SAP Technical Assessment

Technical Design, Ideas, Evaluations, and Architecture for Generative AI COE

Tu Weile

Table of Contents

Problem Statement

Problem Statement: Generative AI COE

Problem: Build a Generative AI model that is able to introduce myself to others.

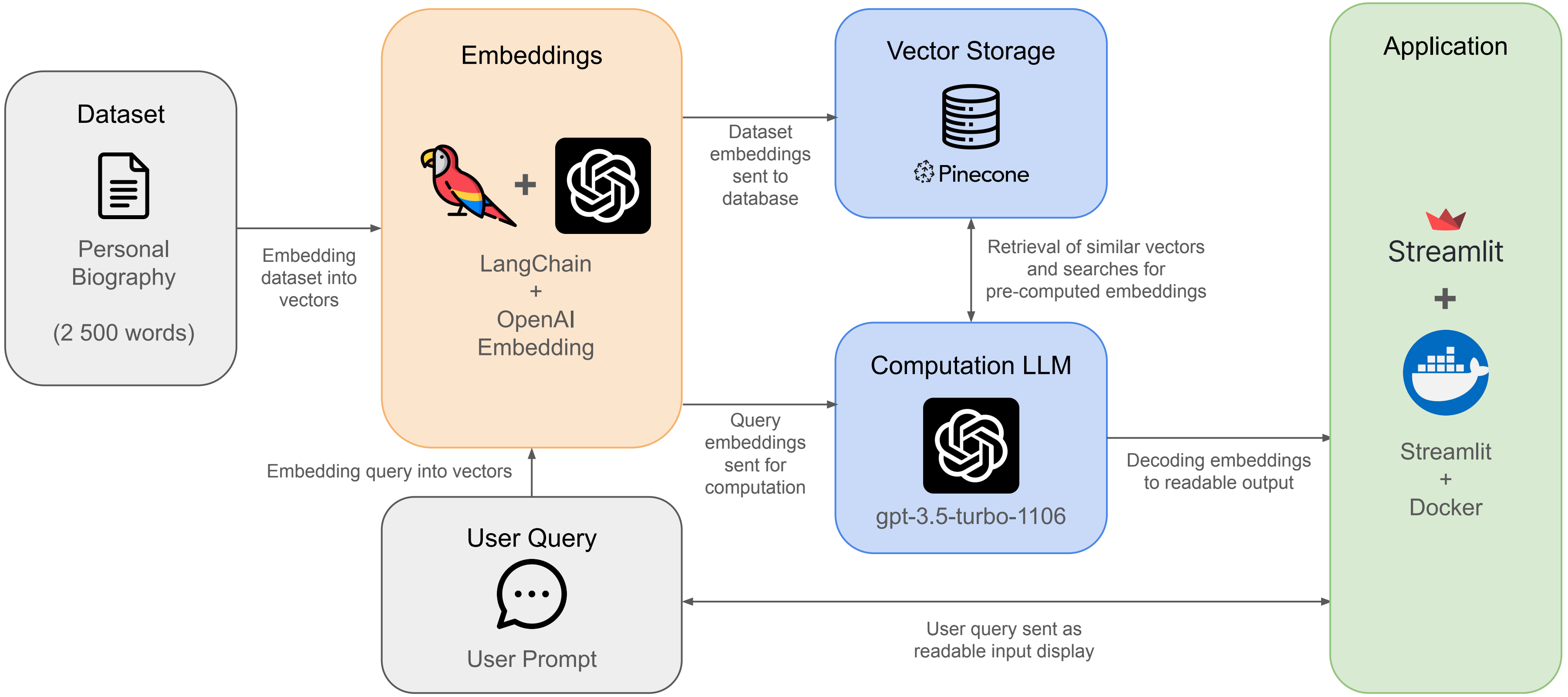
From the problem statement, there were several issues to address, mainly:

1. Which LLM model to use? Proprietary (OpenAI, Google Vertex) or open models?
2. What frameworks and languages to choose?
3. Which vector database to store the embeddings? In-memory or cloud?
4. What kind of data to give to the model? Contextual understanding and generation?
5. Scalability and resource considerations and requirements?
6. User experience and customizable features for end users?

These issues will be answered in the architecture and the rationale sections.

Technical Architecture

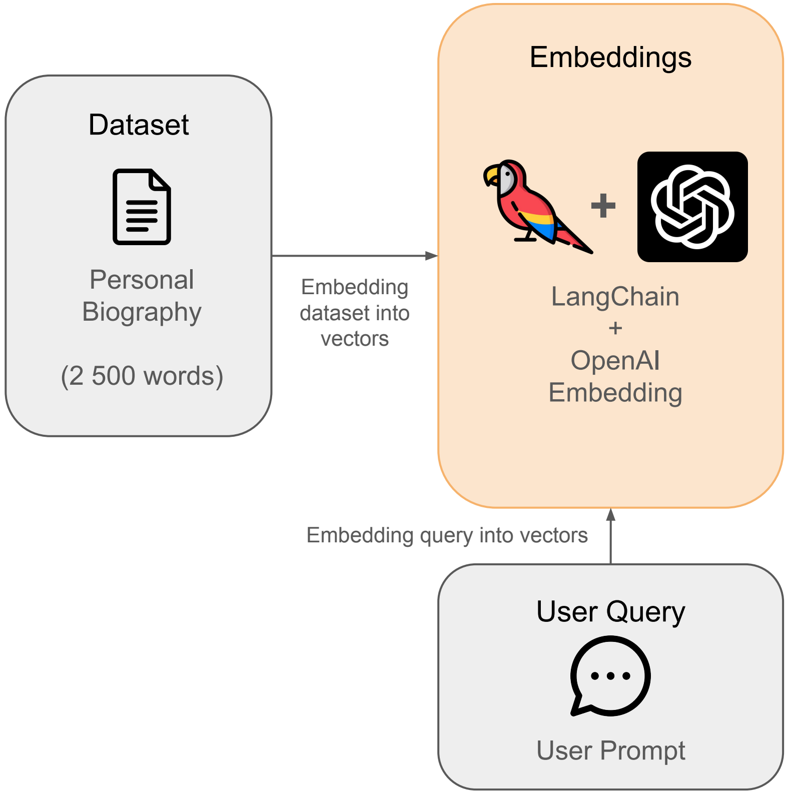
Technical Architecture


Dataset
Personal Biography
(2 500 words)
Embedding dataset into vectors
Embeddings
LangChain + OpenAI Embedding
Dataset embeddings sent to database
Vector Storage
Pinecone
Retrieval of similar vectors and searches for pre-computed embeddings
Computation LLM
gpt-3.5-turbo-1106
Query embeddings sent for computation
Embedding query into vectors
User Query
User Prompt
Decoding embeddings to readable output
User query sent as readable input display
Application
Streamlit
Streamlit + Docker

Architecture Workflow

A dataset comprising a personal biography and any queries made to the application are embedded using OpenAI Embeddings via LangChain library.

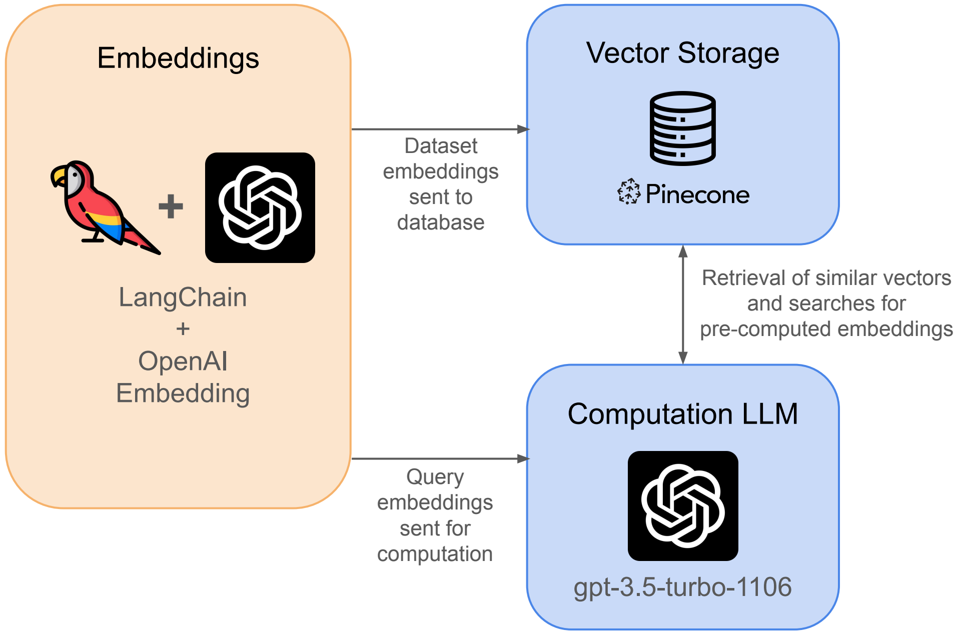
The embeddings are then stored in a vector database, which in this case is Pinecone.



Architecture Workflow

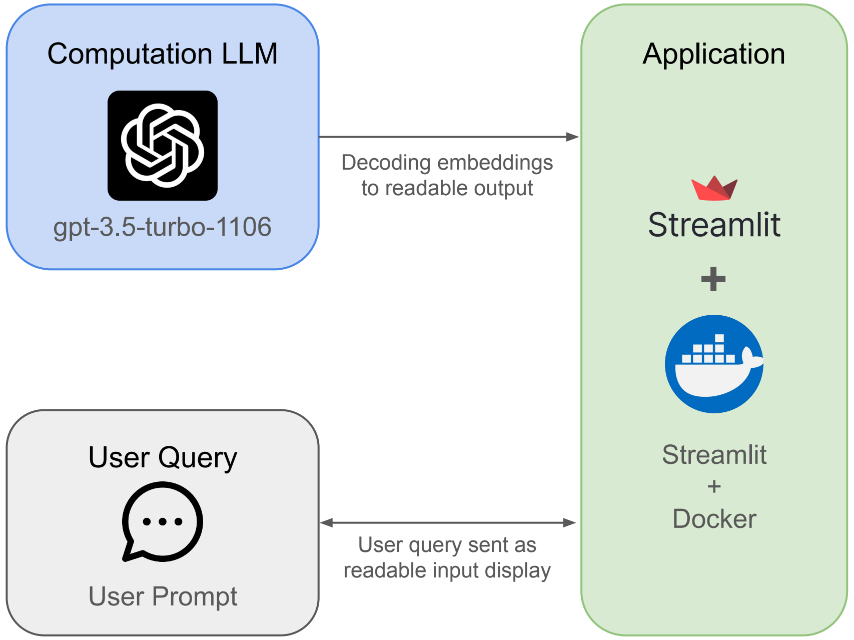
Embeddings from the dataset (personalized biography) is split into chunks (default chunk size: 512, default chunk overlap: 32) before being stored in a vector database (Pinecone).

Embeddings from the query (user input) is sent to the computational LLM (gpt-3.5-turbo-1106), which in turn, will retrieve the k most similar chunks (default k value: 5) from the vector database before being computed together with the query to formulate the response.


Embeddings
LangChain
+
OpenAI
Embedding
Dataset embeddings sent to database
Vector Storage
Pinecone
Retrieval of similar vectors and searches for pre-computed embeddings
Query embeddings sent for computation
Computation LLM
gpt-3.5-turbo-1106

Architecture Workflow

Upon the generation of the response embeddings by the computation LLM, the embeddings are then decoded into a readable output alongside with the user input to a front end application (Streamlit). The application is also dockerized to allow for easier installation.



Rationale for Design and Architecture

LLM Model Selection

1. Which LLM model to use? Proprietary (OpenAI, Google Vertex) or open models?

For LLM model selection, there were several considerations to be made:

A) Proprietary LLM models (OpenAI, Google Vertex):
   - Proprietary LLM models provide higher model performance and quality as these models often come with dedicated support and updates.
   - User-friendly interfaces, APIs, as well as technical documentations allow for easier implementation and integration with existing applications.
   - However, these models may not be ideal for air gapped systems (no internet connection) and may not be suitable for sensitive data as computation processing is done on external servers.

LLM Model Selection

1. Which LLM model to use? Proprietary (OpenAI, Google Vertex) or open models?

B) Open-source models

- Open source models often provide transparency in their architecture, which is useful in customizing the model according to specific use cases.
- These models can also be hosted in air gapped systems, ensuring privacy and that sensitive data will not be compromised during computation processes.
- However, these models are computationally expensive to host and may also require substantial data preprocessing or additional fine-tuning to match up to the performance of proprietary models. They also lack continuous updates and maintenance.

LLM Model Selection

1. Which LLM model to use? Proprietary (OpenAI, Google Vertex) or open models?

From these considerations, utilizing a proprietary model such as OpenAI is ideal given the following rationale:

- Developmental effort to match performance of open-source models to proprietary models is significantly higher than vice versa; may not be ideal.
- Time and resource considerations (2 weeks) is also a factor of consideration; open source model may take a long time to generate a response with limited memory unlike proprietary models, which requires a short period of time given its huge resources.

With this in mind, using OpenAI API and their models is the best suited for this case.

Framework selection of LLM application

2. What frameworks and languages to choose?

For languages and frameworks, we can use OpenAI API directly to generate the response that we want. However in this application, using third-party frameworks such as LangChain may be ideal for this application given the following considerations:

- LangChain framework offers more customization and compatibility over both proprietary and open-source models as opposed to using OpenAI API directly. Its standardized interface makes it ideal in providing agents and tools to fit multiple use cases.
- Should there be a better performing model, LangChain allows for more control over the models used and the developmental effort for experimentation is lesser due to its in-built embeddings and retrieval methods within the LangChain library.

Vector database selection of LLM application

3. Which vector database to store the embeddings? In-memory or cloud?

For vector databases, we can use in-memory or cloud databases to store our vector embeddings. There are several considerations to be made:

A) In-memory vector database (ChromaDB)

- Data retrieval is done locally, which can be ideal in building air-gapped applications. It can also be ideal against cloud databases in terms of latency.
- In-memory vector databases are constrained by the available resources and memory, which may not be ideal for large datasets.
- Furthermore, in-memory vector databases can be lost during application failure as it lacks adequate redundancy mechanisms, unlike cloud vector databases.

Vector database selection of LLM application

3. Which vector database to store the embeddings? In-memory or cloud?

B) Cloud vector databases (Pinecone)

- Cloud vector databases are hosted on providers with massive amounts of resources and memory, making them ideal for large datasets. With redundancy measures such as backup cloud vector databases, the risk of data loss is reduced.
- However, the cost effectiveness is lost as it costs money to maintain these vector databases. This may not run cheaply with increasing dataset sizes.
- There is also the added concern of network latency and possible data security and privacy considerations, since this can impact the overall performance of the model and sensitive data should preferably be stored on-premises.

Vector database selection of LLM application

3. Which vector database to store the embeddings? In-memory or cloud?

From these considerations, using a cloud vector database such as Pinecone may be ideal given the following rationale:

- In enterprise architecture designs, application uptime and availability is of paramount importance. While cost and latency are valid considerations, having to rebuild the embeddings after a catastrophic data loss event will be time consuming.
- The data added into the cloud database is not confidential in any nature as this is public information that can be found on my LinkedIn, so a cloud database is more appropriate.

With this in mind, using Pinecone cloud vector database is applicable for the application.

Dataset of LLM application

4. What kind of data to give to the model? Contextual understanding and generation?

For the data inputted into the model, I wrote up a ~2 500 words biography describing myself, my experiences and skills into the model. While it may not be complete, it serves to augment my professional persona in my LinkedIn profile.

Upon receiving a user prompt, the user prompt is embedded into vectors and with that reference, the vector database is queried for the **k** most similar chunks (default is set to 5) to the query before being sent together with the user query to generate an appropriate response. The data chunks and overlap (512, 32) are also set respectively to ensure that the responses generated have enough contextual understanding from the user prompt.

Scalability and resource management

5. Scalability and resource considerations and requirements?

Given that the frontend is hosted on Streamlit and that consideration must be given to accommodate for different OS and make installation easier, using Docker is optimal for this use case. The rationale for dockerizing the application is as follows:

- By creating a Dockerfile, this helps to encapsulate the application and its dependencies together, making the application platform-agnostic. Dependency management is streamlined to provide the user a more optimal experience during installation.
- Version control and scalability allows the principle of iterative improvements in the application, as well as defining the environment and configuration of the application.
- Docker complements with container orchestration through Kubernetes and Docker Swarm to ensure easy scaling, load balancing, and management of multiple instances.

User customization for LLM application

6. User experience and customizable features for end users?

Placing API keys from OpenAI and Pinecone as hard-code is bad practice as it exposes the application and API endpoints to attacks and unwanted operations. Furthermore, users should be able to customize the chunks size, overlapping, and k value, to obtain a varied answer for every user query given. For this, the following are implemented:

- API keys are entered into the Streamlit interface through a separate PDF file locked by a password. This ensures that the API is not compromised in any way.
- Customized numerical inputs for users to change the chunk size, chunk overlaps, and k value in order to generate a more detailed or short response depending on their user requirements and cases.

Potential Improvements

Potential Improvement #1: Data scraping

As the datasets required for this application requires personalized information in order to build a Generative AI application that emulates personas to others, it might be ideal to implement data scraping on various platforms in order to capture all of the relevant information related to the professional profile of the individual.

This includes, but is not limited to: LinkedIn, Facebook, and other social media platforms.

From these newly incorporated datasets, it is possible to capture a more holistic overview of the individual of how they behave and what they do. It will also improve the quality of the response generation, as there will be more similar embeddings to the user prompt for the model to choose from and generate its response.

Potential Improvement #2: Vector database optimization

While Pinecone is a good vector database to use for this application, several improvements can be made to enhance the performance of the vector database. This is because while vector search is great for similarity search embeddings, vector databases may require additional setup and resources to maintain that particular use case. Furthermore, Pinecone does not support any other use cases such as structured data search and analytics.

In order to increase the scope of the use cases, alternative databases that support vector database operations such as ElasticSearch and Weaviate may be optimal for this application. Weaviate provides contextualized embeddings for natural language understanding which can be useful to understand the relationship between data points, whereas ElasticSearch allows for a broader range of search and analytics use cases.