Customizing Large Language Models for Automated Academic Advising at Universities

Ronit Banerjee, Kathryn Butziger, Jose Fabrizio Filizzola Ortiz, Matthew Kiszla

WORCESTER POLYTECHNIC INSTITUTE

Advisor:

Xiangnan Kong

A Major Qualifying Project
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for
the Degree of Bachelor of Science in Computer
Science and Data Science.

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on the web without editorial or peer review.

Abstract

In this report, we explore the customization of Large Language Models (LLMs) for automated academic advising at universities, with a focus on streamlining access to the constantly evolving academic information at Worcester Polytechnic Institute (WPI). This includes course details, room bookings, and updates to tracking sheets. To overcome LLM knowledge gaps, we built customized LLM solutions through three distinct pipelines: one utilizing the BGE embedding model with Pinecone vector database for context mapping, one employing Pinecone's Retrieval-Augmented Generation (RAG) model for efficient data retrieval, and one integrating OpenAI's Assistants API for precise query responses. In an experiment with 150 WPI academic advising questions, each of these pipelines achieved promising results compared to almost all incorrect answers from ChatGPT without context. Through our research, we not only built a tailored Q&A system that tackles the knowledge gap at WPI but also showcased the potential for LLMs to be customized for similar challenges in other institutions.

Contents

1 Introduction 1

2 Background 3

2.1 A Brief History of Chatbots 3
2.2 Prompt Engineering 4
2.3 Current Limitations of LLMs 6
2.4 Retrieval Augmented Generation and Related Works 9
2.5 Text Embeddings 10
2.6 Pre-trained embedding models 11
2.7 Vector Database 12
2.8 Pinecone Vector Database 14

3 Methodology 16

3.1 Dataset 16
  3.1.1 Collection of Data 17
  3.1.2 Data Cleaning and Modification 19
  3.1.3 Automation and Updating Data 24
  3.1.4 Data Storage 25
  3.1.5 Data Limitations 25
3.2 Tools Used 26
  3.2.1 Versions of GPT 26
  3.2.2 Coding Tools 26
3.3 Pipeline #1: RAG Model with LangChain Pinecone and BGE embedding Model 27
  3.3.1 LangChain Usage 27
  3.3.2 BGE Usage 27
  3.3.3 Pinecone Usage 28
3.4 Pipeline #2: RAG Model with Pinecone Canopy 29
  3.4.2 Key Differences 30
3.5 Pipeline #3: OpenAI Assistants API with Question Classifier 30
  3.5.1 Assistant Generation 30
  3.5.2 Question Classifier 32

3.5.3 Key Differences......33
3.6 Experiment Methodology......33
3.6.1 Data Generation for Questions...... 34
3.6.2 Experiment Format...... 34
3.6.3 Experiment Evaluation Method...... 34
3.7 Integration with User Interface...... 34

4 Experiments and Results **36**
4.1 Pipeline #1 Results...... 36
4.1.1 Time and Cost to Run Experiment for Pipeline #1...... 36
4.1.2 Pipeline#1 Answer Results...... 36
4.1.3 Example of Incorrect Answer Pipeline #1......37
4.2 Pipeline #2 Results:...... 38
4.2.1 Time and Cost to Run Experiment for Pipeline #2...... 38
4.2.2 Pipeline #2 Answer Results...... 38
4.2.3 Example of Incorrect Answer Pipeline #2......39
4.3 Pipeline #3 Results:...... 40
4.3.1 Time and Cost to Run Experiment for Pipeline #3...... 40
4.3.2 Pipeline #3 Answer Results...... 40
4.3.3 Example of Incorrect Answer Pipeline #3......41
4.4 Overall Results......41
4.4.1 Comparison Between Pipelines and Base Model without Context......42
4.4.2 Example of Questions All Pipelines Responded Correctly......45
4.5 Frontend/Backend Results:......48

5 Discussion **49**

6 Conclusion and Future Work **50**

A GitHub Repository **54**

List of Tables

Table 1: Number of parameters vs. estimated training costs for GPT models [10]....................7
Table 2: Info on webpages gathered.. 18
Table 3: Key Resources for Data Gathering.. 19
Table 4: Token Comparison of Collected Data for Different Formats..................................... 25
Table 5: Cost and Time for Pipeline#1 Experiment.. 37
Table 6: Scoring Results for Pipeline #1... 37
Table 7: Time Results Pipeline #1...38
Table 8: Cost and Time for Pipeline #2 Experiment...39
Table 9: Scoring Results for Pipeline #2... 39
Table 10: Time Results Pipeline #2.. 40
Table 11: Cost and Time for Pipeline #3 Experiment... 41
Table 12: Scoring Results for Pipeline #3... 41
Table 13: Time Results Pipeline #3.. 42

List of Figures

Figure 1: A typical prompt given to ChatGPT 3.5 Turbo [12].. 8
Figure 2: An engineered prompt given to ChatGPT [12].. 9
Figure 3: Neural Network architecture for getting sentence embeddings [17]........................ 15
Figure 4: The flow of how vector databases work [22].. 17
Figure 5: The flow of how Canopy implements a RAG application [24]........................ 19
Figure 6: A portion of WPI sitemap.. 21
Figure 7: Breakdown of WPI pages gathered.. 23
Figure 8: JSON Data for Elementary Arabic Course.. 24
Figure 9: Natural Language Data for Elementary Arabic Course.. 24
Figure 10: ChatGPT Response to JSON Format [12].. 25
Figure 11: ChatGPT Response to Natural Language Format [12].. 25
Figure 12: Similarity score for natural language format of the user question "Can you tell me about the course Team and Leadership Fundamentals III?".. 26
Figure 13: Similarity score for structured format of the user question "Can you tell me about the course Team and Leadership Fundamentals III?".. 27
Figure 14: Data Updating Process.. 29
Figure 15: Usage Flow of LangChainTool.. 31
Figure 16: Usage Flow of LangChain Tool.. 32
Figure 17: Pipeline #1 Visual.. 33
Figure 18: Canopy Pipeline Visual.. 34
Figure 19: OpenAI Assistant Example.. 35
Figure 20: OpenAI Assistant Example.. 36
Figure 21: Classifier Flow for User Question.. 37
Figure 22: Example Classifier using an OpenAI Assistant.. 37
Figure 23: Quality Score Breakdown of Pipelines vs Base Model without Context................ 46
Figure 24: Relevance Score Breakdown of Pipelines vs Base Model without Context............ 47
Figure 25: Satisfaction Score Breakdown of Pipelines vs Base Model without Context........... 48
Figure 26: Answer Time Breakdown of Pipelines vs Base Model without Context................ 49
Figure 27: Working Implementation of WPI Chatbot using Pipeline # 3.............................. 52

1 Introduction

In recent years, the advancement of Large Language Models (LLMs) such as OpenAI's ChatGPT [1] and Google's BERT [2] has transformed our interaction with digital platforms. Powered by sophisticated machine learning algorithms, these models excel in producing text that mirrors human writing, adeptly handling intricate queries, and even programming software. Their widespread utilization across diverse sectors highlights their capability to enhance services in customer support, education, and healthcare, among others [3]. This technological leap demonstrates not only the versatility of LLMs in addressing complex informational and computational tasks, but also their role in streamlining processes and improving information accessibility and efficiency in many industries.

Despite their capabilities, LLMs encounter notable challenges, especially when it comes to incorporating the latest information and providing precise responses to detailed queries. This difficulty arises from their dependence on static datasets for training, limiting their knowledge to what was available before their most recent update [4]. Such limitations are particularly problematic in dynamic environments that require up-to-date and specialized knowledge. A prime example of this is at higher education institutions such as WPI, where the constant evolution of course content, room schedules, and degree requirements necessitates access to the most current academic advising information. For the WPI community, navigating these complexities is a frequent reality, demonstrating the need for a simpler, more adaptable way to get information.

In an attempt to bridge this knowledge gap, some models have turned to internet access or search engine integration as a workaround, enabling the LLM to "look up" the latest information. Innovations like Google's Gemini (formerly known as Bard) [5] and Microsoft's Bing Chat [6] exemplify this approach, aiming to bolster their models' ability to retrieve real-time data and thereby improve responsiveness and accuracy. Nonetheless, this strategy comes with its own set of challenges. The reliability of search engine-based updates is limited by the search engines themselves, which may return results that are not entirely relevant, up-to-date, or correct. Furthermore, parsing this externally sourced information accurately in response to a user's specific query presents an additional layer of complexity. This reliance on search engines also means depending on technology that might not be fully aligned with the nuanced informational needs of specific environments like WPI.

Our project addresses these issues head-on by developing a custom LLM solution designed to meet the unique academic advising demands of WPI, thereby enhancing the model's capacity to deliver timely and relevant information specific to the institution. By integrating a dynamic dataset that captures the latest updates on course offerings, room scheduling, and other critical information at WPI, we aim not just to bridge the knowledge gap experienced by the WPI community but also to showcase the potential for LLMs to be customized for similar challenges in other institutions. This approach directly tackles the complexities faced by WPI staff and students, providing a targeted solution that bypasses the limitations of external search engines and ensures the LLM remains a valuable and accurate resource for the community.

In tackling this challenge, our methodology integrated three approaches to refine the process of directing data to an LLM for generating accurate and relevant responses. The first method involved converting WPI's extensive documentation into vectors through a text embedding model, creating links between users' WPI-related queries and the pertinent information. These vectors were then utilized by a GPT model to craft appropriate responses, enabling a nuanced understanding of the query content for accurate information retrieval. The second method employed Pinecone Canopy, a RAG (Retriever-Augmented Generation) model builder, offering a more automated and efficient way to perform tasks similar to our initial text embedding model. This approach streamlined the process of linking queries with the corresponding information, aiming to simplify and enhance the efficiency of embedding and retrieving relevant data. Lastly, for our third and final method, we explored using OpenAI's Assistant API in conjunction with a knowledge retrieval tool to develop several specialized chatbots. Through this, a classifier analyzed the user's question and directed it to the most suitable chatbot, ensuring that the response is as accurate as possible. This multi-chatbot strategy ensures tailored responses to the specific nature of each inquiry, leveraging specialized knowledge bases for optimal accuracy.

Through these methods, our project explores how to build a tailored Q&A system that tackles the knowledge gap at WPI. This effort not only shows how well LLMs can fulfill precise information needs but also highlights the importance of machine learning progress in dealing with today's digital challenges. By incorporating up-to-date, localized information into LLMs, our approach points towards a future where information sharing becomes more personalized and accurate across different areas.

2 Background

This section provides the information that we referenced and based the project on, with detailed context on chatbots, LLMs, and their associated fields.

2.1 A Brief History of Chatbots

To understand the recent popularity of chatbots, we must first understand their history, categories, and applications. Chatbots are computer programs that simulate human conversations, which allows the user to interact more easily with the computer [7]. These chatbots can range in complexity from simple keyword matching to using Natural Language Processing (NLP) and machine learning (ML) techniques. The history of chatbots was marked by the available knowledge and technology at the time to develop them.

In 1950, Alan Turing created the Turing Test, where a computer has to simulate a human conversation with a user and the user cannot distinguish if it is a human or a computer. This idea made people start thinking about chatbots and possible applications. Joseph Weizenbaum's ELIZA was the first chatbot that helped users with daily tasks by retrieving useful information like weather, stocks, and sports. These first chatbots did not understand human language but used keyword matching and other basic techniques to respond to the users [7].

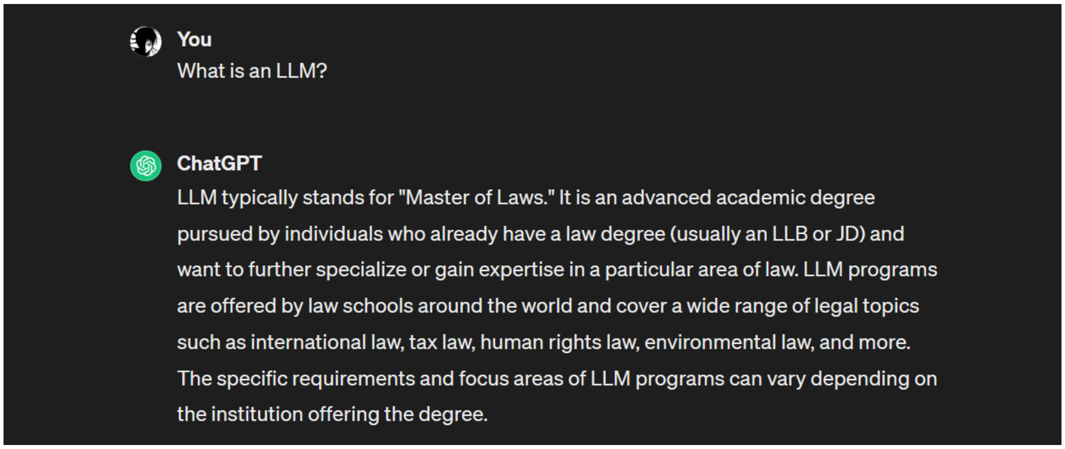
In the early 2010s, chatbots became more sophisticated by leveraging NLP and ML advancements. These chatbots can understand natural language data, including voice recognition, which allows them to perform more complex tasks. With these advancements, the development of personal assistants integrated into smartphones and computers became popular. Apple with Siri, Google with Google assistants, Amazon with Alexa, and Microsoft with Cortana were some of the most popular at the time. These chatbots could receive voice communication and use the internet to have more meaningful conversations. However, this technology still needed to fully understand the user queries and could have been better at understanding the full context of conversations [7].

Around 2018, chatbots took advantage of new machine learning techniques such as transformers and the ability to process large amounts of data. With these machine learning tools, chatbots could be trained with large datasets and take advantage of the transformer models to better understand the context of NLP data. These discoveries lead to the creation of LLMs such as ChatGPT and Bard. These chatbots are not only capable of responding to simple user queries but can also generate text, code, and images for a variety of areas. These chatbots have many applications in areas such as marketing, education, health, and entertainment [7]. They have proved to be very helpful for users, which led most companies to strive to incorporate chatbots in their products.

2.2 Prompt Engineering

Prompt engineering involves designing and refining inputs for language models to optimize their outputs for specific tasks. This method is crucial for directing AI to understand and answer questions with accuracy and relevance [8]. Through prompt engineering, we can tailor the model's response generation by translating human intentions into a format that the AI can efficiently process. For example, a well-designed prompt can guide the model to produce text in a desired style or to answer a question with a specific level of detail [8].

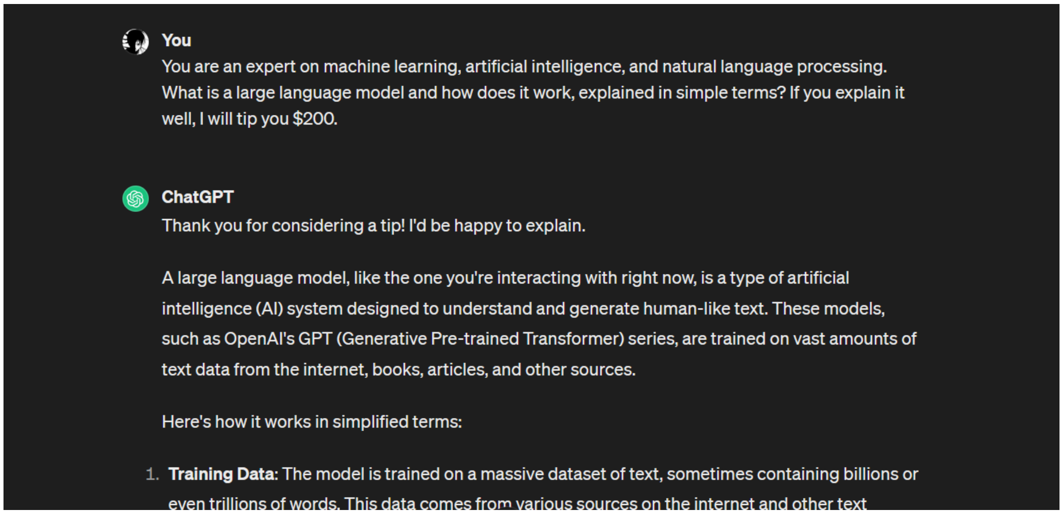
The effectiveness of prompt engineering depends on a deep understanding of the model's strengths and weaknesses. By trying out different prompt styles, engineers can find the best ways to get the responses they want [9]. This often includes careful selection of keywords, providing the right context, and how instructions are framed. Adjusting the tone, format, or specific details included can significantly change the output.



You
What is an LLM?

ChatGPT
LLM typically stands for "Master of Laws." It is an advanced academic degree pursued by individuals who already have a law degree (usually an LLB or JD) and want to further specialize or gain expertise in a particular area of law. LLM programs are offered by law schools around the world and cover a wide range of legal topics such as international law, tax law, human rights law, environmental law, and more. The specific requirements and focus areas of LLM programs can vary depending on the institution offering the degree.

Figure 1: A typical prompt given to ChatGPT 3.5 Turbo [12].



You
You are an expert on machine learning, artificial intelligence, and natural language processing. What is a large language model and how does it work, explained in simple terms? If you explain it well, I will tip you $200.

ChatGPT
Thank you for considering a tip! I'd be happy to explain.

A large language model, like the one you're interacting with right now, is a type of artificial intelligence (AI) system designed to understand and generate human-like text. These models, such as OpenAI's GPT (Generative Pre-trained Transformer) series, are trained on vast amounts of text data from the internet, books, articles, and other sources.

Here's how it works in simplified terms:

1. **Training Data**: The model is trained on a massive dataset of text, sometimes containing billions or even trillions of words. This data comes from various sources on the internet and other text

Figure 2: An engineered prompt given to ChatGPT [12].

Figures 1 and 2 collectively illustrate the nuances of prompt engineering in natural language processing. In Figure 1, the ambiguity of the term "LLM" led to a misinterpretation by ChatGPT, highlighting the challenge of ambiguous prompts and the necessity for precise context specification in prompt design. Conversely, Figure 2 demonstrates the model's adeptness at explaining complex concepts, like large language models, in a straightforward and accessible manner when provided with a clear and contextually relevant prompt. These interactions demonstrate the critical role of refined prompt engineering in guiding AI models toward accurate interpretations and informative responses, showcasing both the potential and limitations of AI in educational and explanatory applications.

In our project, we use prompt engineering to improve how users interact with the WPI chatbot. We refine prompts to ensure that the chatbot's responses to WPI academic advising questions are relevant and accurate. Choosing the right prompts is essential for the chatbot's ability to understand user queries and provide or generate suitable information [9]. Through extensive testing and iteration, we identify prompts that lead to coherent and contextually relevant responses, ensuring the chatbot meets user expectations and conveys information effectively.

Moreover, prompt engineering is not just about optimizing responses. It also involves using insights from user interactions to enhance the chatbot's performance over time. By evaluating different prompts' effectiveness, we can adjust our strategy to match evolving trends and preferences. This ongoing refinement is key to developing responsive and informative AI applications, marking prompt engineering as a fundamental aspect of our project in creating an effective WPI chatbot.

2.3 Current Limitations of LLMs

Training limitations create challenges in the development of today's LLMs. Recently released models, such as GPT-4, owe most of their success to large increases in their model parameters and training dataset sizes [10]. However, the additional hardware and electricity needed to make these changes dramatically affect training costs [10]. As models are trained with more parameters, the cost to train increases.

Table 1: Number of parameters vs. estimated training costs for GPT models [10].

Model	Year	Parameters	Training cost	Source
GPT-2	2019	1.5 billion	$100,000	Wiki, based on BERT cost x10 for model size
GPT-3	2020	175 billion	$10 million	Blog post and forum post, maybe at high end
GPT-4	2023	2 trillion	$100 million	Sam Altman quote, anonymous estimate of 1 trillion which I rounded for consistency
GPT-5	2025	20 trillion	$1 billion	Epoch estimate of 100M every 2 years
GPT-6	2027	200 trillion	$10 billion	As above

The resources needed to train these models are prohibitively expensive, making it almost impossible for most companies to develop an LLM model near the scale of GPT. The massive training cost, along with the large amount of time needed to develop the LLM, make it difficult to repeatedly update any LLM. As shown in Table 1, it may take multiple years to train and develop the next large-scale LLM. During this time, new information is discovered, existing data may be updated, and new world events take place, which means that by the time the model is done training and released to the public, the dataset it was trained on is obsolete or inaccurate. ChatGPT and Google Bard reflect this idea as of January 2024, with their most recent models trained on core datasets from only 2022 and earlier.

Even if an LLM was trained on a consistently up-to-date dataset, it still may produce flawed or incomplete responses. For instance, LLMs are known to "hallucinate", meaning they sometimes make up nonsensical or incorrect data [11]. There are several factors that may contribute to data hallucination. As discussed earlier, many LLMs have improved by increasing their volume of training data, but increasing the training dataset size means that there is much more data to be verified and cleaned, making it difficult to ensure the quality of the entire dataset [11]. This may introduce human bias and incorrect information into the LLM's responses. Furthermore, many LLMs today are trained to be general purpose, utilizing publicly available data from the internet [11]. While this set of training information is broad, enabling the model to answer many generic questions, it lacks depth in various specialized fields such as medicine and law [11]. When LLMs are tasked with domain-specific knowledge of a specialized field, they often hallucinate data in their responses or cannot respond to the query at all because they are not well-trained on that domain.

ChatGPT 3.5 Incorrect Response Example:

User: Who teaches the BCB 510 - BCB Seminar course at Worcester Polytechnic Institute?

ChatGPT: As of my last knowledge update in January 2022, I don't have specific information about the current instructors for courses at Worcester Polytechnic Institute (WPI), including BCB 510 - BCB Seminar. Faculty assignments can change, and it's best to check the most recent course schedule or contact the relevant department at WPI for the latest information on course instructors. You can visit the official WPI website or contact the Biology and Biotechnology Department for the most accurate and up-to-date details on the BCB 510 course and its instructors.

Google Bard Incorrect Response Example:

User: Who teaches the BCB 510 - BCB Seminar course at Worcester Polytechnic Institute?

Bard: Unfortunately, there isn't usually a dedicated instructor assigned to BCB 510 - BCB Seminar at Worcester Polytechnic Institute. This is because it's a seminar course designed for students to present their research and hear talks from guest speakers [1].

The guest speakers for the BCB seminar course will likely vary depending on the semester and the research focus of the department at that time.

An LLM trained on only publicly available data will not have access to domain-specific knowledge needed to discuss some topics. ChatGPT 3.5 [12] (Top) and Google Gemini [13] (Bottom) unable to provide real-time information specific to WPI.

Even with a quality dataset, LLMs' training process and architecture itself can cause hallucinations. This unintended behavior can be introduced into the LLM due to its first stage of training, known as pre-training. Pre-training develops the LLM's foundational understanding of language syntax, world knowledge, and reasoning [11]. In pre-training, the LLM begins generating text by predicting the next token, or section of text, depending on previous values in the sequence in a left-to-right manner [11]. This unidirectional method may result in the model losing some understanding of the context if it is not presented in order, resulting in hallucinations [11]. The next phase of training an LLM, Supervised Fine-Tuning (SFT), adjusts the model with a set of labeled data in the form of annotated (instruction, response) pairs [11]. SFT improves the model's output and its performance on unseen tasks, generalizing its abilities [11]. However, if the annotated data used for SFT exceeds the knowledge boundaries defined in pre-training, the LLM will try to produce content beyond what it knows [11]. This is called Capability Misalignment, and it also may lead to hallucinations [11].

2.4 Retrieval Augmented Generation and Related Works

To understand current methods for improving LLM output, our team read through several research papers. Retrieval Augmented Generation (RAG) [14] and Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast [15] inspired our project.

RAG is a current fine-tuning method used to improve the output of an LLM model. Broadly speaking, it incorporates information from external data sources into the LLM's output using a document retriever feeding into a generator. In this paper, the retriever encodes the user's question using a pre-trained BERT document encoder. Next, maximum inner product search (MIPS) is used to search through the dataset and retrieve one or more documents relevant to the query. The retrieved content is aggregated and concatenated to the original input, and the result is given to the generator, a pre-trained BART-large model that generates the final response.

RAG is an important development for generative NLP models because its responses incorporate real, factual knowledge, making the model less susceptible to data hallucinations. Additionally, it is easier to customize the model and ensure the data it uses is up-to-date with the RAG by simply updating the document pool it draws from instead of having to re-training the entire model.

Since its creation, the RAG has been incorporated into many frameworks for LLM applications. LangChain, one such LLM framework, provides a set of modules that can be chained together to create custom LLM pipelines. LangChain is known for its ease in integrating different data sources and interacting with other applications.

In Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast [15], LangChain is used to rapidly develop custom LLM applications. LangChain supports "chains", or custom sequences of calls to tools, data preprocessing, or the LLM [15], that can be stacked to create the application. In this paper, several chains consisting of an LLM and a prompt are combined to perform multiple operations on the input. One pipeline discussed in the paper involves first feeding the input to a "router chain", which evaluates the input and then redirects it to a subchain specializing in that area. A tutor LLM app, for example, may have a router chain that evaluates a student's question, and then, depending on the subject, routes it to a math chain, history chain, or science chain equipped to handle the question. Any of the subchains may be equipped with RAG to answer the question with current, factual information. The subchains can be customized to specialize in many different areas, and act as a simple building block for a custom LLM application.

To understand methods and tools currently used for other university chatbots, we looked at Saint Louis University's QnABot. This application uses the open source QnABot from Amazon Web Services to retrieve information from the internet using Lambda hook functions. When a student prompts the QnABot, the system retrieves the appropriate answer from online at the time of the query and incorporates it into the response. This gives the student real-time responses and resources to answer their questions.

2.5 Text Embeddings

Word embeddings are vector representations of words that allow the computation of semantic similarities [16]. Word embeddings are very useful for AI to understand and generate NLP data because computers can easily make sense of vector representations. These embeddings can be represented in a multidimensional plot, where similar words will be closer together.

Word embeddings are great for representing words, but most applications need to get the embeddings from sentences and documents. There are many strategies to achieve this goal by leveraging the property of word embeddings. An intuitive approach is to use the frequency of words to compare similar documents. Since common words like "a", "and", and "the" have a high frequency in all documents, we take the inverse of the document frequency. We create a matrix with all the documents and the word frequencies, which results in a row vector for each document. We compare these vectors to get the similarity between documents; this approach is called TF-IDF (Term Frequency-Inverse Document Frequency). This model is easy to interpret but it does not take into account the order of words and is memory intensive.

Another approach is to use neural networks with a softmax layer to predict words. The softmax layer is a function that normalizes the input to a probability distribution for the number of classes. This function is suitable for classification purposes. Training this model consists of predicting the next word based on the previous words. As Figure 3 shows, the model uses a fixed-size window of words and passes the concatenation of their embeddings to a neural network model with a softmax layer to predict the next word. After training the model with each word and adjusting the weights, then we can use all the words to compute the embedding of the document. This approach captures short-term relationships between words but fails to keep track of long-term relationships like long sentences or sentences in a paragraph. The limitation is that a single vector cannot represent all the information to decide on the next word. To solve this limitation, attention mechanisms and transformer models were introduced.

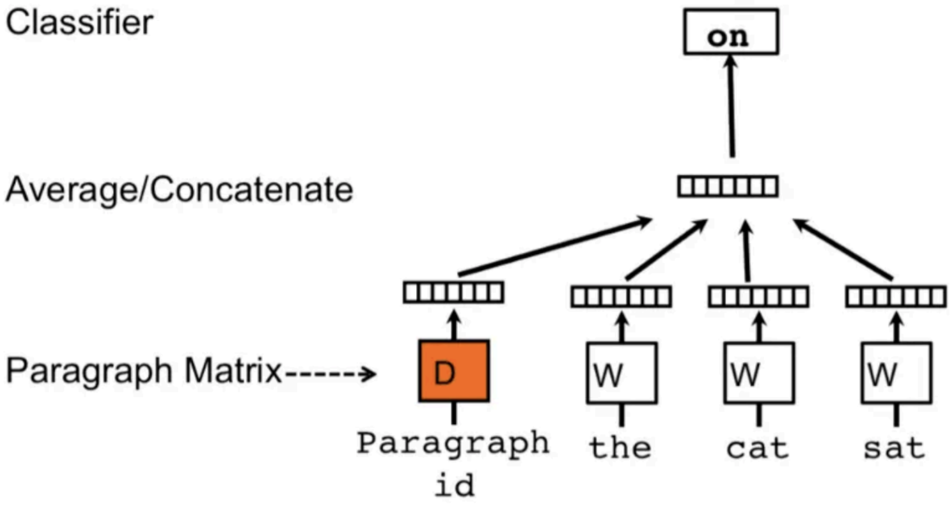


Figure 3: Neural Network architecture for getting sentence embeddings [17].

Models with an attention mechanism have an encoder and decoder part. The encoder processes each item of the sequence and produces a vector called context with the extracted information. The decoder gets the context from the encoder, processes this information, and produces the relevant output. Unlike the previous model, the decoder gets the context of every state to decide on the next word of the sequence. However, processing all these vectors is expensive and not all of them contain relevant information. That is why the decoder uses an attention mechanism to filter the relevant hidden states by assigning them scores and passing the result through a softmax layer. The transformer models optimize this process by providing parallel computation through positional encoding and multi-head attention. Transformers assign unique numbers to words, so they do not need to process the words sequentially and can apply multi-head attention models to different parts of the document. This model performs well in capturing the relationship between words and context, but large training sets are required to achieve accurate results.

2.6 Pre-trained embedding models

For embeddings to perform well, we need to train them with large amounts of data. With this training, embedding models can identify the relationship between words and patterns in the data using the previously discussed models. Machine learning applications usually use pre-trained neural network models that have been trained in large amounts of data. In this project, we chose the embedding model BGE-small and BGE-base because of their high score in the Massive Text Embedding Benchmark (MTEB). MTEB is a benchmark that measures the capabilities of embedding models against eight embedding tasks across 58 datasets and 112 languages [18]. This benchmark ensures that an embedding model is not only suitable for tasks in a specific set of datasets but also applicable to other embedding tasks [18]. The BGE model is at the top in the leaderboard of HuggingFace in the categories of reranking with 58.94 and retrieval with 53.25 scores [19]. These are the two most important categories for the WPI chatbot because we have a set of documents all WPI-related and we want to find relevant documents to the user query for the LLM context. In addition, BGE models are lightweight because they occupy less memory than embeddings from LLMs and can make accurate inferences with fewer computational units [20]. These advantages make BGE models suitable for retrieval and ranking tasks with an efficient computational time.

ChatGPT from OpenAI, one of the most powerful LLMs, also has its embedding model. This model derives from ChatGPT and has the same knowledge about the statistics of words. Since ChatGPT has been trained with large amounts of text and has proven effective at NLP tasks, its embedding model is a state-of-the-art approach. However, the rate limit is very restrictive, with only three function calls per minute in the free account. This limitation means that getting all the embeddings of documents for a big AI application will take an unreasonable amount of time. OpenAI is a state-of-the-art approach for getting document embeddings, but we need to be careful to handle rate limit exceptions.

As discussed in this section, The BGE and Open AI embedding models work well for representing the meaning of words, but to perform similarity and retrieval operations in a scalable way, we need a vector database.

2.7 Vector Database

A vector database is a specialized database in vector embeddings that allows optimized retrieval and similarity search calculation [21]. Since machine learning models are trained with large texts that require embedding manipulations, vector databases have become a very useful tool nowadays. Traditional databases struggle to manipulate embeddings because embeddings usually have large dimensions and encode complex meanings, which require specialized operations to handle them.

Vector databases have three main advantages over traditional databases that make them suitable for machine learning models: Similarity search, unstructured data, and big data management. Vector databases offer a similarity search of vectors, which a traditional database cannot because it only searches for exact matches or predefined conditions [21]. Also, vector databases can handle unstructured data such as video, images, and audio by transforming them into vectors [21]. In terms of performance, traditional databases do not perform well in big data because of latency issues, scalability, and concurrency conflict [21]. On the other hand, vector databases can use different techniques such as caching, partitioning, and sharding to distribute the work into different machines to optimize the process [21]. As a result, vector databases offer the advantages of traditional databases and provide optimizations to perform embedding operations.

Furthermore, traditional and vector databases also have important distinctions in their workflow. Traditional databases search by row looking for the exact match of a query. On the other hand, vector databases use a similarity search to find the vectors that are most similar to the query. Vector databases implement this type of search, Approximate Nearest Neighbor (ANN), using a pipeline that offers fast and accurate retrievals (Figure 4). To add a vector, the database maps the vector to a data structure that allows fast search and retrieval [22]. To process a query, the database compares the index of the query to the indexes in the database using similarity scores [22]. After getting similar vectors, the database could rerank the results with different similarity scores in a process called post-processing [22].

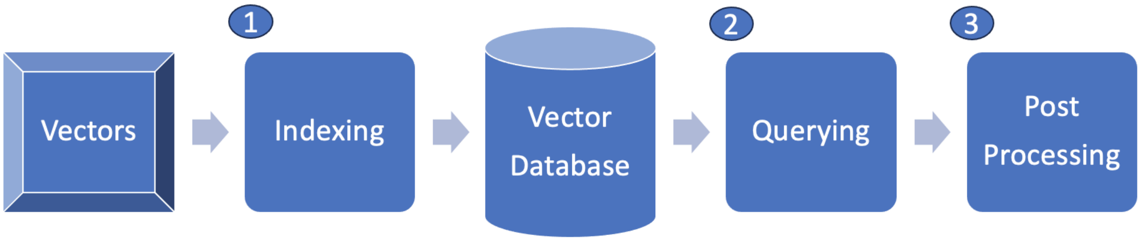


Figure 4: The flow of how vector databases work [22].

2.8 Pinecone Vector Database

Pinecone offers a vector database managed in the cloud that has simple API calls to perform operations with vectors. The storage limit of Pinecone varies depending on pricing with the free version having a capacity of 100,000 vectors and the most expensive option, costing around $30,000, has a capacity of 10B vectors. Pinecone offers a highly accurate and low latency service to operate with vectors. This vector database was tested with the MSMarco V2 dataset of 138M embeddings and the results were 96% in recall and 51ms query latency [23]. These results mean that even when working with a dataset with billions of records, the response time would still be fast. In addition, Pinecone offers automatic index matching, which means that the user would always have access to up-to-date information. Pinecone offers the option to search by metadata allowing for more customizable queries to the database. There is also a functionality called collection, which allows the user to back up the information in the database.

Canopy is an open-source framework built on top of Pinecone that allows users to streamline the process of building a RAG application [24]. Canopy is compatible with all Open AI LLM which provides great flexibility to implement a chat engine. Canopy offers a full RAG workflow with three main components: knowledge base, context engine, and chat engine [24]. The user only needs to provide the text data and the knowledge base chunks and calculate the embeddings to upsert the data to the vector database. With the vector database, the context engine finds the most relevant documents and provides them as context for the LLM. Finally, the chat engine implements the two previous components to answer a user's query. The chat engine understands the chat history, identifies the user's questions, and transforms them into embeddings. It then uses the context engine to identify relevant documents and produce a relevant answer. Figure 5 depicts this process.

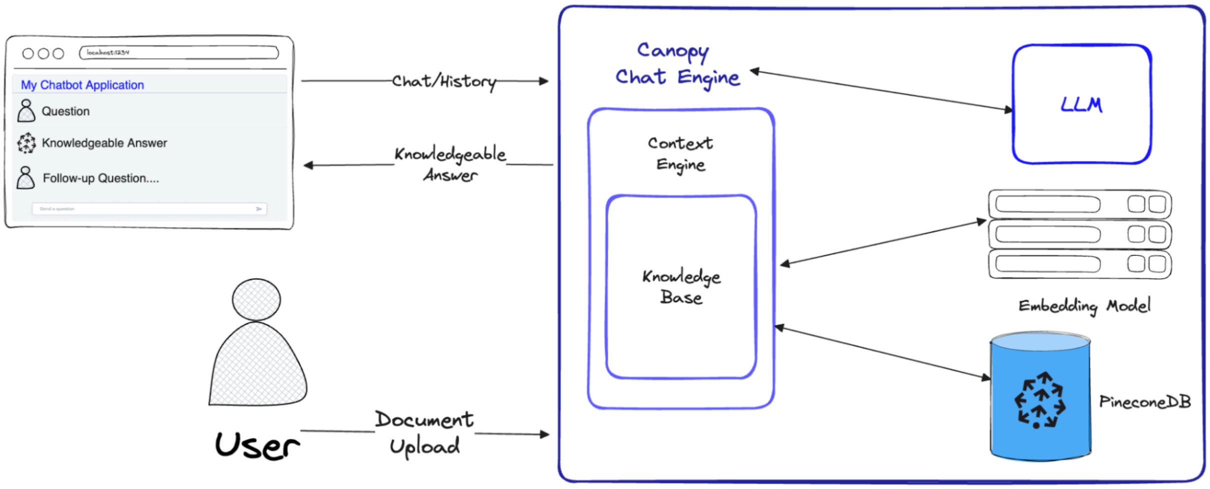


Figure 5: The flow of how Canopy implements a RAG application [24].

3 Methodology

The goal of this project was to explore the possibilities of building a chatbot for academic advising information at WPI and to identify strengths and weaknesses in different techniques along with developing a frontend for user interaction. We tested three different methods with slight variations. We gathered WPI-related data for each method and used it as context to help the Large Language Model (LLM) better understand and answer any question we asked. The first method we tested involved using an embedding model and storing the resulting document representations in a vector database. For the second method, we explored Pinecone's Retrieval Augmented Generation (RAG) model. Our final method utilized OpenAI's Assistants API. To achieve our goals for this project, we developed the following objectives:

1. Explore dataset creation of WPI information through different data-gathering methods.
2. Build a RAG Pipeline using embedding models like BGE-small or BGE-base.
3. Explore the Pinecone Canopy tool.
4. Explore different methods of using OpenAI's Assistant API to build chatbots.
5. Use one of the pipelines to build an example front and back end for the chatbot.
6. Compare and determine strengths and weaknesses for each of the pipelines.

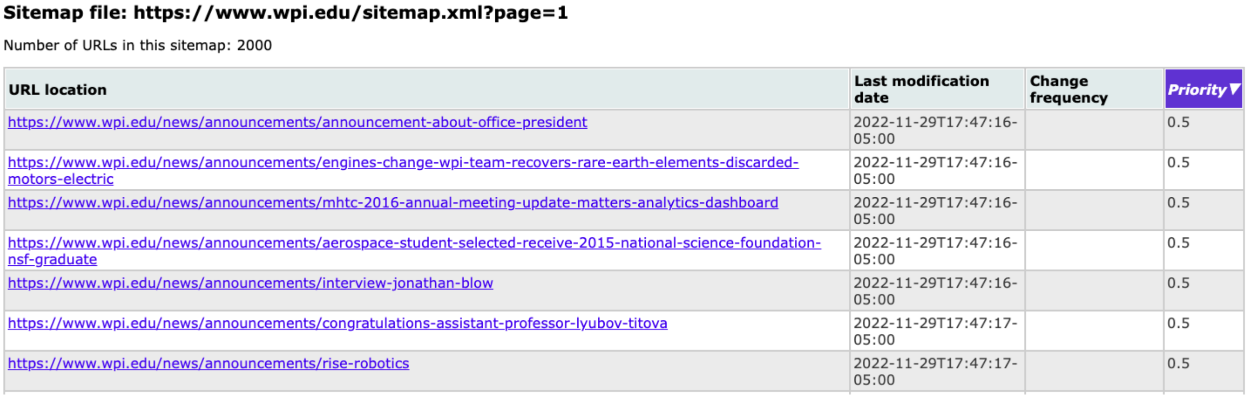
Our objectives will be explained in greater detail in the following subsections. We will discuss the methods we used and the justification behind them to accomplish the goals and objectives of this project.

3.1 Dataset

The dataset was a key aspect of this project as it was used throughout the project in all of our testing methods and was constantly being modified. This section will go over some key points in our data collection journey.

3.1.1 Collection of Data

The data for this project was gathered through WPI online resources. Originally, a Python script was used to gather WPI webpage data using the WPI.edu sitemap.xml [25]:



Sitemap file: https://www.wpi.edu/sitemap.xml?page=1

Number of URLs in this sitemap: 2000

URL location	Last modification date	Change frequency	Priority▼
https://www.wpi.edu/news/announcements/announcement-about-office-president	2022-11-29T17:47:16-05:00		0.5
https://www.wpi.edu/news/announcements/engines-change-wpi-team-recovers-rare-earth-elements-discarded-motors-electric	2022-11-29T17:47:16-05:00		0.5
https://www.wpi.edu/news/announcements/mhtc-2016-annual-meeting-update-matters-analytics-dashboard	2022-11-29T17:47:16-05:00		0.5
https://www.wpi.edu/news/announcements/aerospace-student-selected-receive-2015-national-science-foundation-nsf-graduate	2022-11-29T17:47:16-05:00		0.5
https://www.wpi.edu/news/announcements/interview-jonathan-blow	2022-11-29T17:47:16-05:00		0.5
https://www.wpi.edu/news/announcements/congratulations-assistant-professor-lyubov-titova	2022-11-29T17:47:17-05:00		0.5
https://www.wpi.edu/news/announcements/rise-robotics	2022-11-29T17:47:17-05:00		0.5

Figure 6: A portion of WPI sitemap.

Using the vast amount of URLs on the WPI sitemap, we gathered over 8,000 web pages deemed relevant to the WPI student experience:

Table 2: Info on webpages gathered.

	Total Pages	Total Word Count	Average Word Count	Estimates Token Count
All	8308	3,115,270	~375	4,153,693

However, after review, it was determined that it would be better to test our methods on a smaller data set that was more targeted to certain aspects of the student experience. These aspects were courses, degree requirements, and meeting place reservation times.

Table 3: Key Resources for Data Gathering.

Title	Purpose	Resource
Tracking Sheet	Degree Requirement Data	[26]
WPI Library	Tech Suite Data	[27]
WPI 25Live	Room Availability Data	[28]
WPI Course Listings	WPI Courses Data	[29]

Through these resources, we were able to collect enough useful data to create a WPI academic advising chatbot.

Data Breakdown:

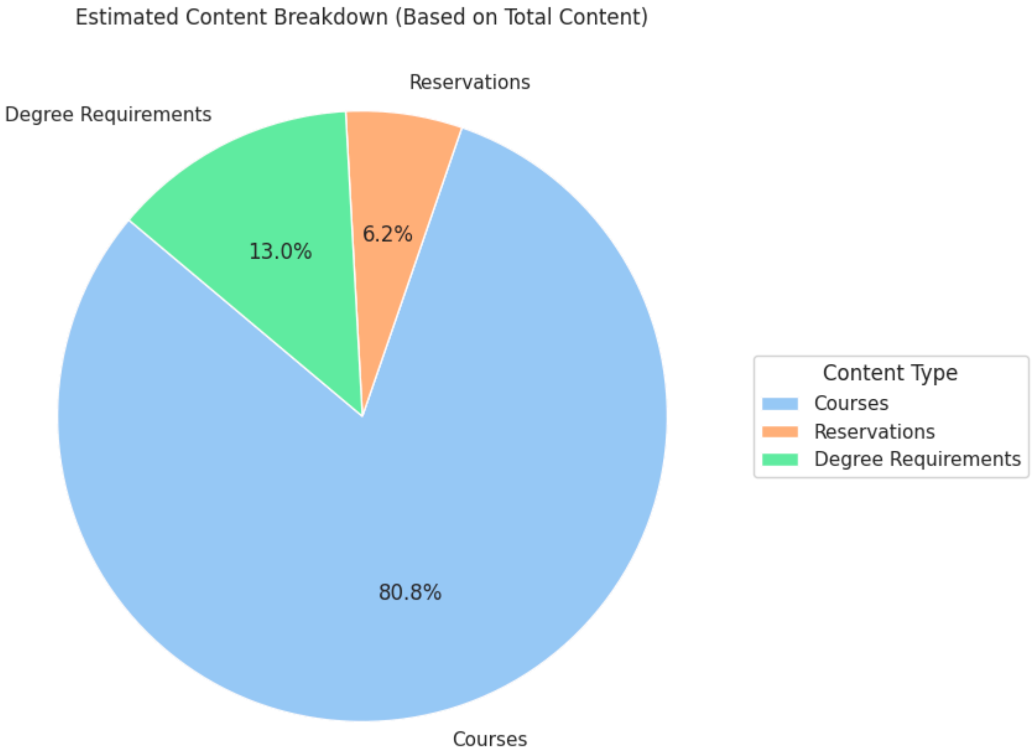
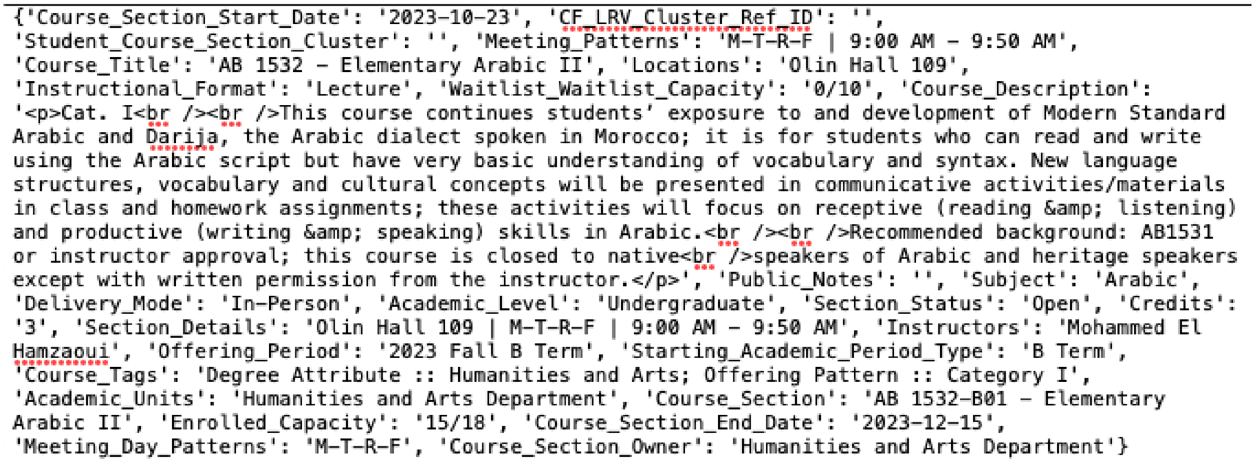


Figure 7: Breakdown of WPI pages gathered.

Figure 7 shows an estimated breakdown of the percentage of data collected for each category after processing. The total amount of data was about 4,654,895 bytes. These values are likely to change somewhat every time we collect and process the data. However, since there are over 3000 sections, courses are likely to be the majority of the data. Overall, the amount of data collected and the variance of that data allowed us to test the WPI chatbot pipelines efficiently.

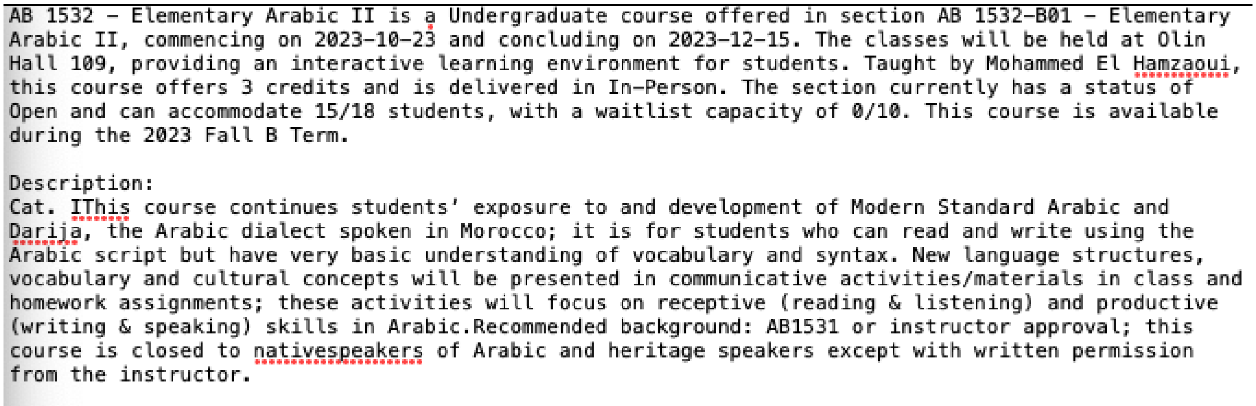
3.1.2 Data Cleaning and Modification

Most of the data collected in its raw form was in JSON, HTML, or PDF format. The data cleaning and modification involved taking these raw formats and converting them into a text-based format. This was done to make the data more natural language-like, which was accomplished through the use of language templates and a Python script. It is important to note that changing the format of the data has minimal impact on the LLM's ability to properly answer a question as seen in Figures 10 and 11. These figures show that ChatGPT can properly answer the question regardless of whether it was given the JSON format of course information or a natural language format of course information. In addition, the cosine similarity score of embeddings from a BGE model and a user query were in a comparable range with the natural language text, slightly beating out the structured format description of the same course.



```
{'Course_Section_Start_Date': '2023-10-23', 'CF_LRV_Cluster_Ref_ID': '',
'Student_Course_Section_Cluster': '', 'Meeting_Patterns': 'M-T-R-F | 9:00 AM - 9:50 AM',
'Course_Title': 'AB 1532 - Elementary Arabic II', 'Locations': 'Olin Hall 109',
'Instructional_Format': 'Lecture', 'Waitlist_Waitlist_Capacity': '0/10', 'Course_Description':
'<p>Cat. I<br /><br />This course continues students’ exposure to and development of Modern Standard
Arabic and Darija, the Arabic dialect spoken in Morocco; it is for students who can read and write
using the Arabic script but have very basic understanding of vocabulary and syntax. New language
structures, vocabulary and cultural concepts will be presented in communicative activities/materials
in class and homework assignments; these activities will focus on receptive (reading &amp; listening)
and productive (writing &amp; speaking) skills in Arabic.<br /><br />Recommended background: AB1531
or instructor approval; this course is closed to native<br />speakers of Arabic and heritage speakers
except with written permission from the instructor.</p>', 'Public_Notes': '', 'Subject': 'Arabic',
'Delivery_Mode': 'In-Person', 'Academic_Level': 'Undergraduate', 'Section_Status': 'Open', 'Credits':
'3', 'Section_Details': 'Olin Hall 109 | M-T-R-F | 9:00 AM - 9:50 AM', 'Instructors': 'Mohammed El
Hamzaoui', 'Offering_Period': '2023 Fall B Term', 'Starting_Academic_Period_Type': 'B Term',
'Course_Tags': 'Degree Attribute :: Humanities and Arts; Offering Pattern :: Category I',
'Academic_Units': 'Humanities and Arts Department', 'Course_Section': 'AB 1532-B01 - Elementary
Arabic II', 'Enrolled_Capacity': '15/18', 'Course_Section_End_Date': '2023-12-15',
'Meeting_Day_Patterns': 'M-T-R-F', 'Course_Section_Owner': 'Humanities and Arts Department'}
```

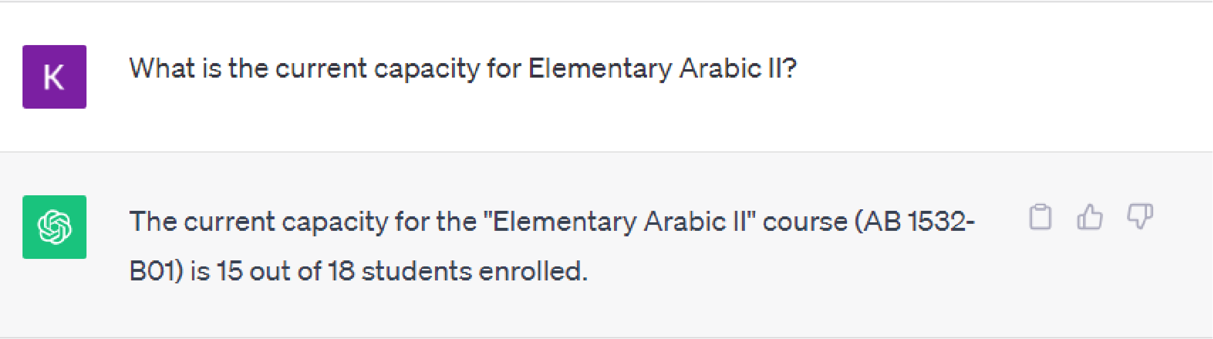
Figure 8: JSON Data for Elementary Arabic Course.



```
AB 1532 - Elementary Arabic II is a Undergraduate course offered in section AB 1532-B01 - Elementary
Arabic II, commencing on 2023-10-23 and concluding on 2023-12-15. The classes will be held at Olin
Hall 109, providing an interactive learning environment for students. Taught by Mohammed El Hamzaoui,
this course offers 3 credits and is delivered in In-Person. The section currently has a status of
Open and can accommodate 15/18 students, with a waitlist capacity of 0/10. This course is available
during the 2023 Fall B Term.

Description:
Cat. IThis course continues students’ exposure to and development of Modern Standard Arabic and
Darija, the Arabic dialect spoken in Morocco; it is for students who can read and write using the
Arabic script but have very basic understanding of vocabulary and syntax. New language structures,
vocabulary and cultural concepts will be presented in communicative activities/materials in class and
homework assignments; these activities will focus on receptive (reading & listening) and productive
(writing & speaking) skills in Arabic.Recommended background: AB1531 or instructor approval; this
course is closed to nativespeakers of Arabic and heritage speakers except with written permission
from the instructor.
```

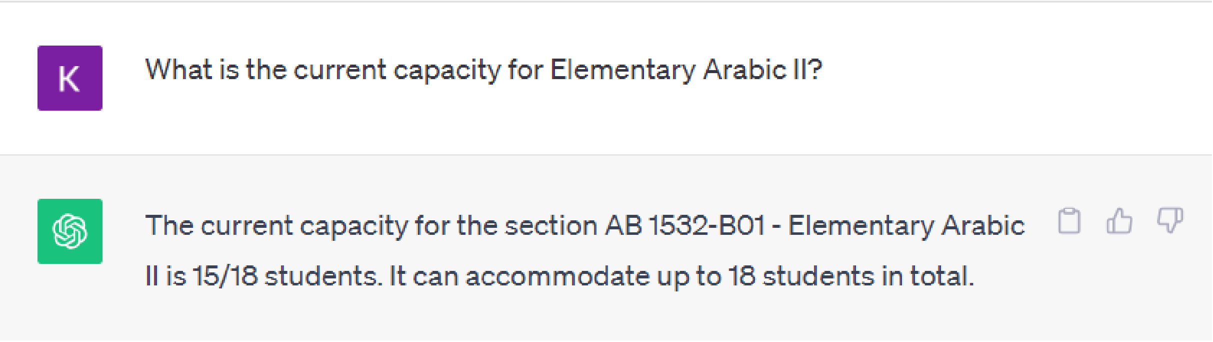
Figure 9: Natural Language Data for Elementary Arabic Course.


K

What is the current capacity for Elementary Arabic II?

The current capacity for the "Elementary Arabic II" course (AB 1532-B01) is 15 out of 18 students enrolled.

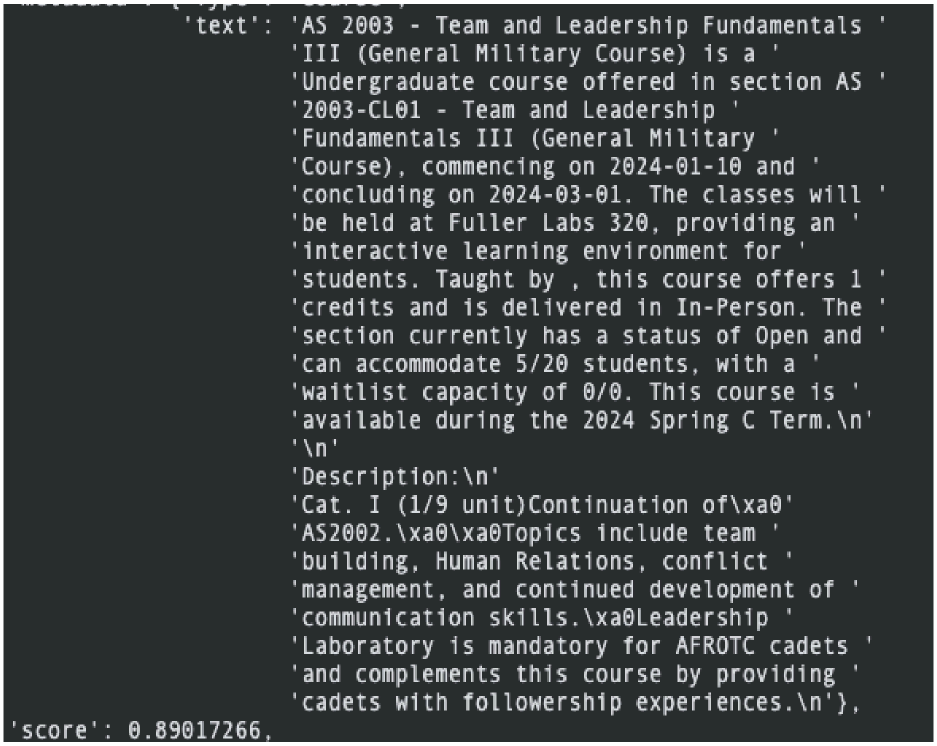
Figure 10: ChatGPT Response to JSON Format [12].


K

What is the current capacity for Elementary Arabic II?

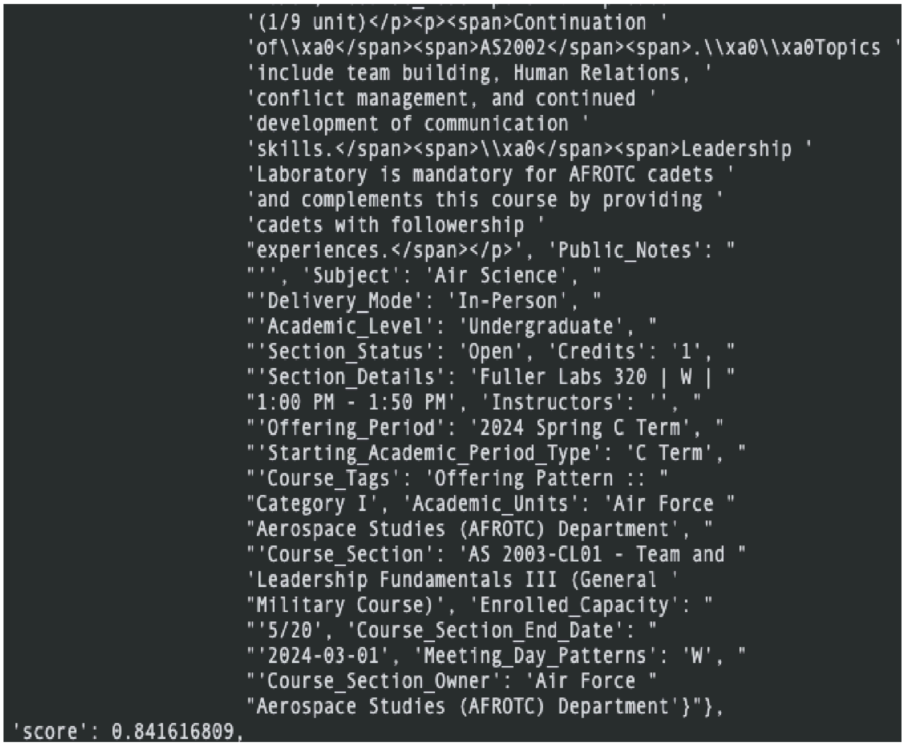
The current capacity for the section AB 1532-B01 - Elementary Arabic II is 15/18 students. It can accommodate up to 18 students in total.

Figure 11: ChatGPT Response to Natural Language Format [12].



```
             'text': 'AS 2003 - Team and Leadership Fundamentals '
                     'III (General Military Course) is a '
                     'Undergraduate course offered in section AS '
                     '2003-CL01 - Team and Leadership '
                     'Fundamentals III (General Military '
                     'Course), commencing on 2024-01-10 and '
                     'concluding on 2024-03-01. The classes will '
                     'be held at Fuller Labs 320, providing an '
                     'interactive learning environment for '
                     'students. Taught by , this course offers 1 '
                     'credits and is delivered in In-Person. The '
                     'section currently has a status of Open and '
                     'can accommodate 5/20 students, with a '
                     'waitlist capacity of 0/0. This course is '
                     'available during the 2024 Spring C Term.\n'
                     '\n'
                     'Description:\n'
                     'Cat. I (1/9 unit)Continuation of\xa0'
                     'AS2002.\xa0\xa0Topics include team '
                     'building, Human Relations, conflict '
                     'management, and continued development of '
                     'communication skills.\xa0Leadership '
                     'Laboratory is mandatory for AFROTC cadets '
                     'and complements this course by providing '
                     'cadets with followership experiences.\n'},
'score': 0.89017266,
```

Figure 12: Similarity score for natural language format of the user question “Can you tell me about the course Team and Leadership Fundamentals III?”.



```
                        '(1/9 unit)</p><p><span>Continuation '
                        'of\\xa0</span><span>AS2002</span><span>.\\xa0\\xa0Topics '
                        'include team building, Human Relations, '
                        'conflict management, and continued '
                        'development of communication '
                        'skills.</span><span>\\xa0</span><span>Leadership '
                        'Laboratory is mandatory for AFROTC cadets '
                        'and complements this course by providing '
                        'cadets with followership '
                        "experiences.</span></p>', 'Public_Notes': "
                        "'', 'Subject': 'Air Science', "
                        "'Delivery_Mode': 'In-Person', "
                        "'Academic_Level': 'Undergraduate', "
                        "'Section_Status': 'Open', 'Credits': '1', "
                        "'Section_Details': 'Fuller Labs 320 | W | "
                        "1:00 PM - 1:50 PM', 'Instructors': '', "
                        "'Offering_Period': '2024 Spring C Term', "
                        "'Starting_Academic_Period_Type': 'C Term', "
                        "'Course_Tags': 'Offering Pattern :: "
                        "Category I', 'Academic_Units': 'Air Force "
                        "Aerospace Studies (AFROTC) Department', "
                        "'Course_Section': 'AS 2003-CL01 - Team and "
                        'Leadership Fundamentals III (General '
                        "Military Course)', 'Enrolled_Capacity': "
                        "'5/20', 'Course_Section_End_Date': "
                        "'2024-03-01', 'Meeting_Day_Patterns': 'W', "
                        "'Course_Section_Owner': 'Air Force "
                        "Aerospace Studies (AFROTC) Department'}"},
 'score': 0.841616809,
```

Figure 13: Similarity score for structured format of the user question “Can you tell me about the course Team and Leadership Fundamentals III?”.

Even though there is not much difference in an LLM’s ability to answer a question given structured vs. natural language data, there are some benefits from changing the data into a natural language format. As seen in Table 4, there are instances where the natural language format has a smaller token count, which can lead to faster performance and lower costs when interacting with an embedding model like BGE or language models' APIs, such as with OpenAI’s models. In addition, with a RAG model, it is often common practice to return the context used to the user for further understanding. If this is the case, it can be useful to return natural language context compared to the structured data context.

Table 4: Token Comparison of Collected Data for Different Formats.

File Group	Text format (NL)	Cleaned HTML (NL)	HTML	JSON
AB 1532	262	--	--	471
AB 1533	260	--	--	475
Admissions	954	638	49326	--
Bursar	914	582	41844	--

3.1.3 Automation and Updating Data

In addition to dataset creation, another important aspect of this project and customized chatbots in general is the use of automation and scripts to update the data set on an interval basis. We developed a Python script that automatically goes to the data sources, gathers the data, and then transforms it into a cleaned format for the different pipelines to use. The script could be run daily to completely replace the data set with more up-to-date information. This would be useful for reservation data and specific course data. Figure 14, found on the next page, shows a visualization of this process.

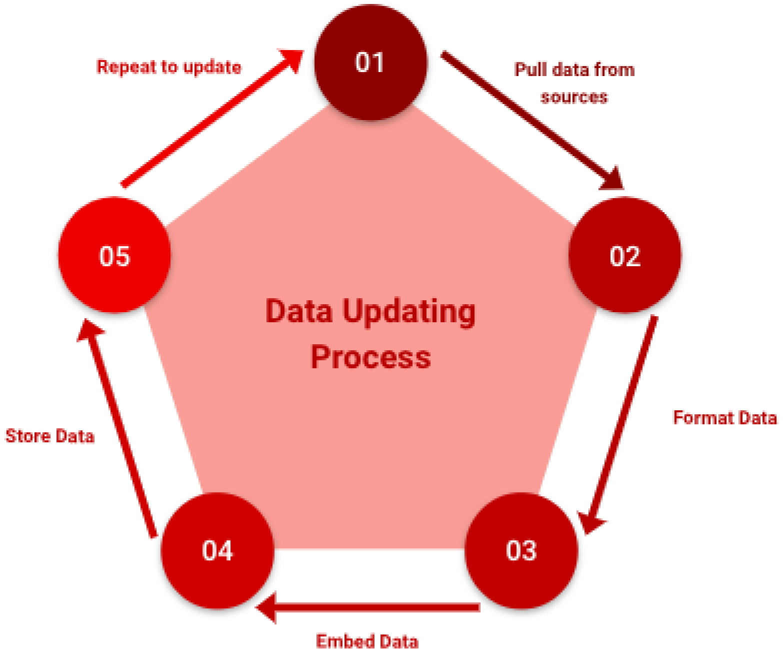


Figure 14: Data Updating Process.

This process was slightly different for our OpenAI Assistants pipeline, as steps 3 and 4 are done through OpenAI. This will be discussed further in section 3.5.

3.1.4 Data Storage

Data storage for this project was also important, as we needed a quick way to to get the information based on a user question and feed it to the LLM in order to get an appropriate response. In this project, data storage occurred in two different ways, one with Pinecone and one with OpenAI's file system. However, it is important to note that we also used services like Google Drive to help store data after gathering information from sources. Pipeline #1 and Pipeline #2 use Pinecone to help store data. After embedding the text, we gathered from sources and stored their vectors in Pinecone for later use.

3.1.5 Data Limitations

There were some limitations of the data we could collect and what we could do with it. The amount of data we had was limited as it takes more time to update the more data you collect. Additionally, a lot of the data changes often, which is why we chose a smaller dataset to test the methods. The data was also not always guaranteed to be completely up-to-date. Even if updated daily, data could still be changed elsewhere before updates.

3.2 Tools Used

In this section, we discuss the technical tools used for the project.

3.2.1 Versions of GPT

In our project, we integrated GPT-3.5-turbo and GPT-4, versions of OpenAI's Generative Pre-trained Transformer, to address varied operational needs in natural language processing (NLP). GPT-3.5-turbo, optimized for computational efficiency and quick response times, improves latency and throughput for immediate feedback applications like real-time user interactions and processing queries swiftly [30]. However, its focus on speed can sometimes limit the depth of response for complex inquiries. On the other hand, GPT-4 advances in text understanding and generation, with a larger dataset and refined algorithms for better context, nuance, and language complexity handling [31]. It generates coherent, detailed responses for tasks requiring deep content understanding and high-quality outputs.

The use of GPT-3.5-turbo and GPT-4 introduces challenges such as increased operational costs and the need to balance computational efficiency with response quality. GPT-3.5-turbo's rapid responses may lack complexity, while GPT-4's detailed outputs come with higher computational costs and potentially slower response times [32]. Both models have their strengths as well as their limitations. In real-world scenarios, managing these aspects is essential for a cost-effective and efficient system. This project's integration strategy utilized GPT-3.5-turbo for efficiency in scenarios demanding quick feedback and GPT-4 for complex inquiries needing depth and quality. This approach optimized performance to meet the project’s objectives more effectively.

3.2.2 Coding Tools

List of coding tools that were at least partially used to help facilitate the pipelines:

Google Colab
LangChain Python Library
Pinecone Python Library

OpenAI Python Library
Requests, CSV, Pandas, Numpy, Time, OS, JSON Python Libraries
OpenAI GPTs

3.3 Pipeline #1: RAG Model with LangChain Pinecone and BGE embedding Model

In this section, we discuss the first key method we used, which was developing a RAG model using LangChain. This RAG model parses and feeds documents to the BGE embedding model, which returns vectors that are stored in a Pinecone database. These vectors can be accessed and compared to help answer a WPI-related question with the assistance of a GPT model.

3.3.1 LangChain Usage

LangChain is a toolkit and framework that assists in the development of applications that integrate or use LLMs [33]. For this scenario, we used LangChain as a document loader to help the process of embedding documents by getting documents into a ready to embed format. This process is visualized in Figure 15.

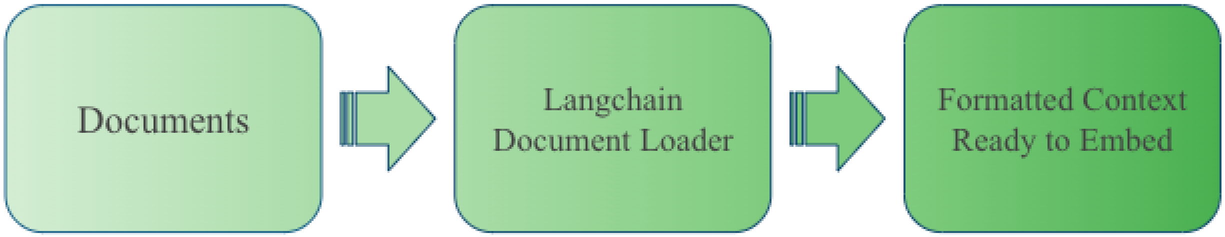


Figure 15: Usage Flow of LangChainTool.

3.3.2 BGE Usage

As mentioned in the background, in this project we use a BGE embedding model. For this pipeline, each of the formatted context segments the LangChain tool formatted we used this model to create a 768 dimension vector for each segment that was used to perform document analysis later with cosine similarity. The BGE model was also used to vectorize a user question so that it could be used to compare similarity to context segments. The process can be seen below in Figure 16.

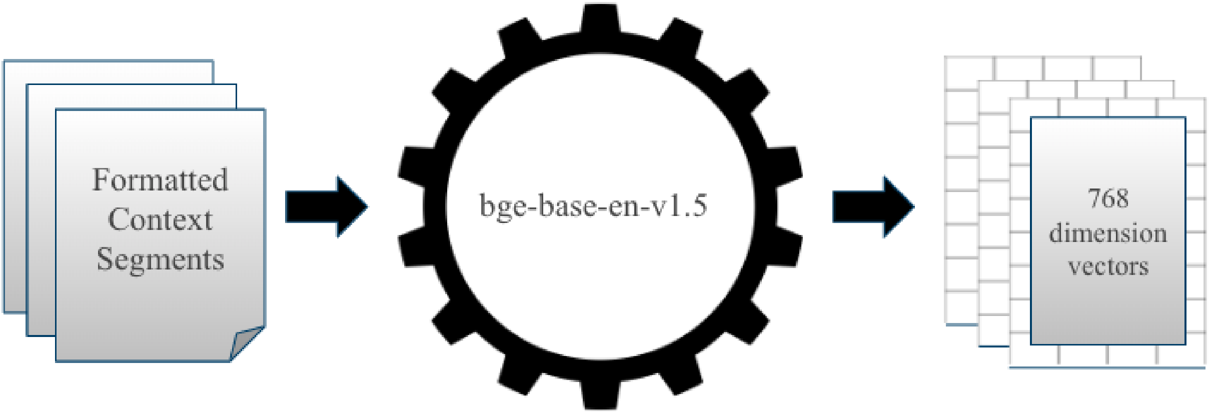


Figure 16: Usage Flow of LangChain Tool.

3.3.3 Pinecone Usage

Finally, these vector embeddings were stored in a pinecone database for continued use. The Pinecone database was also used for querying. Once we embed a user question and get a vector we can use that vector to compare with the stored vectors to find relevant context. Using cosine similarity we can retrieve the context with the highest similarity score, and then feed that into the LLM along with the user question to get back a reasonable answer. An example of this context and score can be seen in section 3.1.2 with Figure 12. Additionally, a general visual of Pipeline #1 is shown in Figure 17 on the next page.

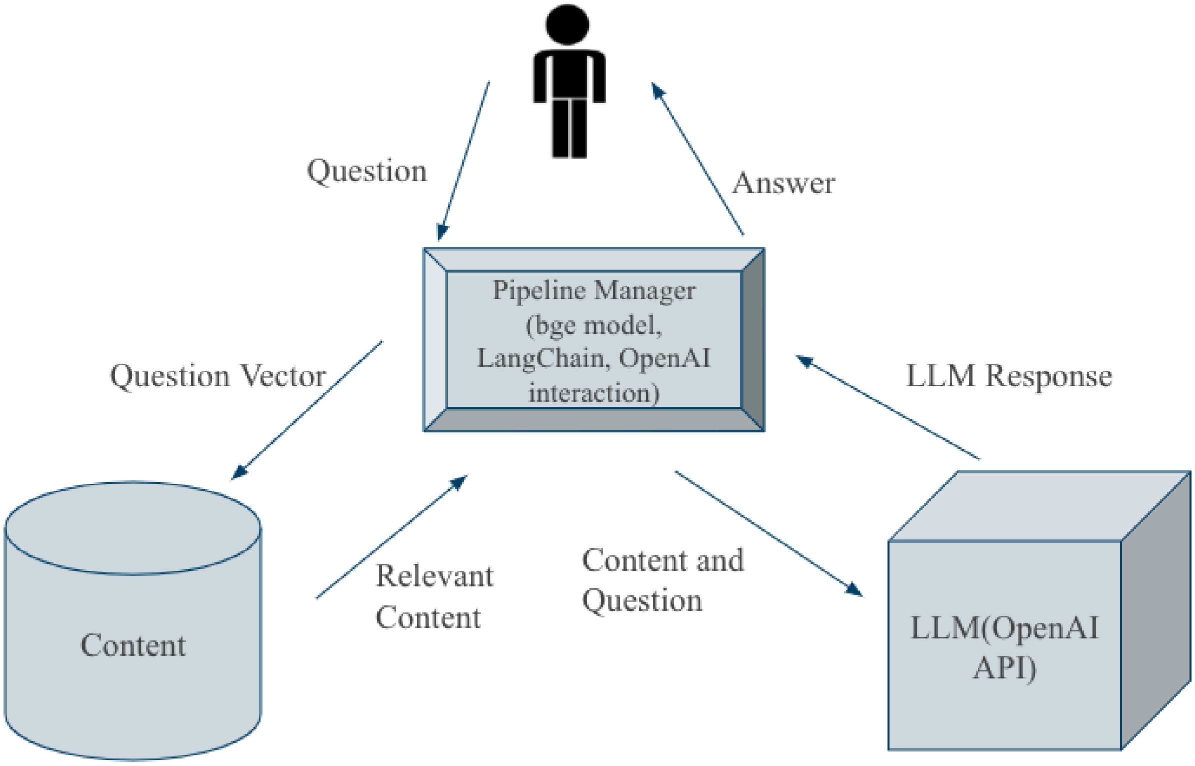


Figure 17: Pipeline #1 Visual.

3.4 Pipeline #2: RAG Model with Pinecone Canopy

To use Canopy, we created a new Canopy Pinecone Index, started a Canopy server, and loaded documents using Canopy. Canopy then automatically parsed and separated the documents and used OpenAI's text embedding model to create vector embeddings of the parsed documents. From there, a user can ask a question to the server. In response, Canopy will find relevant documents and send the documents along with the question to an OpenAI LLM model. Canopy then receives a knowledgeable response and relays it to the user.

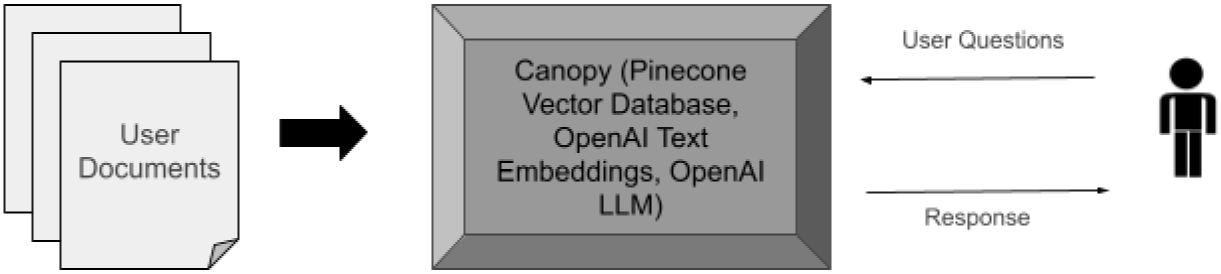

User Documents
Canopy (Pinecone Vector Database, OpenAI Text Embeddings, OpenAI LLM)
User Questions
Response

Figure 18: Canopy Pipeline Visual.

3.4.2 Key Differences

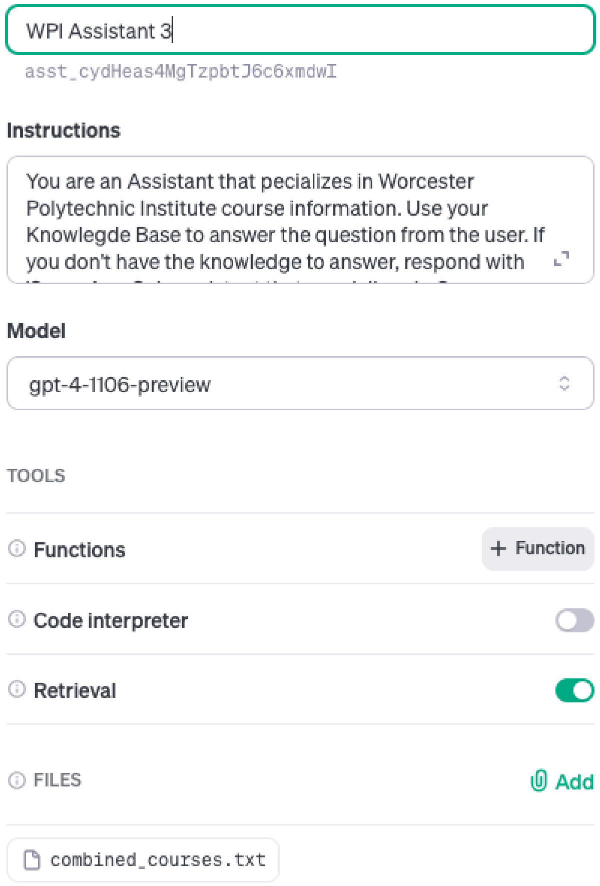
The main difference between Pinecone Canopy and Pipeline #1 is that Canopy does much of the legwork for you. All you need is the proper API keys for both Pinecone and OpenAI and the documents of information you want to upload. From there you can use a Canopy server's RESTful API to implement a chatbot.

3.5 Pipeline #3: OpenAI Assistants API with Question Classifier

In this section we will discuss the use of OpenAI assistants to build a WPI chatbot and Question classifier and how these were used to facilitate answering user questions from our WPI academic advising dataset.

3.5.1 Assistant Generation

The OpenAI Assistant generation was done in Python. For this project, we decided to have multiple assistants, each specializing in a specific WPI topic. To start, we had an assistant for courses, room reservation times, and degree requirements. Having specialized assistants might be done for various reasons, but some reasons we explored are providing specialized instructions and cost reduction. By having specialized assistants, we can provide each of those with specialized instructions to provide a better answer for the user which may be harder to do for a combined assistant. Separating assistants can also have cost benefits, as the knowledge retrieval tool OpenAI provides is often not used if the dataset is distributed between multiple assistants. Figure 19 is an example of one assistant created using a Python script. Additionally, when the data must be updated, we can quickly delete the assistants, prepare the up-to-date documents, and then recreate the assistants all in Python.



WPI Assistant 3

asst_cydHeas4MgTzpbtJ6c6xmdwI

Instructions

You are an Assistant that pecializes in Worcester Polytechnic Institute course information. Use your Knowlegde Base to answer the question from the user. If you don't have the knowledge to answer, respond with

Model

gpt-4-1106-preview

TOOLS

Functions

Code interpreter ○

Retrieval ●

FILES

combined_courses.txt

Figure 19: OpenAI Assistant Example.

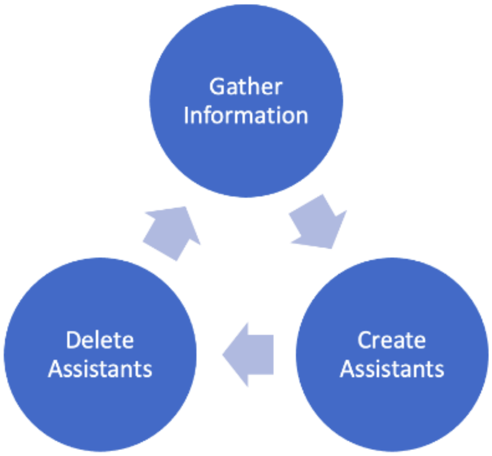


Figure 20: OpenAI Assistant Example.

3.5.2 Question Classifier

To use multiple assistants in a single chatbot pipeline, we need some sort of classifier that can determine which assistant is best for a user question. In this project, we explored two different methods involving a classifier. The first method used an embedding model for classification. Each assistant would have a description that is embedded and represented as a vector. When the user makes a query, the query is also embedded to create a vector and used to perform a similarity search among the embedded assistant descriptions. The closest matching assistant description determines which assistant will ultimately answer the question. The final method, which we incorporated into our web app, involved making another OpenAI assistant to function as a classifier. By giving the assistant classifier descriptions of each of the assistants, it determines which assistant is best for a specific question.

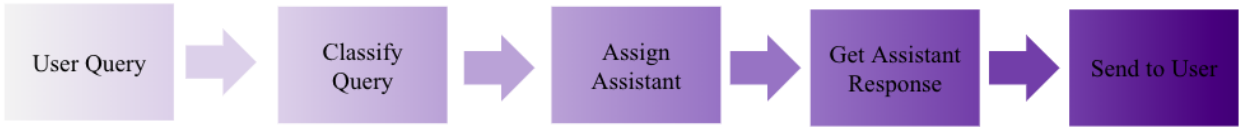
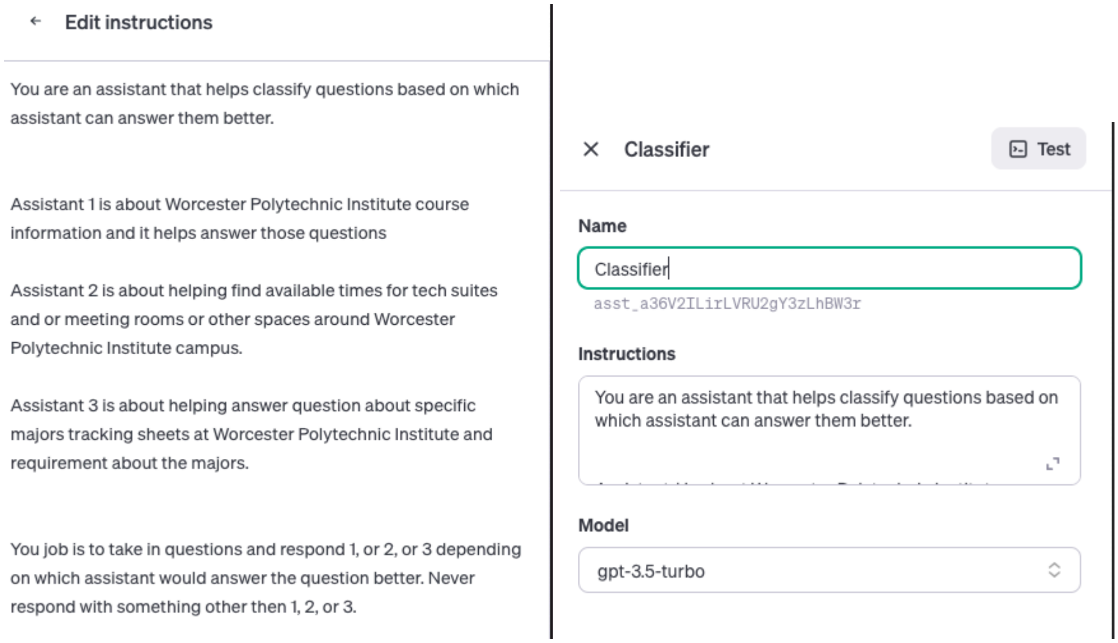


Figure 21: Classifier Flow for User Question.



You are an assistant that helps classify questions based on which assistant can answer them better.

Assistant 1 is about Worcester Polytechnic Institute course information and it helps answer those questions

Assistant 2 is about helping find available times for tech suites and or meeting rooms or other spaces around Worcester Polytechnic Institute campus.

Assistant 3 is about helping answer question about specific majors tracking sheets at Worcester Polytechnic Institute and requirement about the majors.

You job is to take in questions and respond 1, or 2, or 3 depending on which assistant would answer the question better. Never respond with something other then 1, 2, or 3.

Name: Classifier
asst_a36V2ILirLVRU2gY3zLhBW3r

Instructions: You are an assistant that helps classify questions based on which assistant can answer them better.

Model: gpt-3.5-turbo

Figure 22: Example Classifier using an OpenAI Assistant.

3.5.3 Key Differences

This pipeline is similar to the Canopy pipeline as OpenAI does most of the legwork for you. After gathering your API key and desired documents, you can write a Python script to create an assistant for each desired topic and interact with them accordingly. Out of all the pipelines, the most friendly to a beginner would be the OpenAI assistants API.

3.6 Experiment Methodology

In this section, we will discuss how we tested each pipeline to evaluate its performance when answering WPI related questions. We compare the results across pipelines and against and against a model without specific context.

3.6.1 Data Generation for Questions

For the experiment, we used another OpenAI assistant to generate questions based on the context. Using this method we generated 150 questions total (50 for courses, 50 for reservation times, and 50 for degree requirements). Each pipeline has the reference documents to be able to answer the questions.

3.6.2 Experiment Format

To understand the performance of each pipeline and how successfully they handle WPI academic advising information, we designed an experiment. Each pipeline was given 150 questions as defined in 3.6.1, and for each question, we recorded its response along with how long it took to generate. In addition, we gave 150 questions to GPT 3.5 through the OpenAI API to provide a baseline comparison as an LLM that does not have the additional context.

3.6.3 Experiment Evaluation Method

For each question the pipeline answered we gave a human evaluation. Each answer was evaluated based on its Quality, Relevance, and Satisfaction by rating each attribute as a 0 (low rating) or 1 (high rating). Quality was determined by the correctness of an answer, Relevance was determined by the answer's relevance to the original question, and Satisfaction was determined by how satisfactory the answer was. At the end, we calculated scores by summing the 0s and 1s and getting an average. An average of 0 would be the lowest score and an average of 1 would be a perfect score.

3.7 Integration with User Interface

Our project's methodology included integrating Pipeline #3 (OpenAI Assistants API with question classifier) with a user interface and transitioning our backend code to a Flask application. Flask, a Python micro web framework, was chosen for its simplicity, flexibility, and ability to develop web applications quickly. It provides a lightweight solution for creating a web interface that bridges the backend processes with the frontend.

The frontend was developed using an open-source project by GitHub user PandaWhoCodes, which employs Bootstrap and jQuery, ensuring a responsive and user-friendly design [34]. We modified the base frontend code to be more in line with WPI's theme and branding. This choice facilitated an efficient integration with our Flask application, enabling dynamic interaction with the backend functionalities.

The presence of a user interface in a chatbot system is crucial. It transforms complex backend operations into a straightforward, engaging user experience, increasing user satisfaction and engagement by offering immediate responses and interactions. The Flask framework's adaptability, combined with the simple and intuitive design of the frontend, enhances user interaction.

4 Experiments and Results

In this section we will go over the results for our main experiment and results for the front end and back end.

4.1 Pipeline #1 Results

In this section, we will share some of the results and data collected for Pipeline #1 regarding its performance in terms of accuracy, time, and cost among other attributes.

4.1.1 Time and Cost to Run Experiment for Pipeline #1

Table 5: Cost and Time for Pipeline#1 Experiment.

	Gathering Data	Embedding Data	Asking Questions
Time	~5 Minutes	~51 minutes	~ 4.5 minutes
Cost	0	0	~ $0.99

4.1.2 Pipeline#1 Answer Results

Table 6: Scoring Results for Pipeline #1.

	Quality Score	Relevance Score	Satisfaction Score	Combined Score
All Questions	0.8513	0.8918	0.8503	0.8645
Course Questions	0.9200	0.9800	0.9400	0.9466
Reservation Questions	0.7959	0.7959	0.7916	0.7945
Tracking Sheet Questions	0.8367	0.8979	0.8163	0.8503

Table 7: Time Results Pipeline #1.

	Average Time (Seconds)	Standard Deviation (Seconds)	Minimum Time (Seconds)	Maximum Time (Second)
All Questions	1.731	1.459	0.4016	10.87
Course Questions	1.5701	1.144	0.5732	6.318
Reservation Questions	1.912	2.100	0.4016	10.87
Tracking Sheet Questions	1.709	0.8172	0.8321	5.607

4.1.3 Example of Incorrect Answer Pipeline #1

> *User*: What type of course is ID 4000 considered at WPI?
>
> *Pipeline #1*: Based on the provided context, ID 4000 is not mentioned as a course in any of the program tracking sheets for the mentioned majors (MIS, Society, Technology, and Policy Studies, Electrical and Computer Engineering, and Industrial Engineering). Therefore, it is not possible to determine the specific type of course ID 4000 is considered at WPI.

The correct answer should be that ID4000 is a Practicum Seminar that focuses on teacher preparation. Pipeline #1 was not able to retrieve the document that contains the relevant information about ID4000. It seems that the pipeline had the information about different tracking sheets, but none of them had the relevant information. When the class is not specific to any major, this pipeline might not be able to retrieve the relevant document.

4.2 Pipeline #2 Results:

In this section, we will share some of the results and data collected for pipeline #2 regarding its performance in terms of accuracy, time, and cost among other attributes

4.2.1 Time and Cost to Run Experiment for Pipeline #2

Table 8: Cost and Time for Pipeline #2 Experiment.

	Gathering Data	Embedding Data	Asking Questions
Time	~5 Minutes	~1.5 minutes	~ 9.22 minutes
Cost	0	~$0.17	~ $1.36

4.2.2 Pipeline #2 Answer Results

Table 9: Scoring Results for Pipeline #2.

	Quality Score	Relevance Score	Satisfaction Score	Combined Score
All Questions	0.8993	0.9527	0.9048	0.9189
Course Questions	0.9600	0.9600	0.9600	0.9600
Reservation Questions	0.9400	0.9592	0.9400	0.9464
Tracking Sheet Questions	0.7959	0.9388	0.8085	0.8477

Table 10: Time Results Pipeline #2.

	Average Time (Seconds)	Standard Deviation (Seconds)	Minimum Time (Seconds)	Maximum Time (Second)
All Questions	3.763	2.055	1.974	20.53
Course Questions	3.287	1.207	1.974	10.12
Reservation Questions	4.899	2.694	2.076	20.53
Tracking Sheet Questions	3.172	1.415	2.148	12.29

4.2.3 Example of Incorrect Answer Pipeline #2

> *User*: What type of project is associated with the Interactive Qualifying Project (IQP) requirement for Actuarial Mathematics majors?
>
> *Pipeline #2*: The type of project associated with the Interactive Qualifying Project (IQP) requirement for Actuarial Mathematics majors is a Major Qualifying Project (MQP). This MQP must be in ISC with an ISC advisor and meet the specific criteria outlined for Actuarial Mathematics majors. Students interested in pursuing a degree in Actuarial Mathematics should contact Professor Abraham, the Coordinator of the Actuarial Mathematics Program, for more details about this requirement.

The correct answer should be that IQP projects are interdisciplinary teams trying to solve a problem or need that lies at the intersection of science and society. The pipeline describes IQP and MQP as the same project, but they are not. The retrieved documents for this question may contain information about IQP and MQP, as they are both project-based major requirements, but the model should be able to tell that they are different based on their description.

4.3 Pipeline #3 Results:

In this section, we will share some of the results and data collected for pipeline #3 regarding its performance in terms of accuracy, time, and cost among other attributes

4.3.1 Time and Cost to Run Experiment for Pipeline #3

Table 11: Cost and Time for Pipeline #3 Experiment.

	Gathering Data	Embedding Data	Asking Questions
Time	~3 Minutes	N/A	~ 57.8 minutes
Cost	0	0	~ $16.54

4.3.2 Pipeline #3 Answer Results

Table 12: Scoring Results for Pipeline #3.

	Quality Score	Relevance Score	Satisfaction Score	Combined Score
All Questions	0.8322	0.8590	0.8053	0.8322
Course Questions	0.7000	0.7000	0.7000	0.7000
Reservation Questions	0.9000	0.9000	0.9000	0.9000
Tracking Sheet Questions	0.8979	0.9795	0.81632	0.8979

Table 13: Time Results Pipeline #3.

	Average Time (Seconds)	Standard Deviation (Seconds)	Minimum Time (Seconds)	Maximum Time (Second)
All Questions	23.28	5.907	21.88	75.30
Course Questions	22.92	3.984	21.88	42.21
Reservation Questions	23.27	4.903	21.94	42.76
Tracking Sheet Questions	23.65	8.159	21.91	75.30

4.3.3 Example of Incorrect Answer Pipeline #3

User: How many credits can students earn from the BCB Seminar course?

Pipeline #3: Students can earn 3 credits from the AE 5032 Seminar course

The correct answer should be that students do not get any credit for this class because the grading system is Pass/Fail. This pipeline retrieved the information from the wrong course. A possible reason is that AE 5032 is also a seminar course.

4.4 Overall Results

In this section, we will go into the comparison of all the results and the results for the base model without context.

4.4.1 Comparison Between Pipelines and Base Model without Context

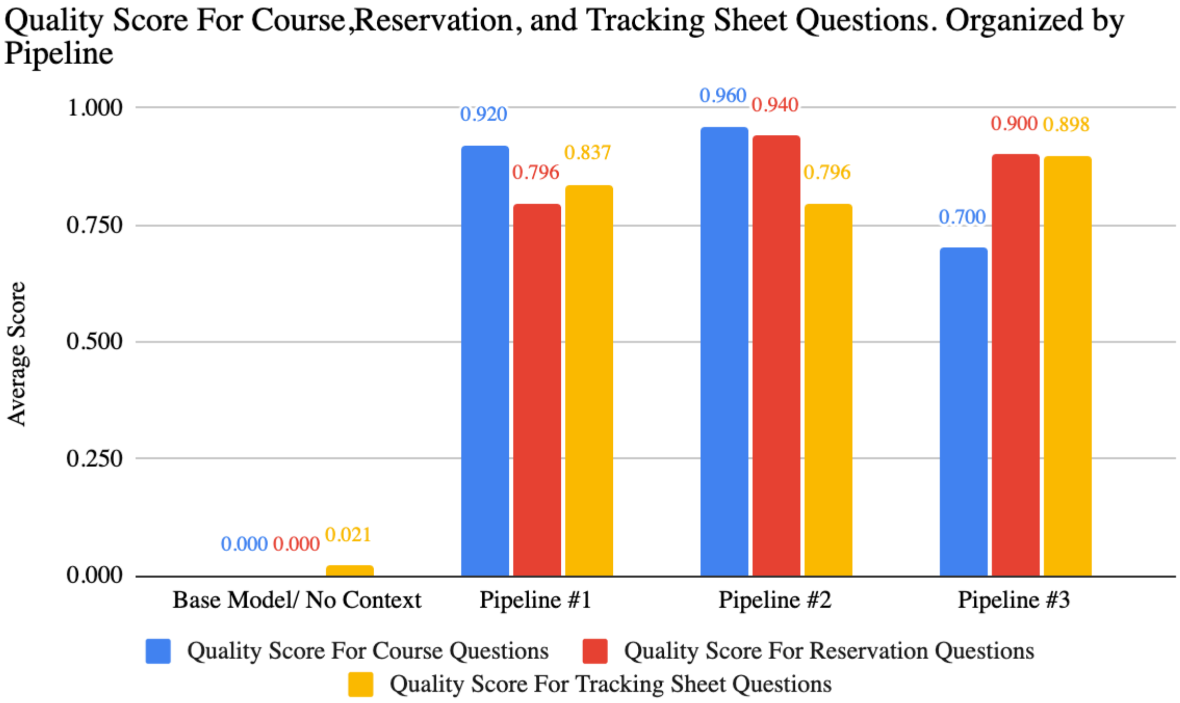


Figure 23: Quality Score Breakdown of Pipelines vs Base Model without Context.

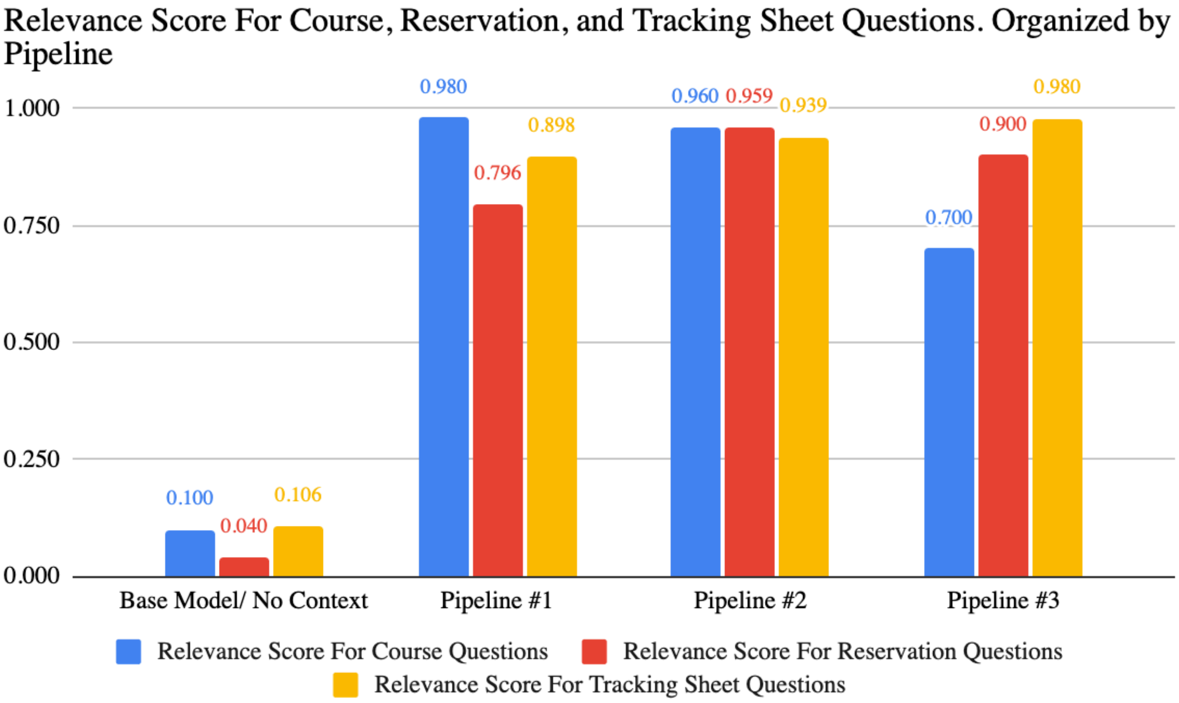


Figure 24: Relevance Score Breakdown of Pipelines vs Base Model without Context.

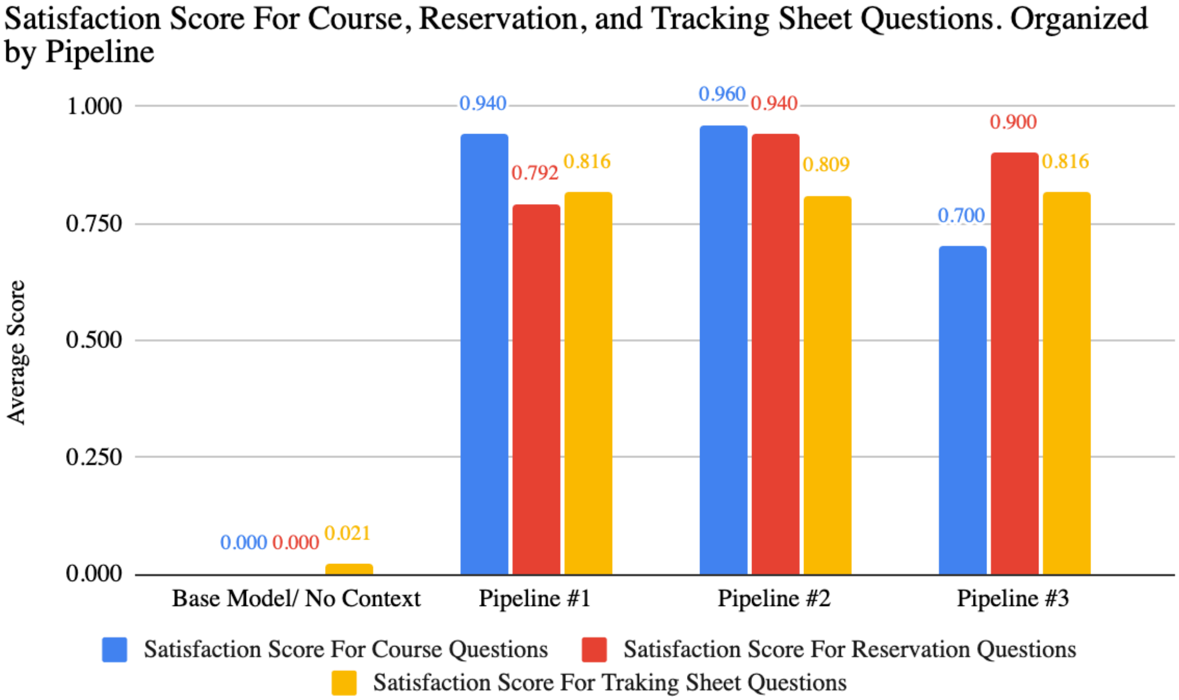


Figure 25: Satisfaction Score Breakdown of Pipelines vs Base Model without Context.

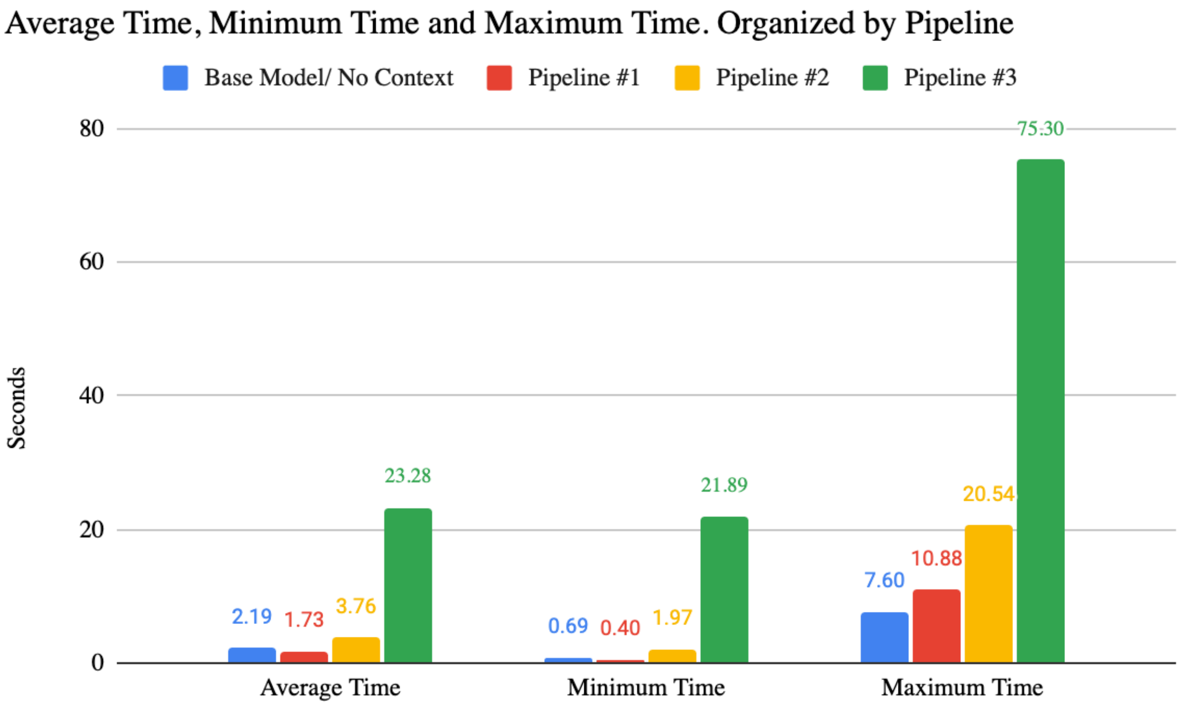


Figure 26: Answer Time Breakdown of Pipelines vs Base Model without Context.

4.4.2 Example of Questions All Pipelines Responded Correctly

Course Question

User: On what dates will the BCB 590 - Special Topics course be offered in 2024?

Pipeline #1: Based on the information provided, the BCB 590 - Special Topics course will be offered from January 10, 2024, to May 1, 2024.

Pipeline #2: The BCB 590 - Special Topics course will be offered from January 10th, 2024, to May 1st, 2024.

Pipeline #3: The BCB 590 - Special Topics course will be offered from January 10, 2024, to May 1, 2024

Reservations Question

User: What is the capacity of Higgins Labs 116 - Classroom?

Pipeline #1: The capacity of Higgins Labs 116 - Classroom is 97.

Pipeline #2: The capacity of Higgins Labs 116 - Classroom is 97 based on the default layout capacity provided in the reservation details.

Pipeline #3: The capacity of Higgins Labs 116 - Classroom at Worcester Polytechnic Institute is 97

Tracking Sheet Question

User: What is the maximum credit allowance for AP courses towards the Humanities and Arts requirement for Actuarial Mathematics majors?

Pipeline #1: Based on the information provided in the Program Tracking Sheet for Actuarial Mathematics majors, only 1 AP credit can be applied to the Humanities and Arts (HUA) requirement. Therefore, the maximum credit allowance for AP.

Pipeline #2: The maximum credit allowance for AP courses towards the Humanities and Arts (HUA) requirement for Actuarial Mathematics majors is 1 AP credit. Only one AP credit can be applied to fulfill the HUA requirement for Actuarial

Pipeline #3: For Actuarial Mathematics majors at Worcester Polytechnic Institute, only 1 AP credit can be applied to the Humanities and Arts (HUA) requirement

Answer For Each Question Given By Base Model Without Context

User: On what dates will the BCB 590 - Special Topics course be offered in 2024?

ChatGPT: I'm sorry, but as an AI language model, I don't have access to specific course schedules for universities. The dates for the BCB 590 - Special Topics course in 2024 would be determined by the university offering the course. I recommend checking with the relevant university or department for the specific dates of the course in 2024.

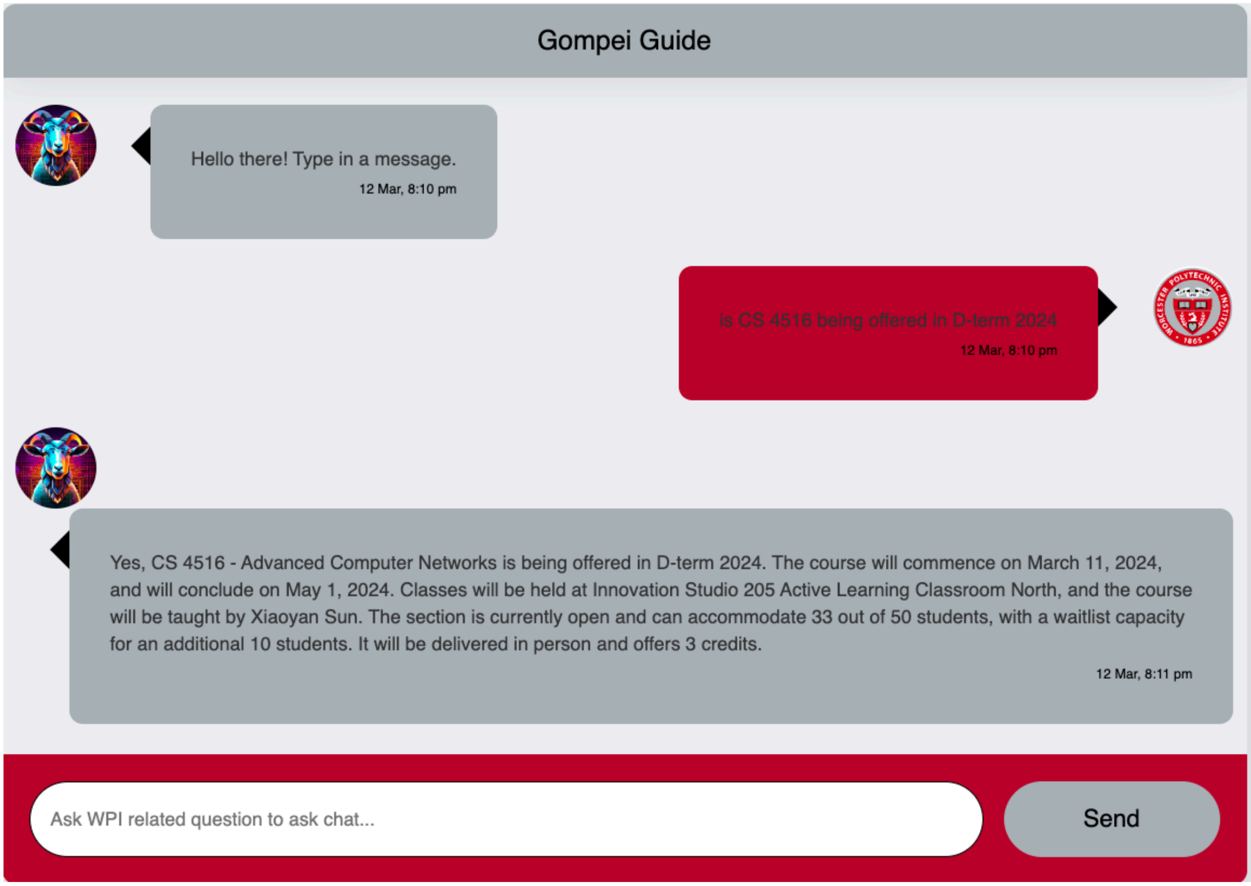
User: What is the capacity of Higgins Labs 116 - Classroom?

ChatGPT: I'm sorry, but as an AI language model, I don't have access to specific information about the capacity of Higgins Labs 116 - Classroom. It would be best to contact the relevant institution or facility directly for accurate information regarding the capacity of that particular classroom.

User: What is the maximum credit allowance for AP courses towards the Humanities and Arts requirement for Actuarial Mathematics majors?

ChatGPT: I'm sorry, but I couldn't find specific information about the maximum credit allowance for AP courses towards the Humanities and Arts requirement for Actuarial Mathematics majors. The credit allowance may vary depending on the university or college you are attending. It is best to consult with your academic advisor or the department responsible for Actuarial Mathematics at your institution for accurate and up-to-date information.

4.5 Frontend/Backend Results:



Hello there! Type in a message.
12 Mar, 8:10 pm

is CS 4516 being offered in D-term 2024
12 Mar, 8:10 pm

Yes, CS 4516 - Advanced Computer Networks is being offered in D-term 2024. The course will commence on March 11, 2024, and will conclude on May 1, 2024. Classes will be held at Innovation Studio 205 Active Learning Classroom North, and the course will be taught by Xiaoyan Sun. The section is currently open and can accommodate 33 out of 50 students, with a waitlist capacity for an additional 10 students. It will be delivered in person and offers 3 credits.
12 Mar, 8:11 pm

Figure 27: Working Implementation of WPI Chatbot using Pipeline # 3.

Figure 27, above, shows the front end of our chatbot that is connected to the back end. The screenshot shows a small portion of a longer conversation.

5 Discussion

In evaluating the performance of various pipelines against a base model without context, it becomes evident that the base model's limitations significantly impact its accuracy, primarily due to its training on static datasets and a general-purpose design that lacks depth in specialized fields such as WPI academic advising information. This resulted in the base model incorrectly answering almost all questions.

Conversely, the specialized pipelines exhibited superior performance, with all achieving over 70% accuracy across evaluated categories. Pipeline #2 was particularly notable for its high accuracy in course (96%) and reservation information (94%), although its performance dipped to 79.6% for tracking sheet information. Pipelines #1 and #3, while slightly less effective for courses and reservations, outperformed pipeline #2 in tracking sheet information with accuracies of 83.7% and 89.8%, respectively. Pipeline #3 uses an Open-AI Assistant that classifies the questions and assigns them to the specific Open-AI Assistant with the corresponding information. Since this classifier can make mistakes, the performance of pipeline #3 is also affected by the accuracy of the classifier.

In terms of user satisfaction, the trends mirrored those of accuracy, with pipeline #2 leading in courses and reservation information. However, the satisfaction scores for pipeline #3 significantly decreased to 81.6%. Relevance scores followed a similar pattern, with pipeline #2 consistently outperforming others, except in specific categories where pipeline #1 excelled in course information.

The analysis of response times revealed pipeline #1 as the fastest, averaging 1.73 seconds, surpassing even the base model. Pipeline #2 had a moderate average time of 3.76 seconds, while pipeline #3 lagged significantly at 23.28 seconds mainly due to the way polling the response was set up the minimum time of 20 seconds.

For pricing, pipeline #1 is the cheapest with a cost of 0.99$ to ask the 150 WPI academic advising questions. Pipeline #2 is slightly more expensive with a cost of $1.36 to ask the questions. For pipeline #3, there is a significant jump in price of $16.54 because of the associated cost of using Open-AI Assistants.

Considering these findings, pipeline #2 demonstrates superior overall performance across various question types. However, the nuanced performances of pipelines #1 and #3 for tracking sheet questions suggest a potential for customizing pipelines based on question specificity in future applications. This tailored approach could enhance the efficacy of LLMs for institutional applications, as evidenced by the promising results of the WPI chatbots. The distinct improvement over the base model emphasizes the value of contextual and specialized training in developing effective chatbot solutions.

6 Conclusion and Future Work

Our project leveraged customized LLMs for WPI, integrating specific institutional data into several models. This effort aimed to surpass the performance of a generic base model by better addressing WPI's unique information requirements. It also investigated how custom pipelines could reduce inaccuracies in LLM responses by addressing knowledge gaps.

The work done through this project successfully demonstrates the potential of custom LLMs to significantly enhance the precision and relevance of responses to academic advising queries for a university such as WPI. By comparing the performance of specialized pipelines with a base model, it is evident that incorporating contextual and institution-specific datasets into LLMs markedly improves their functionality. Among the various pipelines tested, Pipeline #2 (a RAG Model with Pinecone Canopy) emerged as the most effective, showcasing exceptional accuracy in processing course and reservation information, despite a slight decline in tracking sheet queries. This indicates a substantial advancement over the base model, which was constrained by its reliance on static datasets and a lack of specialization.

The findings demonstrate the importance of tailoring LLMs to specific domains to overcome the limitations of generic models. The superior performance of the specialized pipelines not only affirms the feasibility of creating more accurate and efficient LLMs for institutional use but also highlights the potential for these models to be adapted for similar challenges in other settings. This adaptability could significantly improve the way institutions manage and distribute information, making it more accessible and easier to navigate for their communities.

For future work, we recommend exploring further customization of pipelines based on the specificity of queries. The nuanced performance differences among the pipelines suggest that a more targeted approach could yield even higher accuracies and user satisfaction levels. Additionally, integrating real-time data updates and exploring the use of advanced machine learning techniques could further enhance the responsiveness and reliability of these systems. Investigating the scalability of such models to cater to larger institutions or different sectors could also provide valuable insights into the broader applicability of custom LLMs.

Furthermore, continuous feedback loops from users should be incorporated to refine and update the models regularly. This iterative process will ensure the LLMs remain relevant and highly efficient in meeting the dynamic needs of their intended users. By advancing these areas, future research can build on the foundation laid by this project, paving the way for LLMs that are not only highly specialized and accurate but also adaptable to the ever-evolving landscape of information needs across various domains.

Bibliography

[1] "Introducing ChatGPT." Accessed: Feb. 19, 2024. [Online]. Available: https://openai.com/blog/chatgpt

[2] "Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing." Accessed: Feb. 19, 2024. [Online]. Available: https://blog.research.google/2018/11/open-sourcing-bert-state-of-art-pre.html

[3] M. U. Hadi *et al.*, "A Survey on Large Language Models: Applications, Challenges, Limitations, and Practical Usage," preprint, Jul. 2023. doi: 10.36227/techrxiv.23589741.v1.

[4] S. Feng, W. Shi, Y. Wang, W. Ding, V. Balachandran, and Y. Tsvetkov, "Don't Hallucinate, Abstain: Identifying LLM Knowledge Gaps via Multi-LLM Collaboration," 2024, doi: 10.48550/ARXIV.2402.00367.

[5] S. K. Singh, S. Kumar, and P. S. Mehra, "Chat GPT & Google Bard AI: A Review," in *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*, Gorakhpur, India: IEEE, Jun. 2023, pp. 1–6. doi: 10.1109/ICICAT57735.2023.10263706.

[6] Y. Mehdi, "Reinventing search with a new AI-powered Microsoft Bing and Edge, your copilot for the web," The Official Microsoft Blog. Accessed: Feb. 28, 2024. [Online]. Available: https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/

[7] E. Adamopoulou and L. Moussiades, "Chatbots: History, technology, and applications," *Mach. Learn. Appl.*, vol. 2, p. 100006, Dec. 2020, doi: 10.1016/j.mlwa.2020.100006.

[8] "Artificial intelligence prompt engineering as a new digital competence: Analysis of generative AI technologies such as ChatGPT," *Entrep. Bus. Econ. Rev.*, vol. 11, no. 3, pp. 25–38, 2023.

[9] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review," 2023, doi: 10.48550/ARXIV.2310.14735.

[10] J. Fodor, "Intrinsic limitations of GPT-4 and other large language models, and why I'm not (very) worried about GPT-n", Accessed: Feb. 05, 2024. [Online]. Available: https://forum.effectivealtruism.org/posts/6dphu3p8d5mQZEZzk/intrinsic-limitations-of-gpt-4-and-other-large-language

[11] L. Huang *et al.*, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions." 2023.

[12] "ChatGPT." OpenAI, 2024. [Large Language Model]. Available:

https://chat.openai.com/

[13] "Gemini." Google, 2024. [[Large Language Model]]. Available: https://gemini.google.com/app

[14] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, pp. 9459–9474. Accessed: Jan. 25, 2024. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[15] O. Topsakal and T. C. Akinci, "Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast," *Int. Conf. Appl. Eng. Nat. Sci.*, vol. 1, no. 1, pp. 1050–1056, Jul. 2023, doi: 10.59287/icaens.1127.

[16] M. Pelevina, N. Arefyev, C. Biemann, and A. Panchenko, "Making Sense of Word Embeddings." arXiv, Aug. 10, 2017. doi: 10.48550/arXiv.1708.03390.

[17] Q. V. Le and T. Mikolov, "Distributed Representations of Sentences and Documents." arXiv, May 22, 2014. doi: 10.48550/arXiv.1405.4053.

[18] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive Text Embedding Benchmark." arXiv, Mar. 19, 2023. doi: 10.48550/arXiv.2210.07316.

[19] "MTEB Leaderboard - a Hugging Face Space by MTEB." Accessed: Jan. 29, 2024. [Online]. Available: https://huggingface.co/spaces/mteb/leaderboard

[20] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff, "C-Pack: Packaged Resources To Advance General Chinese Embedding." arXiv, Dec. 15, 2023. doi: 10.48550/arXiv.2309.07597.

[21] Y. Han, C. Liu, and P. Wang, "A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge." arXiv, Oct. 18, 2023. doi: 10.48550/arXiv.2310.11703.

[22] R. Schwaber-Cohen, "What is a Vector Database & How Does it Work? Use Cases + Examples | Pinecone." Accessed: Feb. 05, 2024. [Online]. Available: https://www.pinecone.io/learn/vector-database/

[23] "Pinecone Serverless | Pinecone." Accessed: Feb. 10, 2024. [Online]. Available: https://www.pinecone.io/product/serverless

[24] G. Cullen, "Introducing Canopy: An easy, free, and flexible RAG framework powered by Pinecone | Pinecone." Accessed: Feb. 26, 2024. [Online]. Available: https://www.pinecone.io/blog/canopy-rag-framework/

[25] "Sitemap file: https://www.wpi.edu/sitemap.xml?page=1." Accessed: Jan. 31, 2024. [Online]. Available: https://www.wpi.edu/sitemap.xml?page=1

[26] "Program Tracking Sheets | Worcester Polytechnic Institute." Accessed: Jan. 31, 2024. [Online]. Available: https://www.wpi.edu/student-experience/resources/academic-advising/program-tracking-s

heets

[27] "Search Spaces - LibCal - Worcester Polytechnic Institute." Accessed: Jan. 31, 2024. [Online]. Available: https://wpi.libcal.com/spaces

[28] "25live Room Reservations." Accessed: Jan. 31, 2024. [Online]. Available: https://25live.collegenet.com/25live/data/wpi/run/rm_reservations.json

[29] "WPI Course Listing JSON." Accessed: Jan. 31, 2024. [Online]. Available: https://courselistings.wpi.edu/assets/prod-data.json

[30] "OpenAI Platform." Accessed: Feb. 28, 2024. [Online]. Available: https://platform.openai.com

[31] "GPT-4." Accessed: Feb. 28, 2024. [Online]. Available: https://openai.com/research/gpt-4

[32] J. Ye *et al.*, "A Comprehensive Capability Analysis of GPT-3 and GPT-3.5 Series Models," 2023, doi: 10.48550/ARXIV.2303.10420.

[33] "LangChain." Accessed: Feb. 07, 2024. [Online]. Available: https://www.langchain.com/

[34] T. A. Cherian, "PandaWhoCodes/chatbot-frontend." Feb. 19, 2024. Accessed: Feb. 28, 2024. [Online]. Available: https://github.com/PandaWhoCodes/chatbot-frontend

Appendix A

GitHub Repository

We have published a **GitHub repository** containing supplementary materials for our project. There, the following supplementary materials can be found:

- Code for a chatbot using Pipeline #3 (OpenAI Assistants) for the backend, and an open-source chatbot UI for the frontend
- A folder called Pipeline Testing, which contains all the text files that were fed into the customized LLMs as well as spreadsheets with questions, answers, and ratings assessing the performance of each pipeline

Here is the full link as well: https://github.com/Swaggerjee/AdvisorBotWPI.