bender, “Situating Search,” In *Proceedings of the 2022 ACM SIGIR Conference on Human Information Interaction and Retrieval (CHIIR ’22)*, March 14–18, 2022, Regensburg, Germany. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3498366.3505816.

Appendix

A. Suggestions for Further Work

Here are a few items we would suggest for further work:

1. Additional research into semantic routing. With additional training, we think that this can be a powerful tool for honing the accuracy of results in an application like this. Being able to effectively differentiate between questions that can be answered best by syllabi versus those answered best by video lecture transcripts may be particularly helpful.
2. Integrating this pipeline with available University of Michigan resources. The university is making a U-M GPT Toolkit available that would provide API access to OpenAI (and other) models that are private. This would remove the need to run the LLM on local hardware while ensuring that the information processed remains protected within the university infrastructure.
3. Considering additional sources of information, such as discussions in the MADS Slack environment, particularly those in channels that are moderated by the staff. There would be some challenges in redacting student identities from that data.
4. Dockerizing the workflow for improved portability of the solution.