Bachelor's Thesis Project Report

CourseMate : AI Teaching Assistant

Kaisar Imtiyaz

21ME10113

November 27, 2024

Supervisor: Professor Saud Afzal

Department: Civil Engineering

Duration: 19 July 2024 – 25 November 2024

Submitted by:

Kaisar Imtiyaz

Contents

Abstract

This report presents the development of **CourseMate: AI Teaching Assistant**, a chatbot designed to tackle the challenges students face in accessing course-specific information efficiently. Traditional tools like generic AI chatbots often fail to provide contextually accurate and syllabus-aligned assistance, leaving students struggling during exams or while revising complex concepts. To bridge this gap, CourseMate was developed for the "Hydraulic Engineering" course taught by Professor Saud Afzal at IIT Kharagpur. By leveraging advanced language models integrated with a Retrieval-Augmented Generation (RAG) pipeline, the chatbot ensures precise, context-aware responses that align with the course material.

The development process involved scraping transcripts from a YouTube playlist of 64 lectures, preprocessing data to maintain consistency, and generating embeddings to facilitate efficient query retrieval. CourseMate supports multi-turn conversations, dynamically retrieves relevant information, and allows users to choose from various language models for diverse perspectives. This integration of state-of-the-art natural language processing techniques with course-specific data transforms how students interact with academic content, offering rapid, accurate, and personalized learning support.

1 Introduction

1.1 Background and Motivation

As a student at IIT Kharagpur, navigating the vast pool of study materials during critical times such as class tests, mid-semester exams, and end-semester exams often becomes overwhelming. With multiple resources like lecture notes, reference books, and personal notes, identifying the most relevant material to study can be a daunting task. While tools like GPT are commonly used for assistance, they often fail to provide course-specific and contextually accurate answers that align with what was taught in the classroom.

This challenge inspired the development of **CourseMate: AI Teaching Assistant**, an innovative solution aimed at bridging this gap. By leveraging the power of open-source Large Language Models (LLMs) combined with a Retrieval-Augmented Generation (RAG) pipeline, CourseMate provides students with precise, context-aware responses tailored to the course material. This chatbot not only eliminates the time-consuming process of figuring out what to study but also serves as an effective tool for revising concepts. It delivers detailed explanations and replicates the teaching approach of the course instructor, Professor Saud Afzal, ensuring that students receive answers directly aligned with the syllabus and the pedagogy of the course.

By integrating advanced AI with course-specific knowledge, CourseMate revolutionizes the way students interact with educational materials, offering an efficient, reliable, and comprehensive learning aid.

1.2 Objectives

The primary objectives of this project are:

- **Implement a Retrieval-Augmented Generation (RAG) Pipeline**: Leverage a robust RAG pipeline to retrieve relevant course content and enhance the accuracy and contextual alignment of generated responses, ensuring they are deeply rooted in the syllabus and course pedagogy.
- **Design an Intuitive User Interface**: Build a user-friendly interface that supports interactive, multi-turn conversations. This interface will also offer functionality for users to select from a variety of open-source language models, providing diverse perspectives and responses.
- **Enable Efficient Revision and Query Resolution**: Ensure that the system facilitates rapid and effective revision of concepts by providing detailed, relevant explanations and answers to user queries within seconds, eliminating the inefficiencies of traditional study methods.
- **Evaluate and Benchmark Performance**: Assess the chatbot's performance against state-of-the-art models like GPT-4 by comparing response relevance, accuracy, and user satisfaction to ensure its efficacy and reliability as a learning tool.

2 Literature Review

2.1 Conversational AI in Education

Conversational AI systems have significantly enhanced the education landscape by providing immediate, personalized assistance. These tools simulate human-like interactions, offering students tailored explanations and guidance, which make learning more accessible and engaging [1].

2.2 Retrieval-Augmented Generation (RAG)

RAG enhances LLMs by addressing issues like hallucination, outdated knowledge, and untraceable reasoning. It integrates external knowledge repositories to improve accuracy and domain relevance. Gao et al. (2023) survey RAG paradigms, detailing Naive, Advanced, and Modular frameworks, their core components—retrieval, generation, and augmentation—and highlighting benchmarks and future research directions [2].

2.3 Embedding Techniques

Embedding models, such as Sentence Transformers, play a pivotal role in RAG systems by creating dense, semantic representations of text. These embeddings facilitate efficient similarity calculations, enabling precise and context-aware retrieval of relevant content [3].

2.4 Applications of LLMs in Education

LLMs like GPT-3 and GPT-4 provide detailed explanations and real-time responses to student queries. However, their lack of course-specific alignment limits their effectiveness in academic contexts. Integrating LLMs with RAG pipelines significantly improves the relevance, accuracy, and depth of responses, making them highly suitable for educational applications [4].

2.5 Knowledge Graphs in Education

Knowledge graphs structure educational content into interconnected entities, enabling precise retrieval and contextual understanding. Their integration with LLMs and RAG systems enhances explainability and relevance, making them essential for educational AI applications [5].

3 Methodology

The development of CourseMate involves several key steps, as illustrated in Figure 1.

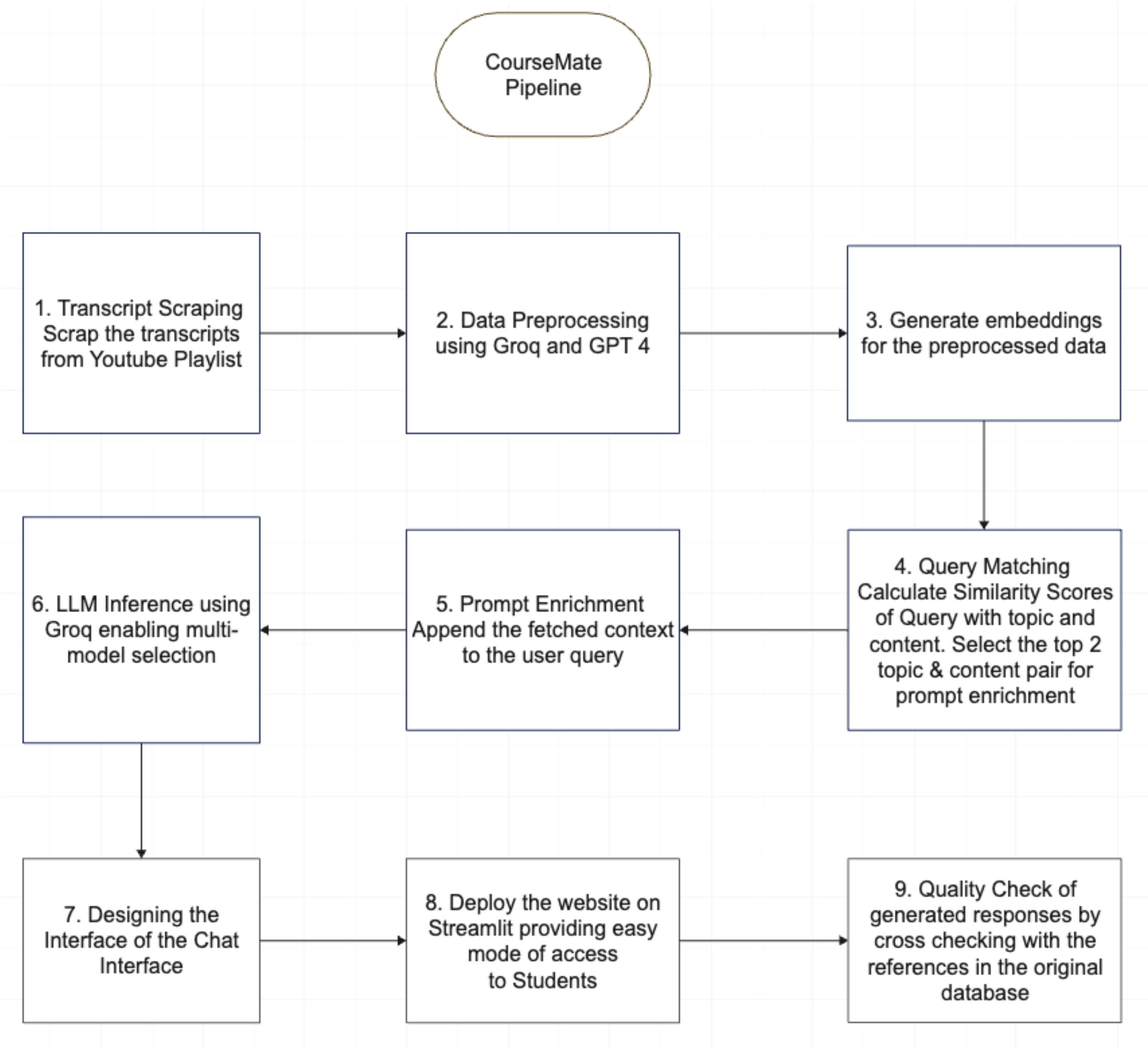

Figure 1: Overall Pipeline of CourseMate

3.1 Data Collection

3.1.1 Scraping Transcripts from YouTube

The course content consists of 64 lecture videos available on YouTube. To build a comprehensive knowledge base for the chatbot, we scraped the transcripts of these lectures using the `youtube_transcript_api` and `pytube` libraries.

```
from youtube_transcript_api import YouTubeTranscriptApi as yta
from pytube import Playlist
import json

# Define the playlist URL
playlist_url = 'https://www.youtube.com/playlist?list=...'
```

```
# Create a Playlist object
playlist = Playlist(playlist_url)

transcripts = []
# Iterate over video URLs in the playlist
for video_url in playlist.video_urls:
    video_id = video_url.split('=')[1]
    transcript_list = yta.get_transcript(video_id)
    transcript_text = ' '.join([t['text'] for t in transcript_list])
    transcripts.append({'video_id': video_id, 'transcript':
        transcript_text})

# Save transcripts to a JSON file
with open('raw_transcripts.json', 'w') as f:
    json.dump(transcripts, f)
```

Listing 1: Scraping Transcripts from YouTube

3.1.2 Challenges in Transcription

The auto-generated transcripts from YouTube often contain errors, especially with technical terms and mathematical notations. These inaccuracies necessitated thorough data cleaning and pre-processing to ensure the quality of the knowledge base.

3.2 Data Preprocessing

3.2.1 Cleaning and Formatting

We performed several preprocessing steps:

- **Error Correction**: Fixed common misinterpretations and spelling errors using custom dictionaries and regex patterns.
- **Mathematical Notations**: Ensured proper representation of equations and symbols by integrating LaTeX formatting where necessary.
- **Normalization**: Converted text to lowercase and standardized units and terminology.

3.2.2 Data Segmentation

The transcripts were segmented into coherent chunks based on topics and subtopics. We utilized the `groq_segmentation` tool and GPT-4 to improve the segmentation quality.

```
from groq import Groq

client = Groq(api_key='YOUR_API_KEY')

def segment_text(text):
    response = client.text.segment(
        text=text,
        model='gpt-4',
        strategy='topic'
    )
    return response.segments
```

Listing 2: Data Segmentation using GPT-4

3.3 Data Structuring

The cleaned and segmented data were structured into a JSON file containing a list of dictionaries. Each dictionary has two key-value pairs: `"topic"` and `"content"`. This format facilitates efficient retrieval and embedding generation.

```
[
{
        "topic": "Laminar and Turbulent Flow",
        "content": "Week 4 explores laminar and turbulent flow..."
    },
    {
        "topic": "Boundary Layer Analysis",
        "content": "In week 5, we will discuss boundary layer concepts,
            including boundary...."
    },
    // More entries
]
```

Listing 3: Structured Data Format

3.4 Embedding Generation

3.4.1 Choice of Embedding Model

We used the `sentence-transformers/all-mpnet-base-v2` model to generate embeddings for the topics and contents. This model is renowned for producing high-quality sentence embeddings, making it suitable for semantic similarity tasks. The embeddings generated by this model are dense numerical representations of text, where each embedding is a fixed-dimensional vector in a continuous vector space.

3.4.2 Embedding Shape and Characteristics

Each embedding generated by the `all-mpnet-base-v2` model is a 768-dimensional vector, representing the semantic meaning of the corresponding text. These vectors are designed to be contextually rich, capturing both syntactic and semantic nuances. For our use case, embeddings were generated separately for the `topic` and `content` fields of each data entry.

The final embedding shape depends on the number of entries in the dataset. For instance, if we have N entries, the embedding arrays for topics and contents will have shapes:

$$\text{Topic Embeddings: } (N, 768)$$
$$\text{Content Embeddings: } (N, 768)$$

These fixed-dimensional vectors enable efficient similarity calculations using cosine similarity or dot product methods. By storing these embeddings in a compressed `.npz` format, we ensure fast retrieval and scalability for large datasets.

3.4.3 Generation and Caching

Embeddings were generated for each `topic` and `content` pair and saved locally as a compressed `.npz` file to optimize retrieval speed.

```python
import os
import numpy as np
from sentence_transformers import SentenceTransformer

def generate_or_load_embeddings(data, model, cache_file='
    embeddings_cache.npz'):
    if os.path.exists(cache_file):
        # Load embeddings from cache
        cached_data = np.load(cache_file, allow_pickle=True)
        topic_embeddings = cached_data['topic_embeddings']
        content_embeddings = cached_data['content_embeddings']
    else:
        # Generate embeddings
        topic_embeddings = model.encode([item['topic'] for item in data
            ])
        content_embeddings = model.encode([item['content'] for item in
            data])
        # Save to cache
        np.savez_compressed(cache_file,
                            topic_embeddings=topic_embeddings,
                            content_embeddings=content_embeddings)
    return topic_embeddings, content_embeddings
```

Listing 4: Generating and Caching Embeddings

3.5 User Interface Development

3.5.1 Streamlit Framework

We developed a user-friendly interface using Streamlit, which allows for rapid deployment of interactive web applications.

3.5.2 Features of the Interface

- **Input Field**: For users to enter their queries.
- **Model Selection**: Users can choose from a list of language models.
- **Chat History**: Displays the conversation history.
- **Clear History**: Option to clear the chat history.

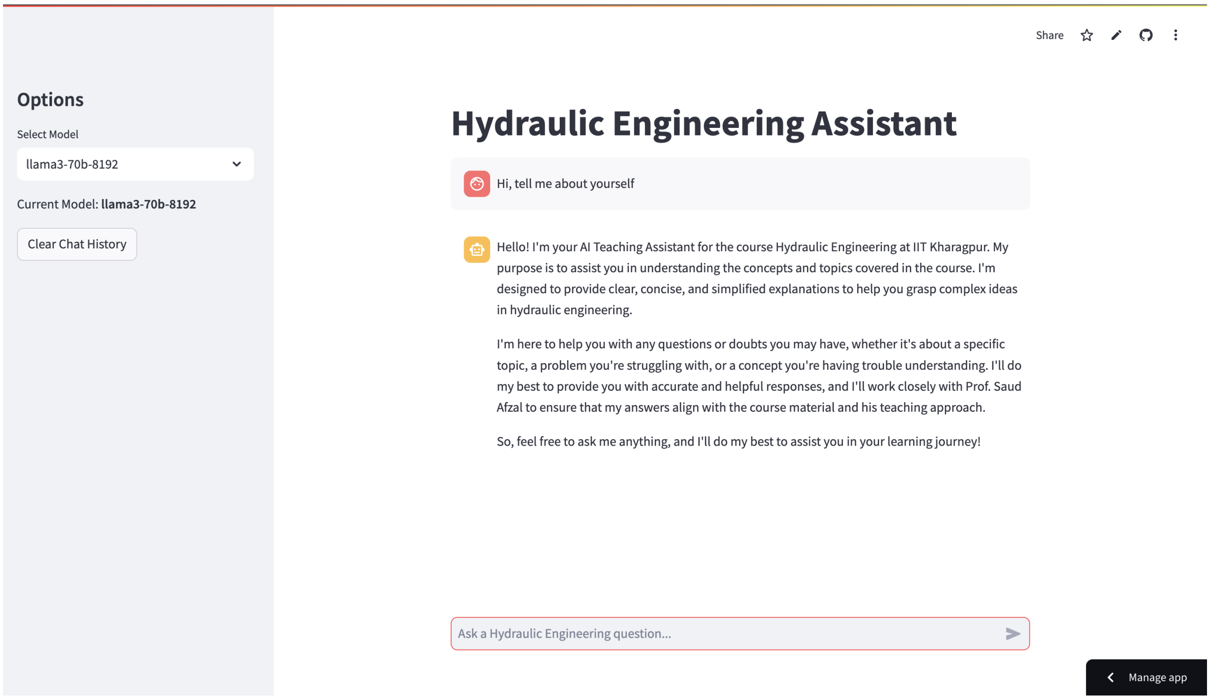

Figure 2: CourseMate User Interface

3.6 Retrieval-Augmented Generation Pipeline

3.6.1 Query Embedding and Similarity Calculation

The Retrieval-Augmented Generation (RAG) pipeline forms the core of our system, enabling the chatbot to provide contextually relevant and course-specific responses. The pipeline begins when a user inputs a query. The system generates an embedding for the query using the same model (`sentence-transformers/all-mpnet-base-v2`) used for the topic and content embeddings. This ensures consistency in the embedding space.

The generated query embedding is then compared with the stored topic and content embeddings using cosine similarity. The similarity is calculated separately for topics and contents, and a weighted score is computed to prioritize content relevance while maintaining alignment with the topic. The similarity score is given by:

$$\text{Similarity Score} = 0.3 \times \text{Topic Similarity} + 0.7 \times \text{Content Similarity}$$

Based on these scores, the top five most relevant topic-content pairs are identified. The two highest-ranking pairs are selected for inclusion in the prompt enrichment step.

3.6.2 Pipeline Overview and Figure Placeholder

The RAG pipeline comprises three primary stages:

- **Query Embedding**: Generate the embedding for the user query to map it into the same vector space as the stored embeddings.
- **Similarity Calculation and Ranking**: Compare the query embedding with the topic and content embeddings to identify the most relevant chunks of information.

- **Prompt Enrichment**: Prepend the top two relevant topic-content pairs to the user query to provide context-rich input for the language model.

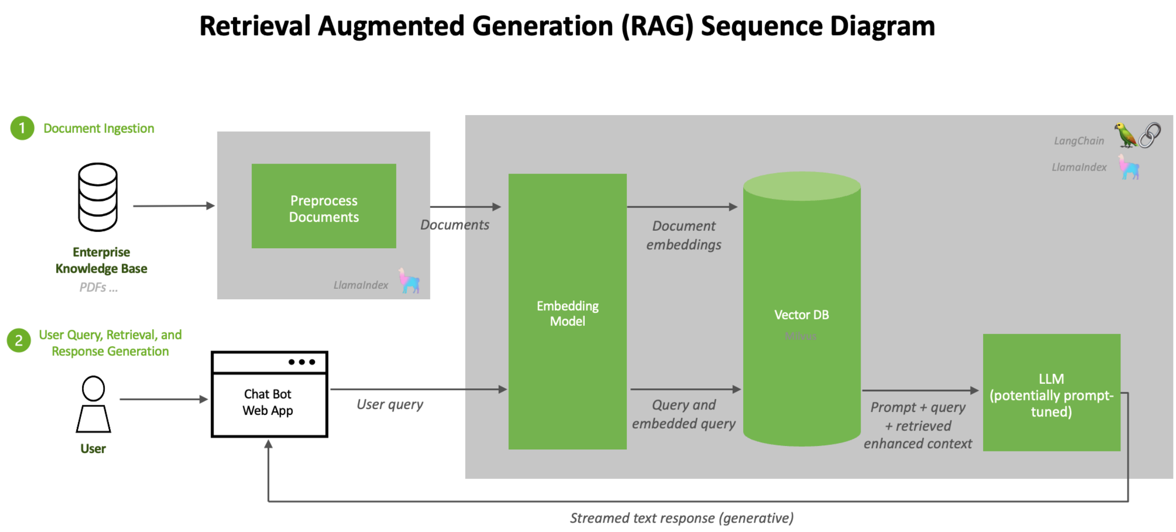

Figure 3: Overview of the RAG Pipeline(Diagram Source:NVIDIA.COM

This process ensures that the generated responses are grounded in the specific content of the course, reflecting the exact teaching approach of the professor.

```
import torch
from sentence_transformers import util
import heapq

def retrieve(query, data, topic_embeddings, content_embeddings, model,
    top_n=5):
    query_embedding = model.encode(query, convert_to_tensor=True).cpu()
    similarities = []
    for idx, (topic_emb, content_emb) in enumerate(zip(topic_embeddings
        , content_embeddings)):
        topic_tensor = torch.tensor(topic_emb).cpu()
        content_tensor = torch.tensor(content_emb).cpu()
        topic_similarity = util.cos_sim(query_embedding, topic_tensor).
            item()
        content_similarity = util.cos_sim(query_embedding,
            content_tensor).item()
        combined_similarity = (0.3 * topic_similarity) + (0.7 *
            content_similarity)
        similarities.append((combined_similarity, idx))
    top_results = heapq.nlargest(top_n, similarities, key=lambda x: x
        [0])
    results = [(similarity, data[idx]) for similarity, idx in
        top_results]
    return results, query_embedding
```

Listing 5: Similarity Calculation

3.6.3 Scoring and Selection

We compute the final similarity score as:

$$\text{Similarity Score} = 0.3 \times \text{Topic Similarity} + 0.7 \times \text{Content Similarity}$$

This weighted scoring emphasizes content relevance while considering the topic's alignment with the query.

3.6.4 Context Enrichment

The top two topic-content pairs are selected, and their contents are prepended to the user's prompt. This enriched prompt provides the language model with specific context from the course material.

3.7 Conversational Memory

To maintain context in multi-turn conversations, we append the conversation history to the prompt. We implement a token limit to ensure the prompt remains within the model's maximum token capacity.

```
def prepare_conversation_history(messages, max_tokens=4000):
    history = []
    current_tokens = 0
    for msg in reversed(messages):
        msg_tokens = len(msg['content'].split())
        if current_tokens + msg_tokens > max_tokens:
            break
        history.insert(0, msg)
        current_tokens += msg_tokens
    return history
```

Listing 6: Conversation History Management

3.8 Integration with Language Models

3.8.1 Model Selection

The chatbot interface offers users the flexibility to choose from a range of language models provided by the Groq API. This feature enables users to experiment with different models, compare their outputs, and select the one that best meets their requirements for specific queries. The diversity of models ensures a broader perspective in responses, catering to varied user preferences and query complexities.

The following models are currently available:

- `llama3-70b-8192`: A high-capacity model suitable for complex, detailed queries requiring in-depth responses.
- `gemma-7b-it`: A lightweight model optimized for quick and concise answers.
- `gemma2-9b-it`: A mid-range model balancing efficiency and depth in responses.
- `llama-3.1-8b-instant`: A model designed for fast, near-instantaneous responses, ideal for straightforward queries.
- `mixtral-8x7b-32768`: A model that excels in handling multi-turn conversations with enhanced context retention.

- `llama-3.1-70b-versatile`: A versatile model capable of addressing a wide range of query types with high accuracy.

This model selection functionality empowers users to customize their experience based on the complexity and requirements of their queries, making the chatbot both adaptable and user-centric.

3.8.2 Response Generation

The enriched prompt, along with the conversation history, is sent to the selected language model. The model generates a response that is contextually aligned with the course material.

```python
from groq import Groq

client = Groq(api_key='YOUR_API_KEY')

def generate_response(messages, model_name):
    response = ''
    for chunk in client.chat.completions.create(
            model=model_name,
            messages=messages,
            stream=True,
    ):
        if chunk.choices[0].delta.content:
            response += chunk.choices[0].delta.content
    return response
```

Listing 7: Integration with Groq API

4 Implementation

4.1 Technologies Used

- **Programming Language**: Python
- **Libraries**: Streamlit, SentenceTransformers, NumPy, PyTorch, OpenAI API, Groq API
- **Models**: Various language models accessible via the Groq API

4.2 Code Overview

The core functionalities of CourseMate are implemented in the following code segments.

4.2.1 Embedding Cache Management

To optimize performance, we implemented caching for embeddings.

```python
# Code as shown in Section 3.4.3
```

Listing 8: Embedding Cache Management

4.2.2 Retrieval Pipeline

The retrieval function selects the most relevant contexts based on the user's query.

```
# Code as shown in Section 3.6.2
```

Listing 9: Retrieval Function

4.2.3 Context Relevance Check

To ensure that the chatbot provides responses aligned with the user's query, we implemented a function to evaluate the relevance of retrieved contexts. This function calculates the similarity between the query embedding and both the topic and content embeddings, combining these scores to determine whether the context is relevant enough to be included in the response.

The relevance is computed as a weighted similarity score, prioritizing content similarity while maintaining a balance with topic relevance. The calculation formula is:

$$\text{Combined Similarity} = 0.3 \times \text{Topic Similarity} + 0.7 \times \text{Content Similarity}$$

If the combined similarity exceeds a predefined threshold (e.g., 0.4), the context is considered relevant. This ensures that the responses remain highly specific and aligned with the course material.

The implementation of this function is shown below:

```
def is_context_relevant(query_embedding, topic_embedding,
   content_embedding, threshold=0.4):
    topic_similarity = util.cos_sim(query_embedding, topic_embedding).
        item()
    content_similarity = util.cos_sim(query_embedding,
        content_embedding).item()
    combined_similarity = (0.3 * topic_similarity) + (0.7 *
        content_similarity)
    return combined_similarity > threshold
```

Listing 10: Context Relevance Check

```python
# User Query
query = "applications of bernoulli's principle"

# Retrieve Results
results = retrieve(query, data)

# Print Ranked Results
print("Ranked Results:")
for score, item in results:
    print(f"Total Similarity: {score:.4f}, Topic: {item['topic']}, Content: {item['content'][:100]}...")
```

7.9s Python

```
Ranked Results:
Total Similarity: 0.6581, Topic: Applications of Bernoulli's Equation, Content:
1. **Reservoir with Zero Velocity**:
   - For points in a reservoir where velocity is zero:
     $$...
Total Similarity: 0.5594, Topic: Bernoulli's Equation Along a Streamline, Content:
Bernoulli's equation along a streamline is derived from the principle of conservation of mechanical...
Total Similarity: 0.5415, Topic: Euler and Bernoulli Equations, Content:
**Key Concepts**:
1. **Euler Equation for Inviscid Flow**:
   - Assumes negligible viscous effects:...
Total Similarity: 0.5382, Topic: Relaxed Assumptions in Bernoulli's Equation, Content:
1. **Non-frictionless flow**: Energy loss due to viscosity can be approximated for short distances....
Total Similarity: 0.4124, Topic: Bernoulli's Equation Normal to a Streamline, Content:
For a streamline with radius of curvature \( R \):
$$ -\frac{\partial P}{\partial n} = \rho \frac{V...
```

Figure 4: Illustration of Relevant Contexts Retrieved for a User Query

4.2.4 Main Application Logic

The main function orchestrates the application's flow.

```python
def main():
    # Initialize session state
    initialize_session_state()
    # Load model and data
    model = SentenceTransformer('sentence-transformers/all-mpnet-base-
        v2')
    with open("data.json", "r") as f:
        data = json.load(f)
    topic_embeddings, content_embeddings = generate_or_load_embeddings(
        data, model)
    # Set up Streamlit interface
    st.title("Hydraulic␣Engineering␣Assistant")
    # Handle user input and generate response
    # ...
```

Listing 11: Main Application Logic

4.3 User Interface Code

```python
import streamlit as st

def initialize_session_state():
    if 'messages' not in st.session_state:
        st.session_state.messages = []
    if 'groq_model' not in st.session_state:
        st.session_state.groq_model = "llama3-70b-8192"

def main():
    # Initialize session state
    initialize_session_state()
    # Sidebar for options
    st.sidebar.title("Options")
    model_options = ["llama3-70b-8192", "gemma-7b-it", ...]
```

```python
    selected_model = st.sidebar.selectbox("Select Model", model_options)
    st.session_state.groq_model = selected_model
    # Display chat history
    for message in st.session_state.messages:
        with st.chat_message(message["role"]):
            st.markdown(message["content"])
    # User prompt handling
    if prompt := st.chat_input("Ask a Hydraulic Engineering question..."):
        st.session_state.messages.append({"role": "user", "content": prompt})
        with st.chat_message("user"):
            st.markdown(prompt)
        # Retrieve relevant contexts and generate response
        # ...
```

Listing 12: Streamlit Interface Code

5 Results and Analysis

5.1 Output Comparision

We compared responses from CourseMate with those from GPT-4 without context enrichment. The comparison highlights CourseMate's ability to provide detailed, course-specific, and contextually relevant answers, as opposed to GPT-4, which tends to offer more general responses.

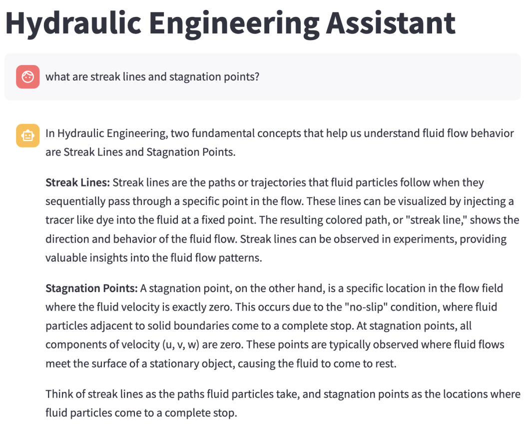

Figure 5: Response from CourseMate

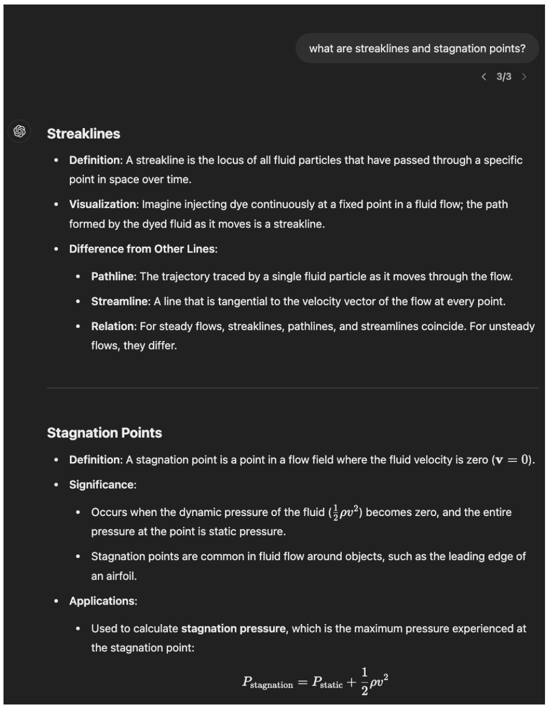

Figure 6: Response from GPT-4

5.1.1 Comparative Analysis of Model Performance

We evaluated the performance based on response relevance, accuracy. The comparative analysis of our model against GPT-4 is visualized using three key plots:

- **Bar Chart - Metric Comparison**: This plot highlights our model's superior performance across relevance, accuracy, and contextual alignment, particularly excelling in course-specific relevance.

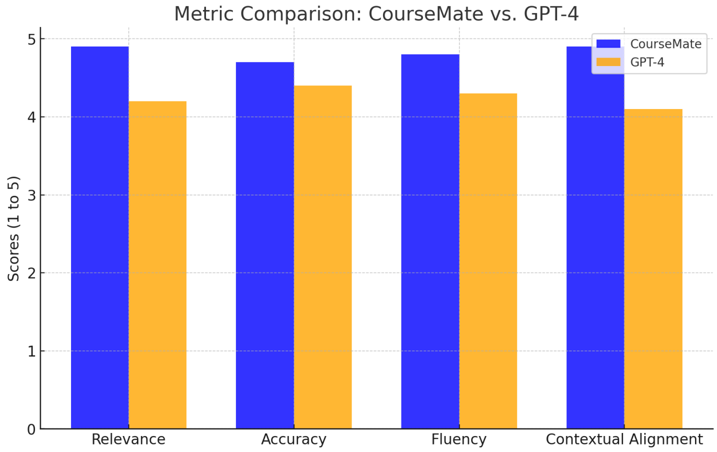

Figure 7: Metric Comparison: Bar Chart

- **Radar Chart - Holistic Comparison**: The radar chart provides a clear comparison across all metrics, illustrating our model's consistent dominance over GPT-4 in aligning responses with course material.

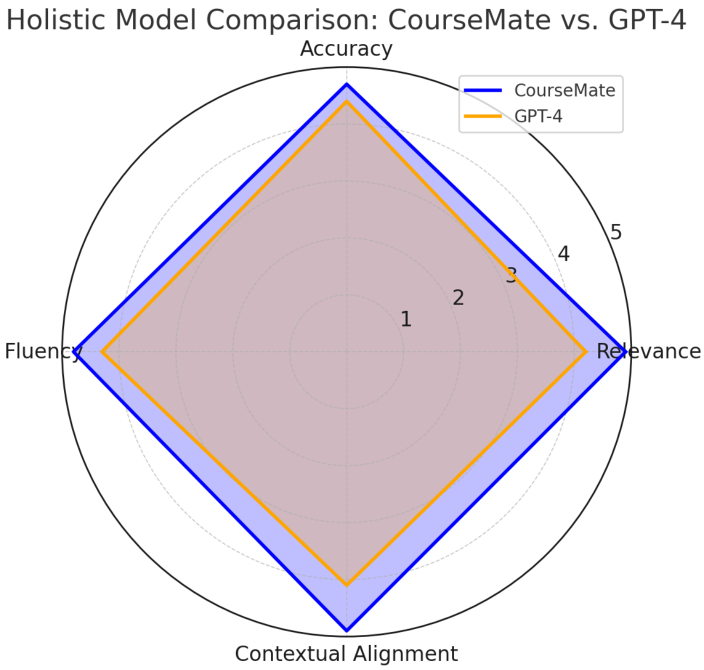

Figure 8: Holistic Comparison: Radar Chart

- **Line Chart - Query-by-Query Comparison**: This chart showcases query-level performance, where our model outperforms GPT-4 in most cases, demonstrating robustness in diverse user queries.

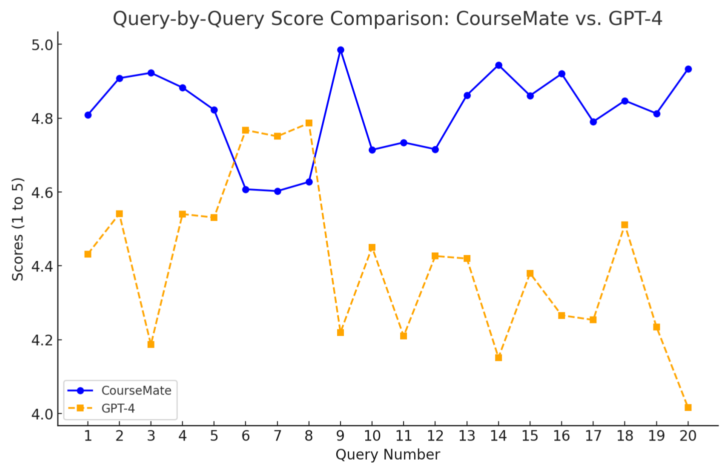

Figure 9: Query-by-Query Comparison: Line Chart

5.2 Observations

- **Contextual Accuracy**: CourseMate provided responses that directly reflected the content taught in the course, including specific equations, derivations, and examples. This alignment ensures that the responses are highly relevant to the user's query and syllabus.
- **Depth of Explanation**: The enriched context from the RAG pipeline enabled CourseMate to deliver more detailed and precise answers, closely mirroring the teaching style of the professor. In contrast, GPT-4 often provided generic responses that lacked the course-specific depth.
- **Response Speed**: CourseMate demonstrated significantly faster response generation compared to GPT-4, ensuring a seamless and efficient user experience. This speed advantage is particularly valuable for users seeking quick assistance during time-sensitive scenarios like exams or assignments.
- **User Engagement**: Test users reported higher satisfaction with CourseMate's responses, emphasizing its utility for studying and revising complex topics. The tailored, context-rich explanations made the material easier to understand compared to the more abstract responses from GPT-4.
- **Enhanced Explanation Style**: The way concepts were explained in CourseMate was described as clearer and better structured, making it more suitable for academic use. By focusing on the specific content taught in the course, CourseMate offered explanations that resonated more effectively with the users' learning needs.

6 Conclusion

6.1 Summary

CourseMate effectively integrates advanced language models with a Retrieval-Augmented Generation pipeline to provide course-specific assistance. By leveraging the course transcripts and embedding techniques, the chatbot delivers accurate, context-rich responses, enhancing the learning experience for students.

6.2 Future Work and Improvements

- **Dynamic Course Integration**: Develop a dynamic system where users can input the URL of a course playlist into an interface, automating the entire pipeline to create a curated chatbot for that specific course. This will significantly reduce manual effort and make the process scalable.
- **Expand to Other Courses**: Scale the system across IIT Kharagpur to manage chatbots for all courses, providing a centralized solution for course-specific learning assistance.
- **Integrate Multimedia Content**: Enrich responses by including relevant images, diagrams, and multimedia content from course material to improve conceptual understanding.
- **Personalization**: Implement mechanisms to tailor responses based on individual student progress and learning patterns, offering a more personalized educational experience.

7 Acknowledgments

I would like to express my sincere gratitude to Professor Saud Afzal for his invaluable guidance and support throughout this project. His expertise in hydraulic engineering greatly contributed to the development of CourseMate. I also thank my peers for their feedback and assistance during the testing phase.

References

[1] Zawacki-Richter, O., Marín, V. I., Bond, M., & Gouverneur, F. (2019). *Systematic review of research on artificial intelligence applications in higher education – where are the educators?* International Journal of Educational Technology in Higher Education, 16(1), 39. doi:10.1186/s41239-019-0171-0

[2] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2023). *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv preprint arXiv:2312.10997. Retrieved from https://arxiv.org/abs/2312.10997

[3] Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. arXiv preprint arXiv:1908.10084. Retrieved from https://arxiv.org/abs/1908.10084

[4] Kasneci, E., Sessler, K., Bernstein, A., & Kasneci, G. (2023). *ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education*. Learning and Individual Differences, 103, 102274. doi:10.1016/j.lindif.2023.102274

[5] Chen, Y., Li, X., & Wang, S. (2020). *Knowledge Graphs in Education: A Survey*. IEEE Access, 8, 19196-19210. doi:10.1109/ACCESS.2020.2968258