Generative AI Project Documentation

Project: HelpMateAI
Author: Ankur Parashar
Version: 1.0
Date: 3rd September 2025

Table of Contents

1. Project Summary

HelpMateAI is a Generative AI project aimed at building an intelligent assistant that leverages advanced natural language processing techniques to provide accurate, context-aware responses. The system integrates semantic search, cross-encoder reranking, and OpenAI models to deliver high-quality outputs.

2. Project Goals

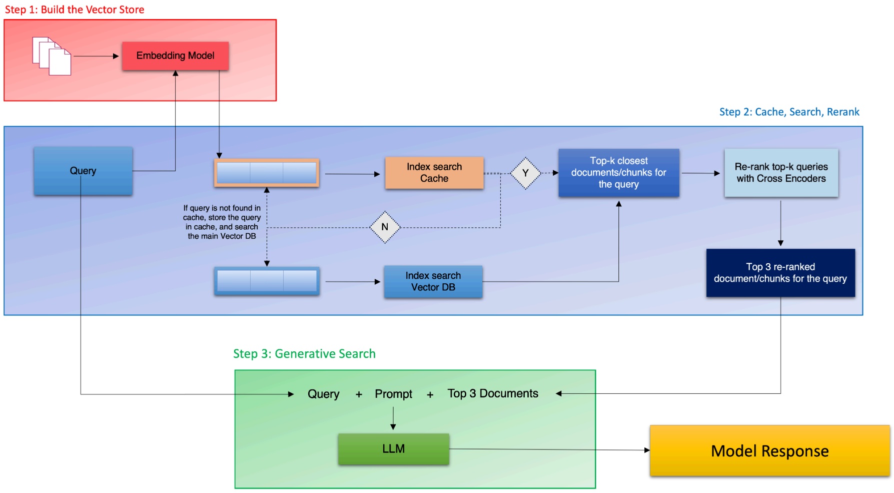
- Develop a Generative AI assistant capable of answering user queries with precision.
- Implement semantic search and reranking mechanisms.
- Ensure scalability, performance, and data security.
- Provide a foundation for enterprise adoption.

3. Data Sources

The project uses structured and unstructured data sources, including documents, knowledge bases, and embeddings stored in a vector database. Metadata is used for filtering and improving contextual accuracy.

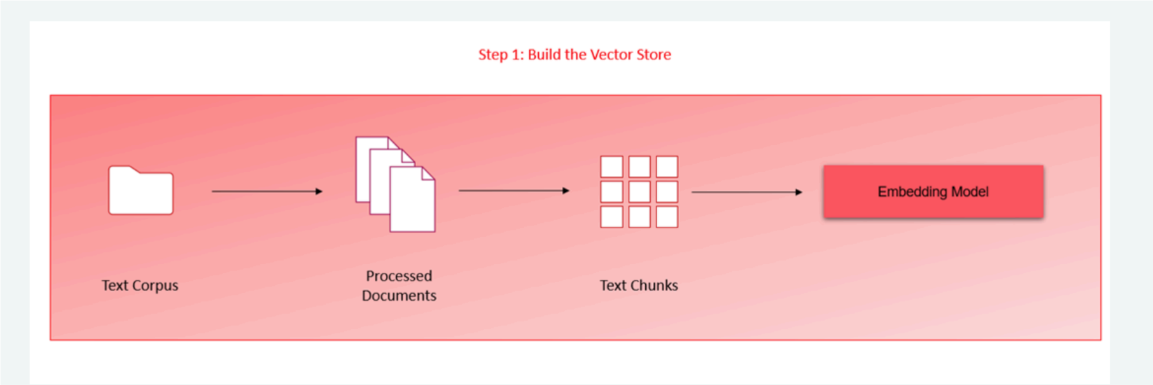
4. System Architecture

The architecture consists of the following components:
- Frontend: User interface for queries.
- Backend: Python-based API handling model calls.
- Embedding Model: Generates vector embeddings.
- Vector Database: Stores embeddings with metadata.
- Cross-Encoder: Reranks semantic search results.
- Response Generator: Uses LLM to provide final answers.


Step 1: Build the Vector Store
Embedding Model
Step 2: Cache, Search, Rerank
Query
Index search Cache
Y
Top-k closest documents/chunks for the query
Re-rank top-k queries with Cross Encoders
If query is not found in cache, store the query in cache, and search the main Vector DB
N
Index search Vector DB
Top 3 re-ranked document/chunks for the query
Step 3: Generative Search
Query + Prompt + Top 3 Documents
LLM
Model Response

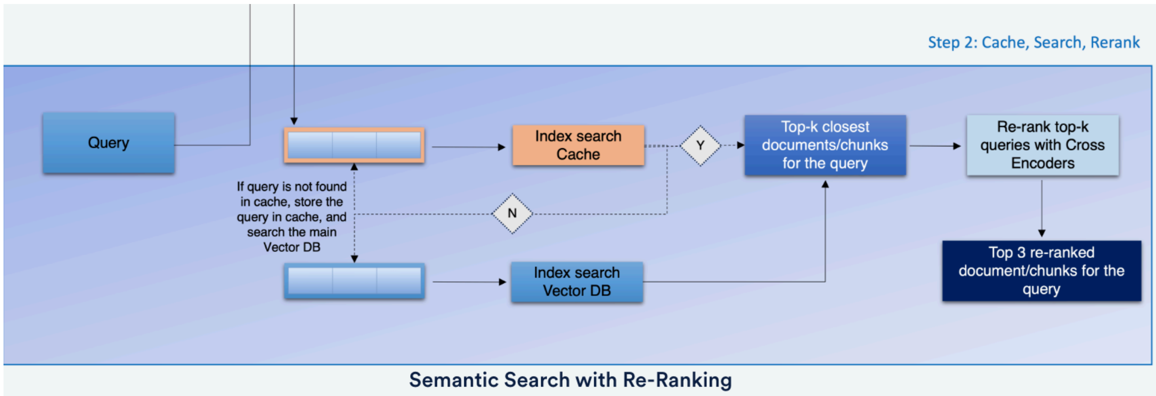
Step1-

Reading & Processing ⟶ Chunking ⟶ Generate Embedding



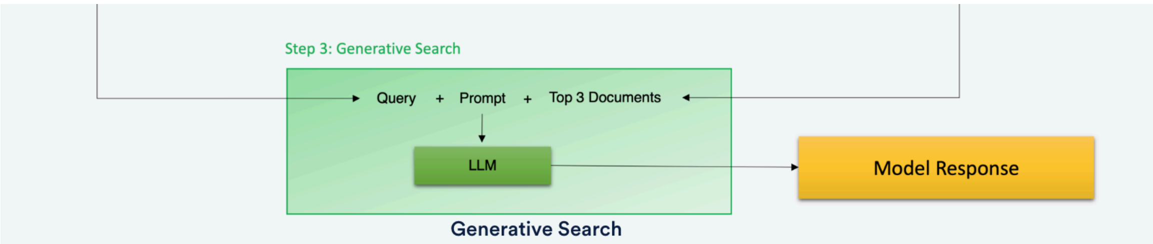
Step2-

Cache ⟶ Search ⟶ ReRank


Step 2: Cache, Search, Rerank
Query
Index search Cache
Y
Top-k closest documents/chunks for the query
Re-rank top-k queries with Cross Encoders
If query is not found in cache, store the query in cache, and search the main Vector DB
N
Index search Vector DB
Top 3 re-ranked document/chunks for the query
Semantic Search with Re-Ranking

Step3-

Generative Search



5. Design Choices

- Embedding model selected: OpenAI Embeddings.
- Reranking: Cross-Encoder for semantic precision.
- Chunking strategy applied to documents for efficient retrieval.
- Metadata-based filtering for context-driven results.

6. Implementation Details

Example: Semantic Search & Reranking Function

```
cross_inputs = [[query, response] for response in results_df['Documents']]
cross_rerank_scores = cross_encoder.predict(cross_inputs)
results_df['Reranked_scores'] = cross_rerank_scores

# Return the top 3 results from semantic search
top_3_semantic = results_df.sort_values(by='Distances')
top_3_semantic[:3]

# Return the top 3 results after reranking
top_3_reranked = results_df.sort_values(by='Reranked_scores', ascending=False)
top_3_reranked[:3]
```

7. Challenges Faced

- Handling hallucinations in LLM responses.
- Optimizing query latency for real-time usage.
- Ensuring data privacy and API key security.
- Scaling vector databases for large datasets.

8. Testing & Validation

Testing strategy includes unit tests for data preprocessing, API endpoints, and ranking functions. Evaluation metrics involve semantic accuracy, recall, and precision in retrieval tasks.

10. Security Considerations

- API key management using environment variables.
- Encrypted database exports and imports.
- Access restrictions for sensitive data.
- Compliance with enterprise security policies.

11. Future Enhancements

- Fine-tuning custom LLMs for domain-specific accuracy.
- Multimodal integration (text + images).
- Distributed architecture for high availability.
- Advanced monitoring with logging and tracing.

12. References & Appendices

- OpenAI API Documentation
- Transformers
- Vector DB (Pinecone)