Bridged AI Engineer Project - Documentation

Introduction

This project presents an intelligent query agent that translates natural language into Pinecone-compatible metadata filters and performs semantic vector search on a dataset of articles. It integrates OpenAI's language and embedding models with Pinecone's vector database and is deployed as a FastAPI application, making it ready for real-world usage and containerization.

Objective

To build a prototype AI agent that:
- Accepts a natural language query from the user
- Extracts metadata (author, tags, publication date) using GPT
- Generates a Pinecone-compatible filter JSON
- Performs vector similarity search combined with structured filtering
- Returns relevant articles with titles, tags, authors, and scores

Architecture

Components:
1. FastAPI Backend: Exposes a `/search` endpoint.
2. OpenAI Embeddings: Converts article titles into 1536-dimension vectors using `text-embedding-ada-002`.
3. Pinecone: Stores vector data and enables metadata-filtered similarity search.
4. OpenAI Chat Model: Extracts metadata fields from the user's natural language query.
5. Docker: Containerizes the application for easy deployment.

Approach

Phase 1: Data Preparation & Indexing
- Loaded the dataset (CSV format)
- Parsed columns: title, author, tags, published_date
- Cleaned and normalized metadata fields
- Generated embeddings using OpenAI and upserted to Pinecone with metadata

Phase 2: Filter Generation Agent
- Built a prompt-based agent using GPT-3.5 to extract metadata like:
  - author
  - tags
  - published_year and published_month
- Returned filters in Pinecone-compatible JSON format

Phase 3: Semantic Search
- Fetched vector embedding for query text
- Used index.query() with:
  - Query vector
  - Metadata filter from GPT
- Returned top k results with metadata and similarity scores

Challenges Faced with Solutions

1. Pinecone Dimension Mismatch
Issue: Default index dimension (1024) incompatible with OpenAI embedding (1536)
Solution: Deleted index and recreated with correct dimension using code

2. Incorrect Tag Formatting
Issue: Tags loaded as strings instead of list
Solution: Used ast.literal_eval to parse tags column into valid Python lists

3. GPT Output Parsing
Issue: GPT occasionally returned JSON-like text that failed parsing
Solution: Used ast.literal_eval with a try-except block and fallback handler

4. Result Titles Not Displaying
Issue: title was not included in metadata
Solution: Updated upsert script to include title in metadata

Conclusion

This project demonstrates an end-to-end intelligent agent system that combines the power of large language models and vector databases. It effectively translates natural user queries into structured and semantic search logic, ready to be scaled via a Dockerized FastAPI service. With clear separation of concerns, well-defined API endpoints, and clean architecture, this solution is highly extensible for production use.