# HelpMateAI: RAG based search system with Cache and reranking Project Report

## Introduction

HelpMateAI is an intelligent assistant designed to provide support and automate tasks for users in various domains such as customer service, technical support, and personal assistance. The project aims to leverage advanced natural language processing (NLP) and machine learning (ML) techniques to deliver a seamless and efficient user experience.

## Objectives

* Develop a robust semantic search system to understand Insurance policy data and process user queries.
* Extract relevant information from PDF documents, store them in a structured format, and generate vector representations using SentenceTransformerEmbeddings or Open AI Embeddings.
* Implement a retrieval-augmented generation (RAG) system to provide accurate and contextually relevant responses.
* Integrate the AI assistant with various communication platforms (e.g., web, mobile, chatbots).
* Ensure data privacy and security in all interactions.

## System Architecture

The system architecture of HelpMateAI consists of the following components:

A screenshot of a computer screen

Description automatically generated

### 3.1. RAG Pipeline:

**Embedding Layer:** Turn text and tables from PDFs into dataframes and create vector representations with OpenAI's text-embedding-ada-002 model. Save these vectors in ChromaDB.

**Search and Rank Layer:** Conduct a semantic search for user queries within the knowledge database, bringing up the most relevant results.

**Generation Layer:** Combine the search results with the original query and a structured prompt to form coherent responses using a language model.

### 3.2. Cache Implementation:

Maintain a minimum 0.2 semantic similarity for cache.

Archive pairs of queries and their outcomes in ChromaDB's cache\_collection for better retrieval.

Utilize ChromaDB's tools for injecting documents, identifiers, and details into the cache\_collection.

## Technologies Used

* **Programming Languages**: Python
* **Frameworks**: Open AI, HuggingFace
* **Databases**: Croma Vector DB

## Implementation:

Use Visual Studio Code for development and leverage libraries such as pdfplumber, tiktoken, Open AI, Chroma DB, and sentence-transformers for document processing, embedding, and caching.

Implement functions to extract text and tables from PDFs, create a data frame, generate vector embeddings, and perform semantic searches using the RAG pipeline.

Develop a cache system using Chroma DB to store and retrieve previous queries and their results.

The three layers for RAG pipeline are:

* Embedding Layer
* Search Layer
* Generation Layer

### Embedding Layer:

Processing and Chunking: Explore and compare various strategies for effective PDF document processing, cleaning, and chunking. Evaluate the impact of different chunking strategies on the quality of the retrieved results.

Embedding Model Choice: Choose between OpenAI's embedding model and SentenceTransformers from HuggingFace. Assess the impact of the selected model on the quality of vector representations**.**

**A screenshot of a computer

Description automatically generated**

### Search Layer:

Query Design: Design a minimum of three queries that reflect information seekers' potential questions in the policy document. Ensure queries cover diverse aspects of the document to thoroughly test the system.

Vector Database Search: Embed queries and perform searches against the ChromaDB vector database. Implement a cache mechanism to store and retrieve previous queries and their results.

Re-ranking Block: Integrate a re-ranking block utilizing cross-encoding models from HuggingFace to enhance the relevance and accuracy of search results.

**A screenshot of a computer

Description automatically generated**

When using cross\_encoder: **A screenshot of a computer

Description automatically generated**

### Generation Layer:

Prompt Design: Focus on designing a comprehensive and instructive prompt for the Language Model (LM) in the generation layer. Ensure the prompt effectively conveys relevant information to the LM for coherent answer generation.

Few-shot Examples: Enhance LM performance by providing few-shot examples in the prompt to guide the model in generating more contextually accurate responses.

### Overall Project Insights:

Performance Evaluation: Conduct thorough evaluations for each layer, considering the impact of different strategies, models, and components on the system's performance.

Scalability: Address concerns about system scalability by considering potential increases in document numbers or user queries. Implement measures such as vector database scaling and compute unit adjustments.

Documentation and Codebase: Ensure a well-documented codebase that includes detailed explanations of implemented strategies, models, and mechanisms. Provide clear instructions for potential future developers or collaborators.

## Challenges:

Performance Scaling: Address concerns about system performance with an increased number of documents or users by implementing vector databases and scaling up compute units.

Cache Storage: Optimize the cache collection to efficiently store and retrieve queries and results.

## Lessons Learned:

Efficient Document Processing: Processing PDFs efficiently is crucial; libraries like pdfplumber and suitable data structures for storage play a vital role.

Semantic Search Optimization: Fine-tune semantic search parameters and thresholds for optimal results.

Cache Management: Implement an effective cache management strategy to balance storage and retrieval efficiency.

## Conclusion:

HelpMateAI successfully achieved its objectives by delivering an intelligent assistant capable of understanding and responding to user queries efficiently. The project demonstrated the effective use of NLP, Generative AI and ML technologies to enhance user experience and automate tasks. Future enhancements will further improve the system's capabilities and expand its reach.