Generative AI Assistant Project

Vinay Sai Rajapu – 002870410

Pavuluri Jaswanth - 002870640

1. Project Overview

The Generative AI Assistant project is designed to provide users with a conversational interface capable of answering questions specifically related to the domain of Generative Artificial Intelligence.
The goal was to create an interactive, reliable, and user-friendly system that leverages state-of-the-art AI models to deliver intelligent, context-aware answers.

The system integrates three core components:

- Embedding generation using OpenAI's text-embedding-3-small model
- Contextual retrieval from Pinecone's vector database
- Natural language generation using OpenAI's gpt-3.5-turbo model

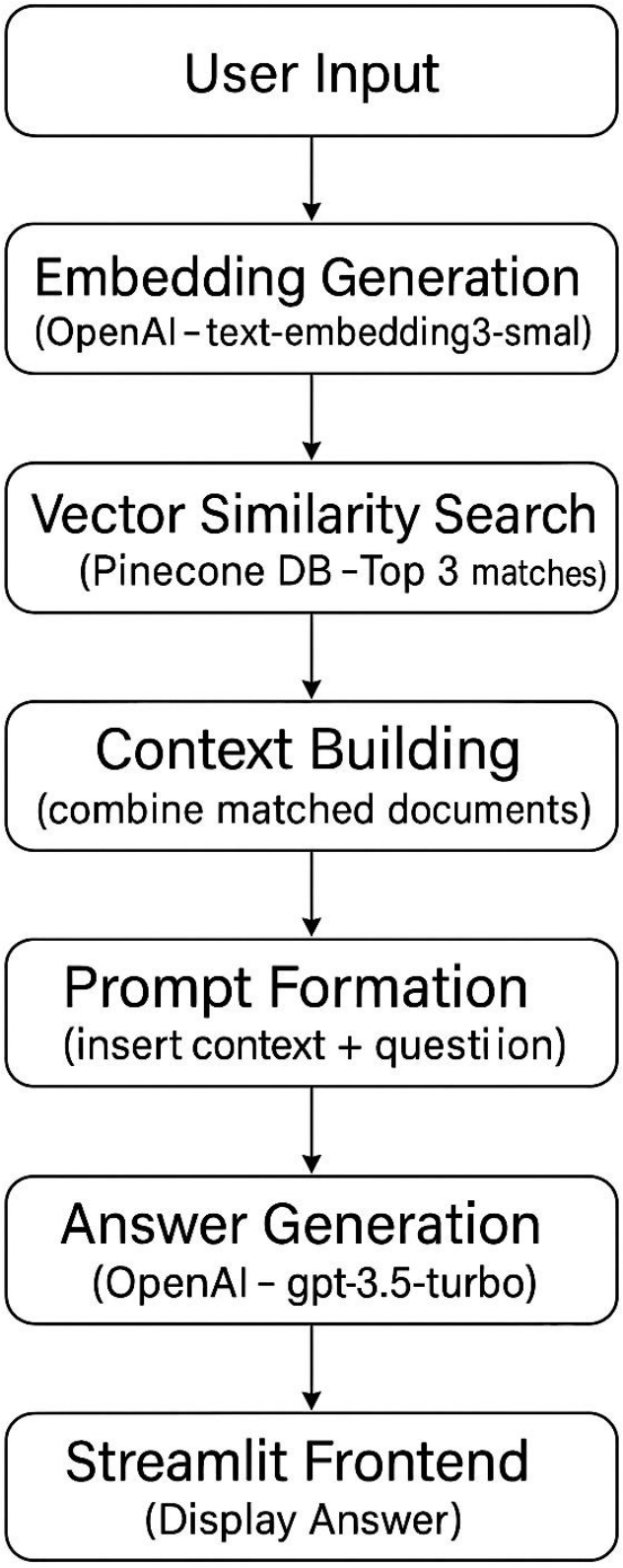
The frontend of the application was built using Streamlit, which provides a smooth web experience for interacting with the chatbot.
The project involved implementing prompt engineering techniques, retrieval-augmented generation (RAG), and managing external APIs securely through environment variables and secret management practices.

Overall, the assistant was tailored specifically to maintain focus on Generative AI topics, ensuring that responses remain relevant, accurate, and helpful within the defined scope.

---

2. Architecture Diagram

Overall Flow:


User Input
Embedding Generation
(OpenAI – text-embedding3-smal)
Vector Similarity Search
(Pinecone DB – Top 3 matches)
Context Building
(combine matched documents)
Prompt Formation
(insert context + questi ion)
Answer Generation
(OpenAI – gpt-3.5-turbo)
Streamlit Frontend
(Display Answer)

Components:

- User Interface (UI): Built with Streamlit for interaction.
- Embedding Generator: OpenAI's embedding API for converting text to vectors.
- Vector Database: Pinecone for efficient retrieval of relevant information.
- Language Model: OpenAI ChatCompletion API for generating natural language responses.

---

3. Implementation Details

Frontend Development:

- Streamlit was used to develop a highly intuitive, minimalistic web interface.
- Dynamic switching between Light and Dark modes was incorporated to improve user experience.
- Custom CSS was applied for better visualization of chat messages in bubble format.

Backend Processing:

- Each user query is first converted into a dense vector representation using OpenAI's embedding model.
- Pinecone's vector similarity search retrieves the most contextually relevant chunks from the knowledge base.
- The retrieved chunks are assembled into a detailed prompt.
- OpenAI's chat model (gpt-3.5-turbo) then generates a contextually accurate response based on the prompt.

Technologies Used:

- Streamlit: Frontend and backend coordination
- OpenAI API: For both embeddings and conversational responses
- Pinecone Vector Database: For efficient semantic search and retrieval
- Python: Primary programming language for logic and integration
- Streamlit Cloud: For deployment and hosting

Deployment Strategy:

- All environment variables were managed securely using .env files locally and secrets.toml in the cloud.
- Application hosted on Streamlit Cloud for easy access, testing, and sharing.

---

4. Challenges Faced

Environment Management:

- There were repeated issues with loading environment variables correctly between local and cloud environments.
- Streamlit Cloud uses a secrets.toml mechanism, while local development uses .env with python-dotenv.
- Final solution: A helper function was implemented that prioritizes fetching values from st.secrets but falls back to os.getenv() if not found.

API Key Handling:

- Errors such as missing Pinecone API keys were frequent during early deployments.
- Careful verification of environment setup was necessary before each deployment.

Pinecone Connection Problems:

- Pinecone client sometimes failed when region names or environment variables were misconfigured.
- Issue was solved by verifying region settings (us-east-1) and manually setting up the Pinecone client correctly.

Streamlit Reruns:

- Streamlit reruns the script on every button click or user interaction.
- Session state management (st.session_state) was critical to ensure the chat history was maintained across reruns without looping infinitely.

Deployment and Secret Management:

- Early mistakes included missing the correct formatting in Streamlit's secrets.toml, leading to app crashes.
- Proper structure and validation of keys were established to prevent failures.

---

5. Future Work

Enhanced User Authentication:

- Implement OAuth login or simple username/password authentication to maintain personalized chat histories.

Dynamic Knowledge Base Upload:

- Allow users to upload and query their own domain-specific knowledge bases dynamically.

Advanced Retrieval Mechanisms:

- Integrate reranking models such as Cohere or OpenAI's re-ranking tools for smarter retrieval.
- Implement hybrid retrieval (dense + sparse retrieval) strategies.

Fine-Tuning a Domain-Specific Model:

- Fine-tune a lightweight model like Llama-2 or OpenAI custom GPTs to make the system faster and even more context-aware for Generative AI questions.

Multimodal Expansion:

- Enable users to ask questions based on images, PDFs, or audio clips in addition to text.

Error Correction & Clarifications:

- Build an error-handling system that suggests possible rephrased versions of misunderstood questions automatically.

Analytics and Feedback Collection:

- Implement feedback collection at the end of each chat session to improve the model's effectiveness continuously.

---

6. Ethical Considerations

Bias Mitigation:

- Since large language models can inherit biases from training data, the assistant has been designed to answer factually and avoid generating biased, harmful, or politically charged content.
- Answers are constrained to the Generative AI domain to minimize ethical risks.

Data Privacy:

- No personal user information is collected, stored, or tracked.
- API keys are kept confidential using environment variables or cloud secrets management.

Transparency:

- The assistant clearly states when it lacks sufficient knowledge rather than hallucinating answers.

Fair Use:

- Only publicly available knowledge and ethical AI practices were utilized during the development.
- No copyrighted or private data was used in training or retrieval.

Sustainability:

- The system is lightweight and efficient to reduce unnecessary compute usage, promoting environmentally friendly deployment.