**SMART AI RETRIEVAL ASSISTANT**

**PROJECT REPORT**

**CS790 - CLOUD COMPUTING**

**FALL 2023**

**TEAM CLOUD9ERS**

MUSTAFA KHAN   
[Khan72@uwm.edu](mailto:Khan72@uwm.edu)  
ZUHAIB SHAKEEL KHAN   
[zskhan@uwm.edu](mailto:zskhan@uwm.edu)  
PHANI RAM TEJA RAVIPATI   
[ravipati@uwm.edu](mailto:ravipati@uwm.edu)  
LAKSHMI SAI KARTHEEK SURABHI   
[lsurabhi@uwm.edu](mailto:lsurabhi@uwm.edu)

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S. NO** | **CONTENT** | **PAGE NO** |
| 1. | Abstract | 03 |
| 2. | Architecture | 04 |
| 3. | Features | 05 |
| 4. | How It Works | 12 |
| 5. | Try It Yourself | 20 |

**ABSTRACT**

The Smart AI Retrieval Assistant, nicknamed SARA, is an AI tool which would read your personal documents like a resume or class notes and answer questions from within those documents. To accomplish this, it uses 3 main components:

1. A Vector Database
2. An Embedding Function for the Vector Database
3. A LLM for generative answering

For this project, the **ChromaDB** Vector Database was used, OpenAI’s embedding model ‘**text-embedding-ada-002**’ was used for embedding and for intelligent generative answering, we use OpenAI’s ‘**gpt-3.5-turbo-16K**’ model.

These types of chatbots are also referred to as Retrieval Augmented Generation (RAG) chatbot. They retrieve some data as context, in our case, the vector store, they augment it as a prompt by using the retrieved information as context, and then use the prompt to generate an answer.

We provided an early concept version of SARA to students from multiple majors studying in UW-Milwaukee, and we got a positive response and a need for such product in the interest of the students.

The most positive response came from Business Majors, who had a plethora of Power point files which contained source material, and they found SARA useful in answering their questions.

**ARCHITECTURE**

A diagram of a server

Description automatically generated

SARA’S AWS EC2 ARCHITECTURE

SARA is hosted on 4 AWS EC2 instances. The architecture is designed in such a way that the admins can only access the servers via a bastion host to configure them. The 4 servers are divided into 2 subnets. The public subnet has the bastion host and the webserver which the user accesses. The private subnet will have the MySQL Database and the ChromaDB vector database.

**The Bastion Host**The only way to access the instances for SSH access is to do via the bastion host. Admins need to set up the bastion as a proxy server with agent forwarding, so when the admin tries to login to a server, it will be forwarded via the bastion.

**The Web Server**

The Webserver is the heart of the application. It contains most of the logic which is required to run the application. It contains a flask application, which contains multiple API endpoints and the front-end files, which were made using HTML, CSS, and vanilla JavaScript. It also has multiple Python scripts which handle chat functionality, Upload functionality and Database Operations.

**The MySQL🐋 Server**

The MySQL server instance will have tables in it which will contain the user login information, the chroma collection information and the uploaded files documentation. It helps in seamless integration of the front-end with back-end.

**The Chroma Server**

The chroma server is the instance which will contain the chroma vector database. This instance will store all the document vector embeddings which we have uploaded to the vector database. Like MySQL, this responds to requests in a client-server style.

**FEATURES**

SARA has quite a few features in it, they are the following:

1. User authentication (login and register)
2. ChromaDB collections operations.
3. Upload Documents to a ChromaDB collection.
4. Chat with files inside the ChromaDB collection.

**USER AUTHENTICATION**

When the user first access SARA, they are redirected to a login page. If the user exists, they can log in using their credentials, else they must register a new account.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Upon logging in, the user will be redirected to the homepage.

A screenshot of a computer

Description automatically generated

**CHROMADB COLLECTIONS OPERATIONS**

In the above picture, if the user clicks on collections, the user will see few collections related operations.  
A screenshot of a computer

Description automatically generated

**CREATE COLLECTION**

The user can create a collection in ChromaDB, in which documents can be stored. A collection is like a table in MySQL.  
A screenshot of a computer

Description automatically generated

**LIST COLLECTION**In this feature, the user can see what collections has been created.

A screenshot of a computer

Description automatically generated

**SEE DOCUMENTS IN COLLECTION**This feature shows the documents uploaded to a collection.  
A screenshot of a computer

Description automatically generated

**DELETE COLLECTION**

This feature deletes a collection and all the documents inside of it.

A screenshot of a computer

Description automatically generated

**UPLOAD DOCUMENTS TO A CHROMADB COLLECTION**

This feature is used to upload file(s) to a collection. For this concept product, only .pptx and .pdf files are supported  
A screenshot of a computer

Description automatically generated

For the sake of demonstration, let’s upload the below .pdf file  
A screenshot of a computer

Description automatically generated

**CHAT WITH FILES INSIDE THE CHROMADB COLLECTION**

We finally chat with the files we uploaded to the Chroma Collection. We first select the collection we want to chat with in the sidebar, and then ask SARA questions from the documents we uploaded to it.

A screenshot of a chat

Description automatically generated

To Prove that SARA, does not utilize information from the internet and utilizes information only from within the uploaded documents, we can test it with a random question, and she will reply ‘I cannot find the answer’.

A screenshot of a chat

Description automatically generated

**HOW IT WORKS**

SARA has 4 main modules. They are:

1. User authentication
2. Collections
3. Upload
4. Chat

Although limited right now, these can be further expanded to provide a more stable and comprehensive experience with document management and chat functionality. The application is deployed using Flask.

**USER AUTHENTICATION**

The user authentication module consists of 2 features, Login and Register. No one can directly access SARA’s feature, and user must login or register first. When registering, the credentials of the user are stored in a MySQL table called **‘user’**.  
  
A black screen with white text

Description automatically generated  
  
It has 5 rows:

1. Id – an incremental id row
2. public\_id – a UUID generated at registration unique to each user.
3. name – name of the user
4. email – email of the user
5. password – password of the user which is encrypted with SHA-256.

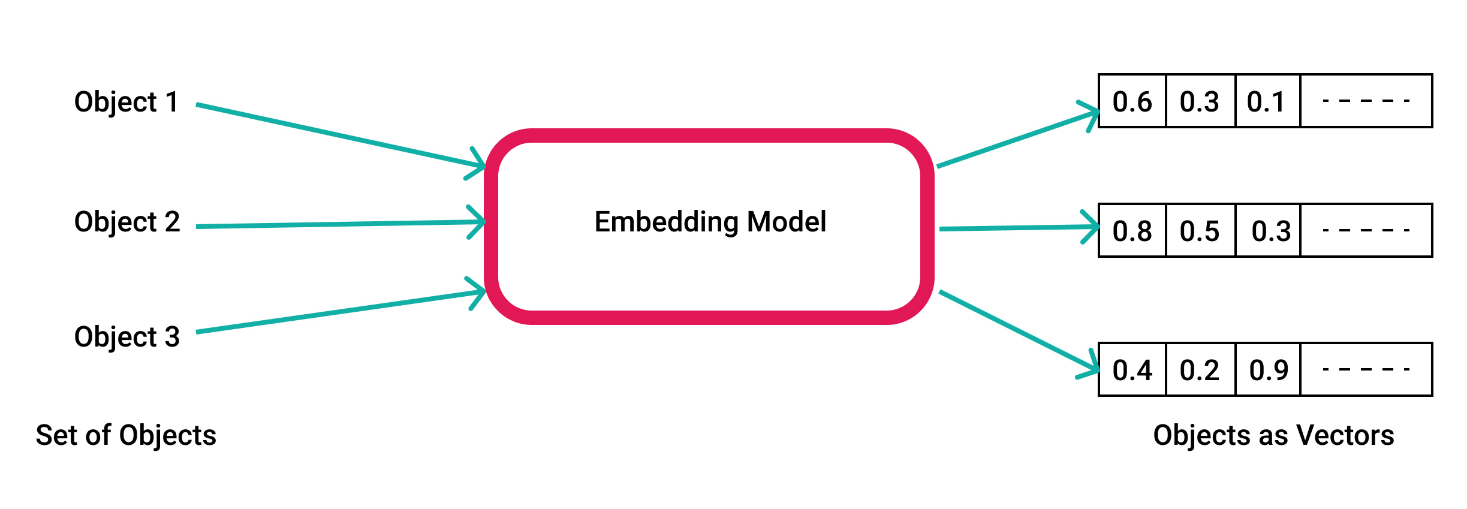
When logging in, details are verified from the user table. To maintain session security, we make use of JWT objects. Once a user logs in, the user details are stored as flask session objects and the JWT is valid for 30 minutes or until the user logs out. This helps us maintain user separation among users.

**THE COLLECTIONS MODULE**

For us to fully understand how collections module works, we first need to understand Vector embeddings, how they are stored in vector databases and how ChromaDB comes into play here.

Vector Embeddings:

In natural language processing (NLP), vector embeddings are numerical representations of words or phrases in a continuous vector space. These embeddings capture semantic relationships between words and are essential for various NLP tasks, such as language modeling, machine translation, sentiment analysis, and more. The idea behind vector embeddings is to represent words in a way that preserves their meaning and relationships with other words. Traditional methods, like one-hot encoding, represent words as sparse vectors with a dimension for each unique word in the vocabulary. For our project, we have used OpenAI’s vector embeddings from the **‘text-embedding-ada-002’** model.



Vector Embeddings

Vector Databases:

A vector database stores data points as vectors, which are numerical representations of objects. These databases excel at quick similarity searches using mathematical operations and are widely used in applications like recommendations systems, image recognition, and natural language processing. Vector databases often integrate vector embedding, learned data representations capturing intricate relationships. Embeddings encode data into meaningful numerical forms, aiding algorithms in understanding complex data relationships.  
In simple terms, it is a database for our vector embeddings.

A diagram of a diagram with dots and lines

Description automatically generated

Vector Embeddings represented in multi-dimensional space.

Chroma Vector Database:  
ChromaDB is an open-source vector database. We use this database to store our vector embeddings. It is also directly recommended by OpenAI as the go to vector database. ChromaDB provides the concept of **collections.** A collection is like a table in MySQL DB. You can perform operations on a collection inside the database and not the whole database itself. Inside the collections, we will be storing vector embeddings, which are in turn, the transformed file data.

Now that we know about vector embeddings, Vector databases and chroma, let’s talk about collections module.

CREATE COLLECTION:

When we create a collection, 2 requests are sent. The first request is sent to the chroma server, which will initialize a collection of that name inside the vector database and the second request is sent to the MySQL database, to update the ‘**collections’** table in the Database. We will use this collections table later to for the ‘List Collections’ feature.

A screen shot of a computer program

Description automatically generated

This table stores the logged in users name, UUID and the collection name they created the collection with.

DELETE COLLECTION:

Delete collection works the same as create collection. We first send a request to the Chroma server to delete all the data inside that collection and that collection itself and then we send a request to the MySQL server to delete that collection name from the collection table.

A diagram of a web server

Description automatically generated

Create and Delete Collection

LIST COLLECTION:

List collections is simple, we just make a call to the MySQL table and fetch the contents of the ‘collections’ and display it for that user. We filter using UUID.

LIST DOCUMENTS:

List Documents is also simple and like the List Collection module. We just make a call to the uploaded \_files table (which will be discussed in the later part of this document) and display all the documents related to a particular collection.

A black square with blue arrows

Description automatically generated

List Collection and List Documents

**THE UPLOAD MODULE**

SARA’s upload module currently only supports only .pptx and .pdf files, upto 100MB only. The document upload works in the following way:

* The user selects the documents which are to be uploaded with the file browser.
* Once the user selects documents and hits upload, the documents are sent to the webserver.
* The documents are then given to the Document Parsers based on their extension. If it’s a PDF file, then it’s handled by the **PyPDF** document parser and if it’s a PPTX file, then it is handled by the **UnstructuredPowerPointLoader** Document parser. Both document parsers are provided by the Langchain LLM framework.
* The Document after passing through the respective document parsers are converted, have all their text extracted and converted into a huge string.
* We split this huge string recursively into multiple smaller strings of 1000 characters each and store it in a list, we call this smaller string a ‘**chunk’**. This is accomplished using a Character Splitter called **RecursiveCharacterTextSplitter** which is also provided by Langchain.
* We then combine a chunk with a unique vector-id and some relevant metadata and prepare our vector.
* Finally, we upload the vector embeddings to chroma, and we upload the details of the uploaded file to the uploaded files table in MySQL.

A screen shot of a computer program

Description automatically generated

The ‘uploaded\_files’ table has 5 rows:

1. Id – an auto-incrementing id field
2. file\_name – the name of the uploaded file
3. UUID – a unique UUID for the file
4. collection\_name – the name of the collection where the file was uploaded.
5. timestamp - the time at which the file was uploaded.

A diagram of a diagram

Description automatically generated

The Upload Process

**THE CHAT MODULE**

The Chat module is used to communicate with the documents uploaded in the collection. It works in the following way:

* The user first selects the collection to ask the questions from.
* The user then asks a question, which is sent to the webserver.
* The webserver then converts that question into a vector embedding using “text-embedding-ada-002” model and sends it to chroma server.
* Chroma receives the embedded question and performs a similarity search inside that collection.
* In SARA, we use cosine distance metric for similarity search, and we return the top 5 most relevant chunks, related to that question.
* We use these chunks as context and stitch a prompt by combining the provided question and context from chroma. Click on the below file to see an example prompt.  
  
* We then send the prompt to OpenAI’s ‘gpt-3.5-turbo-16k’ API, which intelligently forms a response and sends it back, which we display back to the user in the chat screen.

A diagram of a web server

Description automatically generated

**TRY IT YOURSELF.**

SARA has 2 versions which can be tried:

1. SARA CLI : a local CLI application to be run on local machine.
2. SARA WEB-UI – A structured web application to be deployed on multiple servers.

It can be accessed at [MUSTAFAKHAN1999/SARA: Smart AI Retrieval Assistant (github.com)](https://github.com/MUSTAFAKHAN1999/SARA)

The Web and CLI applications are separated into two branches.

SARA is also hosted on <https://www.cloud9ers.cloud>

(If the website is unreachable, it might be because the instances are turned off. A quick email to [Khan72@uwm.edu](mailto:Khan72@uwm.edu) can have them turned on).

**SARA CLI:**

To run SARA CLI, you need the following:

* Python v3.8.10 only
* OpenAI API key

SARA CLI has no authentication and just runs directly with the features.

**SARA WEB-UI:**

To run SARA WEB-UI, you need to first provision 4 AWS instances according to the above project architecture.

1. Web Server – t2.medium
2. Bastion – t2.micro
3. MySQL – t2.medium
4. Chroma – t3.small ( [☁️ Deployment | Chroma (trychroma.com)](https://docs.trychroma.com/deployment#simple-aws-deployment))

The instructions for installation will be present in the readme.md file in the repository.