Savant

Hamid Nazari
Yosafe Feseha Oqbamecail
Victor Long Tran

Date: 10.11.2023

Generated by Savant

Introduction

This report presents a web application that integrates the OpenAI API to do text generation, image generation, and similarity/semantic search on documents. Our objective is to create a web platform which incorporates several distinct generative AI models. This platform gives potential users the ability to do many things in one place; generate content (text/images) and do simple document analysis. The motivation behind selecting this project was to explore the capabilities of OpenAI models and create a user-friendly interface for utilizing these models in real-world applications. Last but least, it's worth acknowledging that the inspiration for this project originated from three YouTubers who have created somewhat similar products and shared their work on YouTube.

Background

Generative AI models are built upon machine learning algorithms, particularly neural networks. These models are trained on large datasets, typically using unsupervised learning methods. The key idea behind generative models is to learn the underlying patterns and structures within the training data.Once trained, these models can generate new data that resembles the training data but is unique and original. Existing applications of generative AI models include chatbots, language translation, content generation, and more.

Pinecone is a fully managed vector database platform designed to handle high-dimensional vector data at scale. It provides high-performance semantic and similarity search capabilities. Semantic search is a technique that focuses on understanding the meaning of words, phrases, and concepts instead of merely matching keywords. Vector embeddings are numerical representations of text that allows us to do comparison and mathematical analysis, which is something that is required for semantic search.

Our project aligns with established practices in that it utilizes generative AI models for text generation, image synthesis, and semantic search. However, what sets our project apart is the integration of these capabilities within a unified web application. By combining text generation, image synthesis, and semantic search in one platform, we offer users a versatile and comprehensive experience. Additionally, our project diverges

from traditional applications by incorporating Pinecone's vector database for efficient similarity search.

Methodology

For text generation, we chose to integrate the OpenAI API and utilize the GPT-3.5 model. The main reason we chose this model was because it is the cheapest one, and also the best one relative to its price.

To enable image generation, we explored different alternatives, such as DeepAI Image Generation API and Bannerbear Image Generation API. However, in the end, we ultimately decided to go with DALL-E 2, which is an image generation API from OpenAI. DALL-E 2 allows us to create realistic images based on text descriptions. By leveraging this model, our application provides users with the ability to generate images by simply providing text prompts.

For similarity/semantic search on documents, we used the OpenAI Embedding API to generate embeddings for the text data. These embeddings were then indexed and stored in the Pinecone vector database, which offers fast and scalable vector search capabilities. By integrating Pinecone, we enable users to perform efficient similarity/semantic search on their documents.

All these models/APIs are integrated into our web application, which is built with React, TypeScript, and Tailwind CSS for the frontend. The backend of our web application is developed using Python, Flask, MySQL database, and SQLAlchemy, which is an object-relational mapping framework. Additionally, we have used many other small, convenient libraries both in the frontend and backend that are not mentioned here. These small libraries are used for various purposes.

Application Design and Development

The application follows a client-server architecture, where the frontend interacts with the backend via API calls. The user interface design was inspired by the work of three YouTubers, as briefly mentioned in the introduction, incorporating their ideas into the application's visual aesthetics and layout. We made the decision to use these

YouTubers' designs because we wanted to speed up the development process and ensure a visually appealing and intuitive user experience. By leveraging pre-designed elements, we focused more on integrating the generative AI functionality effectively

The backend part of the application handles user requests, interacts with the OpenAI API for text and image generation, and interfaces with the Pinecone vector database for document search operations. The choice of Flask was driven by its simplicity and flexibility in building web applications, while Python provided a robust and efficient environment for integrating the OpenAI models.

Experiments and Results

We conducted a series of experiments using various prompts and documents to assess the performance of our application, and everything functioned as intended. Regarding the data used for training, we did not use any data because there was no need to train our own generative AI model. To assess the quality of the generated text and images,, we did not utilize any benchmarks or performance metrics, as there is not a directly relevant metric for our application that we know of.Instead, we focused on ensuring that the generated outputs were coherent and contextually appropriate with the help of some clever prompt engineering.

However, in the case of similarity/semantic search, we employed the cosine similarity metric to evaluate the performance of the vector database. The scores were consistently above 80% in terms of accuracy.

Challenges and Problem-Solving

During the development of our web application we faced several challenges. The first significant challenge that we encountered was the conversion of documents from JSON objects to binary objects and vice versa. This challenge arose in the context of seamless data transfer between the frontend and backend components of the application. Documents needed to be transformed into a format suitable for

transmission and storage while preserving their integrity and structure. We solved this hurdle by using some rarely used built-in Javascript and Python libraries/API.

Another major challenge was the integration of the Pinecone vector database for similarity/semantic search. The process involved generating embeddings for the documents and indexing them in Pinecone. The reason it was challenging was because we needed to split the documents/texts into chunks before creating embeddings of them. But deciding the appropriate chunk size is not easy. After some consideration, we decide to split the documents/texts into pages.

Discussion

The application's performance surpassed not only our expectations but also those of everyone around us to whom we demonstrated it. Regarding ethical considerations, our application is not advanced enough for us to be concerned about them.

What we have developed is visually appealing and provides an enjoyable user experience. However, in the end, it possesses little to no commercial value unless the OpenAI API becomes completely free to use.

Reproducibility

To ensure the reproducibility of our work, we have used a GitHub repository that shows the development evolution of our application, from start to the end. The GitHub repository is organized with clear folder structures and detailed instructions on setup and usage. Additionally, we have created a comprehensive README file to guide users through the setup process and provide them with the necessary information to replicate our experiments.

Conclusion

In conclusion, the web application successfully integrates the OpenAI API to provide text and image generation capabilities as well as similarity/semantic search on documents. The chosen generative AI models, GPT-3.5 and DALL-E, demonstrate impressive performance in generating high-quality content. The application's architecture, user interface, and user experience considerations contribute to an intuitive and seamless user interaction.

Future work

Moving forward, there are several avenues for extending and improving the application. One of the initial steps to enhance the application is implementing an authentication mechanism. This measure will ensure that only authorized users can access the application's features and data. Additionally, a rate limiting system can be integrated, which is essential for preventing abuse of the OpenAI API and managing costs effectively.

Other functionalities that can be added include integrating a payment gateway, such as Stripe, PayPal, or Square, to enable subscription-based services, and creating a user dashboard where authenticated users can manage their account settings, view their subscription status, and monitor their API usage.

References

YouTubers:

Youtuber: Josh
- Channel: Josh tried coding
- Specific Video: Build a Complete SaaS Platform with Next.js 13, React, Prisma, tRPC, Tailwind | Full Course 2023

Youtuber: Elliott Chong

- Channel: Elliott Chong
- Specific Video: Build and Deploy a Full Stack AI SaaS | Next JS 13, DrizzleORM, OpenAI, Stripe, TypeScript, Tailwind

Youtuber: Antonio

- Channel: Code With Antonio
- Specific Video: Build a SaaS AI Platform with Next.js 13, React, Tailwind, Prisma, Stripe | Full Tutorial 2023

Other Documentations:

- **Pinecone Documentation**
- **OpenAI API Documentation**