Report

CHATBOT FOR MS BAIS Program

**Anantha Sai Ram Padala**

business analytics & information systems | ISM6564

2023

**CONTENTS**

**PHASE 0 : Initial Consultation With Client**

**PHASE 1 : Knowledge Database Design Document**

**PHASE 2 : Pinecone Vector Database Implementation**

**PHASE 3 : Technical Report of Chatbot Implementation**

**PHASE – 0**

**Initial Consultation With Client**

**Executive Summary**

The purpose of this PHASE 0 is to provide a detailed account of our initial meeting with Dr. Han Reichgelt, held on September 27, 2023. The meeting, which lasted for approximately 45 minutes, was focused on discussing the prospects of implementing a chatbot using an LLM to create a chatbot designed to handle student queries efficiently. Over this period, we discussed the current pipeline, the options for automation that currently exist, the pain points and redundancies, and what tasks would improve efficiency or reduce latency if automated.

**Business Process Analysis**:

Detail the existing MSBAIS business process landscape, identify bottlenecks, and explore areas where a ChatGPT-like interface could add value.

A diagram of a keyword

Description automatically generated

Fig: Current BAIS Student Query Workflow Diagram

**Workflow:**

The MS-BAIS program at the University of South Florida has a specific workflow for handling incoming queries via their email address "[muma-msbais@usf.edu](mailto:muma-msbais@usf.edu)".

Initially, all emails received are converted into "tickets" using Jira Software, a commonly used ticketing software. These tickets are then categorized based on 'keywords' to determine whether they are of an administrative or academic nature.

Once categorized, each email undergoes a manual evaluation to assess its content and decide the next steps.

Depending on the evaluation, emails are either redirected to the relevant personnel for further action, replied to directly, or, in cases where the query is not clear, a request for additional information is sent back to the sender.

**Pain Points**

* While the process is structured, it has some potential drawbacks. For instance, manual evaluation can be time-consuming, especially if crucial information such as UID isn’t mentioned.
* Most frequent queries are solved by redirecting students to the “Canvas Current Student Portal”.

**Client Needs and Objectives:**

While there will always be a need for manual intervention for specialized queries and administrative decisions, the process could be streamlined by making sure there is an automated method of making sure all the necessary information is made available.

Additionally, repeated and frequent queries with simple default responses can have automated replies.

Another area of interest for automation is the queries, which are solved by redirecting the student to the relevant “Canvas Current Student Portal”.

**Data Assessment**:

The data made available for this project is around 5,000 emails sent through the ticketing system, canned responses, and the data that we collected by scrapping USF MUMA related websites.

Given that emails can contain sensitive information, considerations about data privacy and confidentiality must be addressed; hence, the data provided is anonymous.

Data Limitations:

* Volume: While around 5,000 emails provide a sizable dataset, it may not be enough to capture the full range of student queries.
* Variability: The dataset may have inherent biases based on the time period it covers; for example, more queries about admissions may be received during a specific season.

**Scope of Use**:

* For frequent and straightforward queries, the chatbot can be trained to automatically provide relevant information, such as redirecting students to the Canvas Current Student Portal.
* Machine learning algorithms can be trained to improve upon the existing keyword-based categorization, making it more accurate and efficient.

**Next Steps & Follow-up activities and action items**

Moving forward, the immediate next step is to conduct a thorough data cleaning and preprocessing operation to make around 5,000 emails, canned responses, and scrape information from USF MUMA websites suitable for analysis and model training.

Once the data is preprocessed, a pilot phase should be initiated to train initial machine learning models for automating responses.

We also scheduled a follow-up meeting with Dr. Han Reichgelt to present these initial findings and plans and to discuss any adjustments to the project scope or objectives.

Furthermore, a plan should be laid out for continuous monitoring and improvement, which will include metrics to evaluate the chatbot's effectiveness in streamlining the query management process.

**Conclusion Of PHASE 0:**

In conclusion, the initial meeting with Dr. Han Reichgelt and the subsequent analysis laid a strong foundation for the implementation of a chatbot system aimed at streamlining the MS-BAIS program's query management process at the University of South Florida. The availability of a sizable dataset of around 5,000 emails, canned responses, and scraped data along with a detailed understanding of the current workflow and its pain points, provides us with a unique opportunity to create a meaningful impact.

Through careful data assessment, targeted machine learning algorithms, and strategic integration into existing systems, the project aims to reduce manual workload by building a ChatBot to answer student queries, increase efficiency, and enhance the overall experience for both students and administrative staff. By adhering to the outlined next steps and follow-up activities, the project is well-positioned to meet its objectives in a timely and effective manner.

**PHASE – 1**

**Knowledge Database Design Document**

**Data Sources Identification**

Here are the identified data sources for augmenting LLM (Language Model) prompting for the MSBAIS Chatbot:

* **Jira Ticketing System (CSV File):** Extracts current issues, resolutions, and updates from the Jira ticketing system.
* **Canned Responses Document (Potential Queries):** Contains predefined responses for common queries or topics often asked by applicants or reapplicants, enabling quick and accurate responses.
* **MSBAIS-FAQ Document:** Includes a compilation of frequently asked questions (FAQs) about the MSBAIS program, serving as a reference for informative responses.
* **MSBAIS Course Catalog:** The provided syllabus has been used to extract this information which is the Docx format. The catalog includes syllabi from multiple professors, providing a clear academic roadmap spanning three semesters. This empowers students to align their studies with career goals in business analytics and information systems, facilitating informed decision-making and efficient program navigation.
* **MSBAIS Current Students Canvas Course Modules:** Extracts information from course modules within the MSBAIS Canvas platform, covering topics such as Course Advising and registration, Graduate Teaching/Research Assistantships, CPT Internships, etc. To extract this Information we need to scrape the webpage.
* **Other Academic and Administrative Topics**: Covers various academic and administrative aspects, including Reduced Course Load (RCL) for international students, Practice Center projects, independent study, Academic integrity, Transfer credits, Academic probation, Professional development certificates, and guidelines on how to apply for graduation.

These diverse data sources collectively enrich the knowledge base of the MSBAIS Chatbot, enabling it to respond accurately and comprehensively to a wide array of inquiries, thereby enhancing the overall user experience for both prospective and current MSBAIS students. Regular updates and maintenance of these data sources are essential to ensure the chatbot remains up-to-date and reliable.

**Data Schema Development**

Pinecone is a vector database that can be used to store and search for embeddings of data from a variety of sources. Embeddings are a type of data that represent the semantic information. Vector databases offer optimized storage and querying capabilities for embeddings.

To use Pinecone, we first need to create an index. An index is a collection of records, each of which has a unique ID, an array of floats representing a vector embedding, metadata (key-value pairs), and sparse vectors.

Below is the figure of how a record looks in a vector database:

A screenshot of a phone

Description automatically generated

Once you have created an index, you can start adding data to it. For example, to add data from a Jira Ticketing system which is a json data file with questions and answers, we would first need to create chunks of data initially. A chunk is a collection of related records. Once after creating chunks of data, we can generate embeddings for each chunk and then upsert the embeddings into the Pinecone index created.

The following schema is used to store and search for embedding in a Pinecone index for all of our data sources:

ID: unique ID for each record

Values: embedding vector values

Metadata: text (e.g., the question or response from the Jira ticket)

A close-up of a white background

Description automatically generated

This schema allows to search for related information about msbais course, registration, or about university details etc. For example, you could use a similarity search to find documents that are similar to a given query document. You could also use the metadata fields to filter your search results.

**Vectorization Techniques**

Vectorization techniques are essential for converting textual data into numerical vectors, allowing machine learning models to process and analyze the text. The choice of vectorization technique depends on the specific task, dataset, and desired outcomes. Here are some widely used vectorization techniques suitable for converting textual data into numerical representations:

1. **TF-IDF (Term Frequency-Inverse Document Frequency):**

* Represents word importance in a document relative to a corpus, emphasizing less common words within a document.
* Ideal for text classification, clustering, and information retrieval tasks.

1. **Word2Vec:** Maps words to dense vectors, preserving semantic relationships and enabling applications like sentiment analysis and language modeling.
2. **Doc2Vec (Paragraph Vectors):** Extends Word2Vec to learn fixed-length feature representations for variable-length pieces of text, useful for document-level analyses.
3. **BERT (Bidirectional Encoder Representations from Transformers):**
   * BERT is a transformer-based model that generates contextualized word embeddings.
   * Contextual embeddings capture word meanings based on surrounding context, improving representation accuracy.
   * Ideal for a range of NLP tasks including text classification, question-answering, and named entity recognition.
4. **GloVe (Global Vectors for Word Representation**): Learns word vectors by analyzing global word co-occurrence statistics, offering a global perspective of word meanings and relationships.
5. **Tf-Idf weighted Word Embeddings:** Enhances word embeddings using TF-IDF weighting, giving more weight to important words, improving word representations.
6. OpenAI Ada:

* OpenAI Ada is a highly advanced language model based on the GPT-3.5 architecture, known for its ability to generate human-like text and comprehend context effectively.
* Ada can be used for vectorization by encoding text into numerical representations, making it suitable for various NLP tasks, including text summarization, language translation, chatbot development, and more.

The Document Embedding API, from open API, is the best approach relative to TF-IDF, one Hot encoding & word2Vec approach that we have seen because of the below:

1. Contextual Embeddings:

* OpenAI Ada: Ada generates contextual embeddings, considering the entire context of a sentence or paragraph. It understands and represents words based on their surroundings, capturing nuances and context effectively.
* TF-IDF, One-Hot Encoding, Word2Vec: These techniques generate fixed embeddings without considering context, potentially losing nuances and subtle meanings.

1. Semantic Understanding:

* OpenAI Ada: The model demonstrates a deep understanding of semantics, allowing it to generate embeddings that encapsulate the underlying meaning of the text.
* TF-IDF, One-Hot Encoding, Word2Vec: While effective, these approaches may fall short of representing semantic relationships accurately.

1. Adaptability to Various NLP Tasks:

* OpenAI Ada: The Ada model is versatile and can be fine-tuned for specific NLP tasks, making it adaptable and suitable for a wide array of applications.
* TF-IDF, One-Hot Encoding, Word2Vec: While these techniques are versatile, they may not be as easily adaptable to different NLP tasks without further modifications.

**PHASE – 3**

**Pinecone Vector Database Implementation**

**Data Collection:**

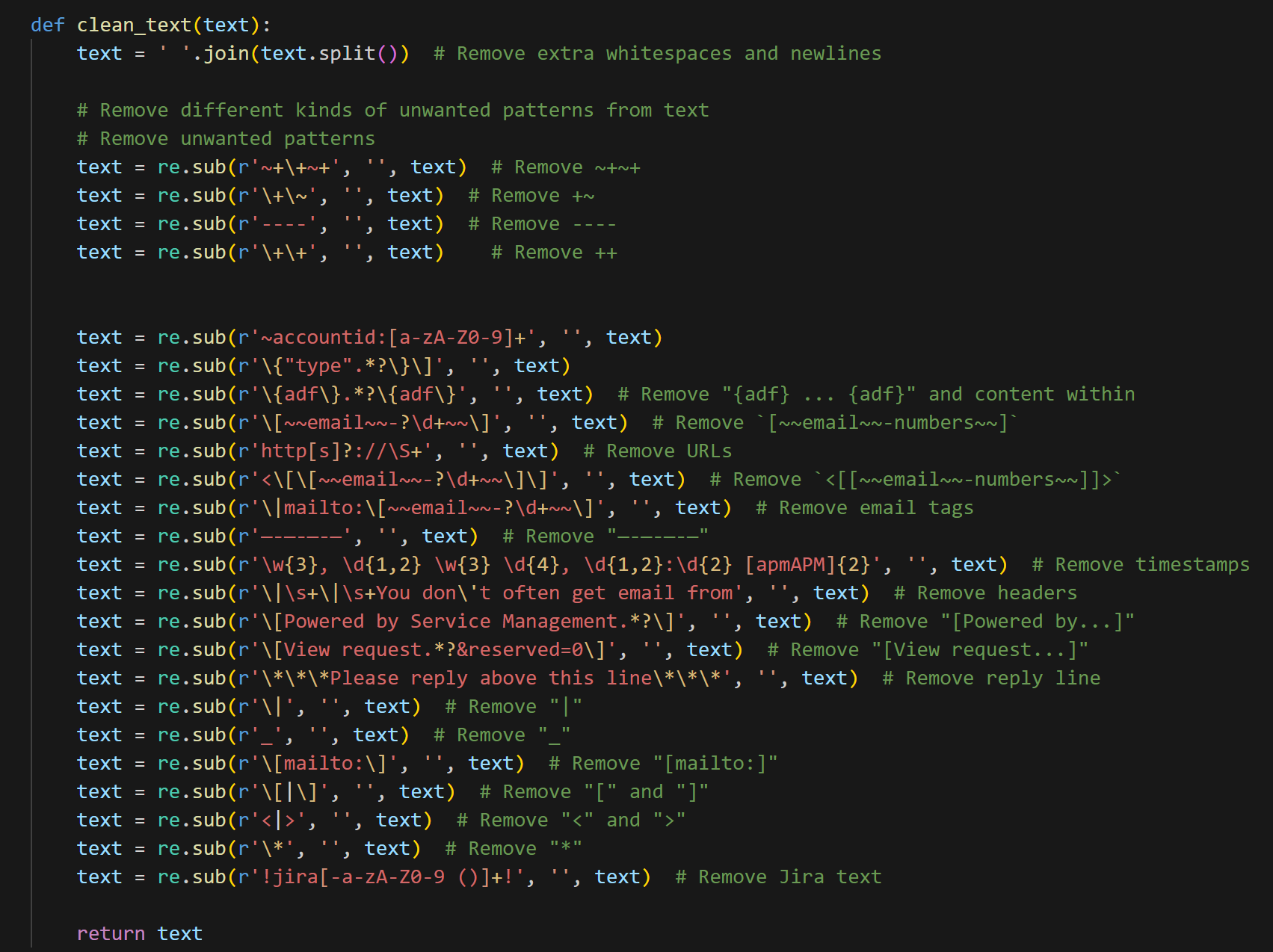
The data from below provided data sources has been collected in Phase I.

* + 1. Jira ticketing system
    2. PDFs, Scraping documents: Canned responses, course catalog, and Canvas modules are covered in these.

**Data Preprocessing:**

The Data obtained from Phase I Data sources like the Jira ticketing system has been cleaned as needed by removing the patterns, newline characters, etc.

Below is the sample code for cleaning JSON data:



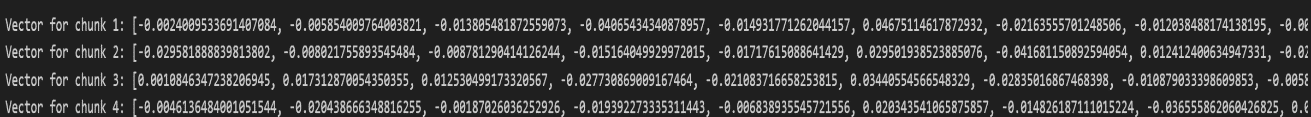
**Vectorization:** The Preprocessed data has been converted into chunks and further converted into vectors to store it in the vector database.

Sample code for creating chunks:

A screen shot of a computer program

Description automatically generated

Vectors for each chunk id provided below for sample data:



**Vector Database Implementation**

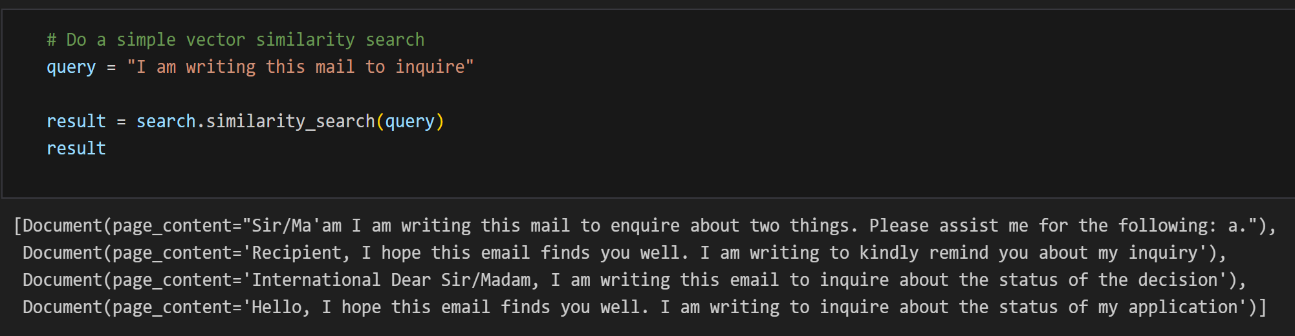
Below is the sample code for Vectorization and Pinecone database implementation of one such process(Jira ticketing system):



**Vector Database Testing:**

In the Database testing, we will check the efficacy of our pinecone vector database for the provided queries and see if it is giving the matching information.

Sample queries & their output:



**A black screen with white text

Description automatically generated**

**PHASE – 4**

**TECHNICAL REPORT OF CHATBOT IMPLEMENTATION**

In this phase, we want to develop a chatbot to assist incoming and current students of the University of South Florida, Master of Science in Business Analytics and Information Systems (MSBAIS) program. The primary goal is to provide information about the program, syllabus, courses, and any inquiries, and direct the students to the appropriate resources. In cases where the chatbot cannot answer a question adequately, it will route the inquiry to the designated email address, muma-[msbais@usf.edu](mailto:msbais@usf.edu), where university staff will provide further assistance.

**Technical Architecture:**

This chatbot leverages data sources such as past email conversations and USF Canvas modules, using LangChain for NLP processing and Pinecone for vector storage. Here's an overview of the overall technical architecture:

**User Interface (UI):**

* The chatbot's front end consists of a user-friendly interface accessible to incoming and current MSBAIS students. Users interact with the chatbot by typing messages or questions.

**Data Sources:**

* The chatbot has access to multiple data sources, including:
  + Past email conversations: Historical emails containing valuable information about program-related topics.
  + USF Canvas modules: Canvas is used to manage course-related information, including course advising, registration, assignments, and resources.
  + Other program-related data: This includes information about Graduate Teaching/Research Assistantships, CPT Internships, Reduced Course Load (RCL) for international students, Practice Center projects, independent study options, Academic integrity guidelines, Transfer credits, Academic probation information, Professional development certificates, and guidelines on how to apply for graduation.

**Data Chunking and Vectorization:**

* The unstructured data from these sources is processed into structured "chunks" that represent discrete units of information. These chunks are designed to capture key information.
* LangChain helps convert these chunks into vectors, which represent these units in a mathematical format. This allows for easier storage and retrieval.

**Pinecone Vector Database:**

* The vectors generated by LangChain are stored in the Pinecone vector database. Pinecone is an example of a vector similarity search service that allows for efficient searching and retrieval of similar vectors.
* Pinecone enables the chatbot to quickly find relevant information from the chunks and vectors generated from the data sources.

**Data Flow Diagram:**

**A diagram of a company

Description automatically generated**

The data flow diagram represents different data sources like PDF, and JSON files which have information about the MS BAIS program, courses, syllabus, etc that we have collected for our knowledge database. In the next step, we created chunks of the information from data sources. These chunks look like documents. These are converted to embeddings and stored in a Pinecone vector database. Now, we will create a prompt template to input the user prompt and do a similarity search to get the documents from the Pinecone vector database and give this content to the chatgpt-3.5-turbo to get the refined responses.

**Component Diagram:**

**A screenshot of a diagram

Description automatically generated**

**LangChain:**

LangChain is a comprehensive library offering a suite of tools and utilities tailored for the efficient handling of language models, text embeddings, and text processing tasks. It simplifies a range of operations, including the development of chatbots, managing document retrieval, and conducting question-answering tasks by seamlessly integrating key components like language models, vector stores, and document loaders.

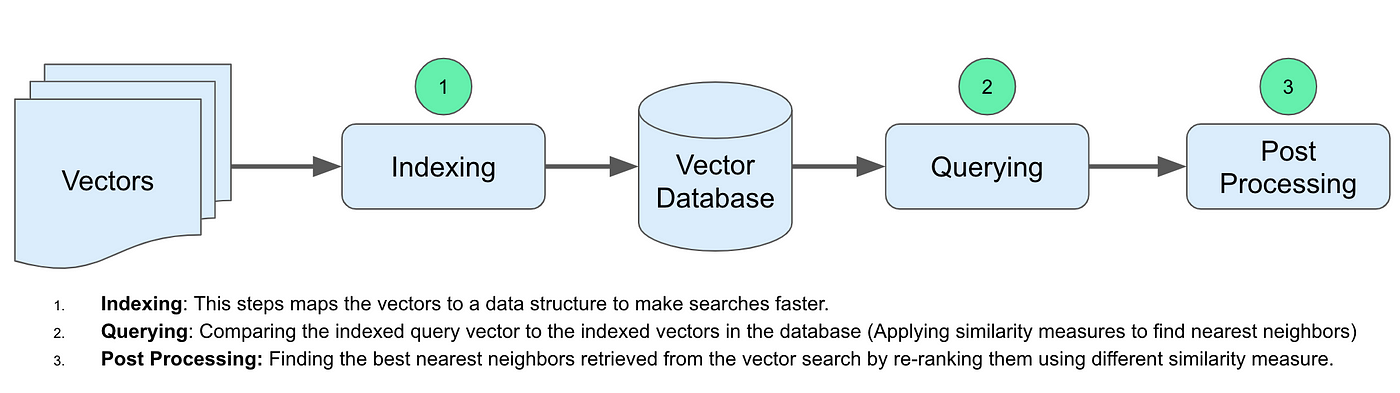
The below figure describes the different components we used for the ChatBot implementation.

A diagram of a project

Description automatically generated with medium confidence

**Pinecone:**

Pinecone simplifies the integration of long-term memory for high-performance AI applications by offering a managed, cloud-native vector database with a user-friendly API, eliminating infrastructure complexities. It excels at delivering low-latency query results, even at the scale of billions of vectors. Leveraging vector embeddings, Pinecone caters to applications reliant on language models, generative AI, and semantic search, allowing AI systems to maintain enduring memory for intricate tasks. These vector databases, like Pinecone, are designed to efficiently store and query embeddings, a task that conventional scalar-based databases struggle with due to the complexity and scale of such data. Pinecone's indexes store records featuring unique IDs and dense vector embeddings, bridging the gap between traditional databases and optimized vector indexes.



**Streamlit:**

Streamlit is an open-source Python library that simplifies the process of creating web-based user interfaces. It is widely used for building interactive, data-driven applications, and can be effectively leveraged as the frontend component of our chatbot system for the University of South Florida MSBAIS program.

**ChatGPT:**

ChatGPT, powered by the GPT-3.5 Turbo model, is a cutting-edge AI language model designed by OpenAI. This model excels in understanding and generating natural language text, making it a remarkable tool for various applications. Here, we are using it by inputting the prompt, user question, and chunks to get the response for the user questions.

**\*Code snippets to show how the Python, Lang chain, and OpenAI APIs are integrated are provided in the documentation section.**

**User Interface Design:**

**Wireframes or mock-ups:**

Below we are providing the sample UI of our chatbot.

**Before asking a question:**

**After asking a question:**

**User experience flow:**

A diagram of a process

Description automatically generated

We implemented Streamlit to create the user interface for our chatbot. When a user initiates a conversation by posing a question, the chatbot first inquires whether the user is currently enrolled as an MSBais student or a prospective one. In the case of a current student, the chatbot requests the student's ID. Subsequently, it proceeds to process the user's query by conducting a similarity search against the pre-stored vectors within Pinecone, which then yields relevant documents, the initial query, and a prompt template. This information is presented to ChatGPT and this gives a refined response to the user and asks if the response provided answers the user's question or not. If yes, the user can continue with any other queries, or else it will trigger an email to muma-msbais@usf.edu

**Test Cases:**

**TC 1:**

**Pass/ Fail:** Pass, as the chatbot gave information relevant to the question asked for prospective student category.

**TC 2:**

**Pass/ Fail:** Pass, as the chatbot gave information relevant to the question asked for prospective student category.

TC 3:

**Pass/Fail:** Pass, as the chatbot gave information relevant to the question asked for current student category.

**Documentation:**

Here in the documentation, we will see the code snippets of the Data processing, Using Lang chain for vectorization, and upsert these vectors into our chosen Vector Database – Pinecone. Based on the user Query a similarity search is made in the vector database retrieving all the closest matches, which is fed to the chat Open AI object that would generate a response to the query. What if the chatbot is not able to answer a particular question? it would generate an email and send an email to the concerned team.

Step 1: The first step would be importing all the libraries that are used in our application.

A screenshot of a computer program

Description automatically generated

Step 2: Create an environment file(.env) to store configuration variables and sensitive information that our application needs to function correctly. This file is crucial for separating configuration details from our code, enhancing security, and simplifying the management of environment-specific settings.

A screen shot of a computer

Description automatically generated

Step 3: Loading and preprocessing of data

**JSON data:**

3a. Loading the Json data

A screen shot of a computer program

Description automatically generated

3b. Cleaning/preprocessing

A screen shot of a computer code

Description automatically generated

**Web Scrapped pages:**

3c.Loading of scrapped pages:

**A screen shot of a computer program

Description automatically generated**

**Syllabus files:**

3d. Loading of syllabus files:

**A screen shot of a computer program

Description automatically generated**

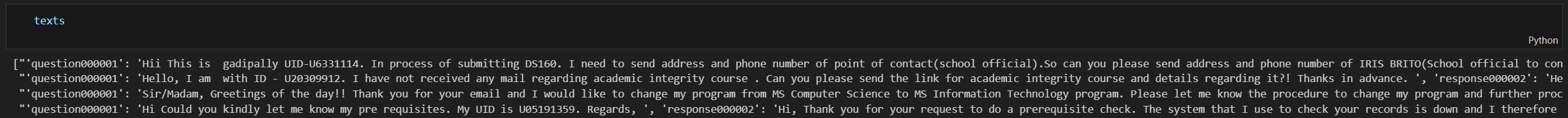
**Canvas modules files:**

3e. Loading the canvas module files:

A screen shot of a computer program

Description automatically generated

**Outputs of Json, PDF data**



A computer screen with white text

Description automatically generated

Step 4: Chunking

The split\_text function is used to chunk the PDF, scrapped, canvas, and json data.

A computer screen shot of text

Description automatically generated

Step 4: Pinecone vector database: Pinecone vector database is initialized to store the vectors.

A screen shot of a computer program

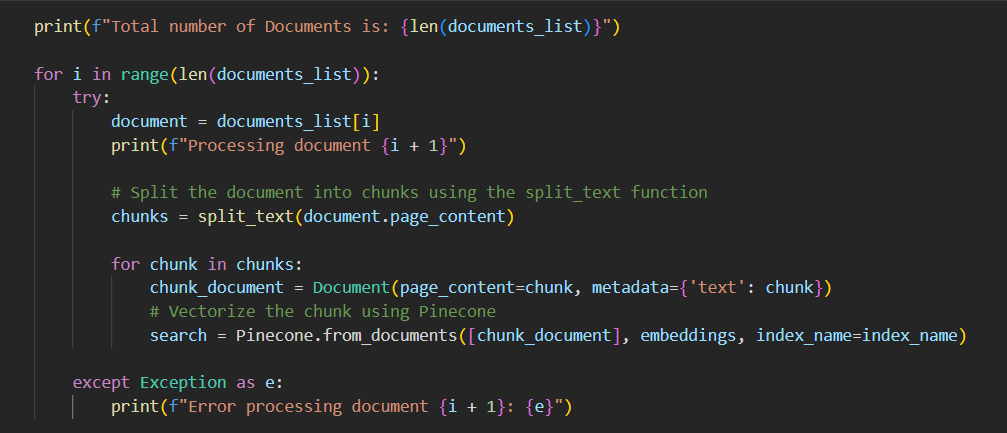
Description automatically generated

Step 5: Vectorization

Using the OpenAI Embeddings component from Lang Chain to convert page content from documents into vectors. This instance of OpenAI Embeddings converts text data into numerical vectors and stores them in pinecone.

A computer screen with text

Description automatically generated



The above steps helps us to store all the vectors of data sources in Pinecone.

In the chatbot.py file to create the chatbot application using the Streamlit application

Step 6: Vector Database initialization

Again initialized the pinecone.init() function which is the Pinecone API client with the provided credentials and environment settings, allowing us to make API calls to Pinecone for tasks like inserting and querying vectors.

A computer screen shot

Description automatically generated

Now we have our knowledge database ready, and we can use this for our chatbot application.

Step 7: The main program initializes the necessary libraries, loads the CSS file, and initializes the session state variables. It then creates a Streamlit UI with three containers:

Loading CSS styles:

A computer screen shot of a black background

Description automatically generated

* The chat history is displayed in the Streamlit app, including messages from both the AI and human users.

A screenshot of a computer program

Description automatically generated

* The chat input field container displays a text field where the user can enter their prompt.

A screenshot of a computer

Description automatically generated

* The debugging information container displays information about the token count and the Lang Chain conversation chain.

A screen shot of a computer code

Description automatically generated

This function handles the behavior when the submit button is clicked. It involves interaction with OpenAI and Pinecone for chat responses and similarity searches.

A screenshot of a computer program

Description automatically generated

initialize\_session\_state function ensures that the session state variables for chat history, token count, and the conversation with the OpenAI model are set up when the Streamlit application starts. These variables are used to maintain the state of the chat and keep track of tokens used during interactions.

A screen shot of a computer program

Description automatically generated

Finally, How our Chatbot interface looks like: