Healthcare RAG Chatbot - Comprehensive Project Documentation

Table of Contents

1. Project Overview

2. What It Is

3. What It Does

4. Project Goals

5. System Architecture

6. Data Flow Diagram

7. User Flow Diagram

8. Technical Stack

9. AI Terminology Used

10. Working Steps

11. Key Components

12. File Structure

13. Installation Guide

14. Usage Instructions

15. Customization Options

16. Future Improvements

17. Contributing

18. License

Project Overview

The Healthcare RAG Chatbot is an intelligent question-answering system that provides accurate healthcare

This project demonstrates how modern AI techniques can be applied to the healthcare domain to create a

What It Is

The Healthcare RAG Chatbot is a web-based application that combines:

- Retrieval-Augmented Generation (RAG) technology for accurate information retrieval
- Natural Language Processing (NLP) for understanding user queries
- Vector search using FAISS for efficient document retrieval
- Conversational AI for generating human-like responses
- Modern web interface with responsive design and dark/light mode

What It Does

The system performs the following functions:

1. Medical Information Retrieval: Processes user healthcare questions and retrieves relevant information fr
2. Accurate Response Generation: Generates context-aware answers based on actual medical documenta
3. Secure Interaction: Ensures all processing happens locally with no data storage
4. Responsive Interface: Provides a modern, user-friendly chat interface
5. GPU Acceleration: Utilizes available GPU resources for faster processing

Project Goals

- Provide accurate healthcare information without hallucination
- Create a secure system that protects user privacy
- Build a scalable architecture that can handle various medical document types
- Develop an intuitive user interface for seamless interaction
- Demonstrate practical application of RAG technology in healthcare

System Architecture

Data Flow Diagram

User Flow Diagram

Technical Stack

Backend Technologies

- Python 3.8+ - Primary programming language
- Flask 3.1.2 - Web framework for the backend API

- PyTorch 2.9.1 - Machine learning framework for GPU acceleration

- Transformers 4.57.1 - Hugging Face library for pre-trained language models

- Sentence Transformers 5.1.2 - Text embedding generation

- FAISS 1.13.0 - Vector database for efficient similarity search

- NumPy 2.2.6 - Numerical computing library

- PyPDF 6.1.2 - PDF document parsing

Frontend Technologies

- HTML5 - Markup language for web pages

- CSS3 - Styling and layout

- JavaScript - Client-side scripting

- React - Component-based UI library

- Tailwind CSS - Utility-first CSS framework

- Babel - JavaScript compiler

Development & Deployment

- Virtual Environment - Isolated Python environment

- pip - Package installer for Python

- Git - Version control system

AI Terminology Used

Retrieval-Augmented Generation (RAG)

A technique that combines information retrieval with language generation to produce more accurate and co

Vector Embeddings

Numerical representations of text in high-dimensional space where semantically similar texts have similar v

Semantic Search

A search technique that understands the meaning and context of queries rather than just matching keywor

Similarity Search

The process of finding items in a dataset that are most similar to a given query item, typically using distanc

Language Model

A type of artificial intelligence trained to understand and generate human-like text based on patterns learne

Text Chunking

The process of breaking large documents into smaller, manageable pieces while preserving context and m

Vector Database

A specialized database designed for storing and querying high-dimensional vector embeddings, optimized

GPU Acceleration

Utilizing Graphics Processing Units for parallel computation to significantly speed up machine learning and

Working Steps

1. System Initialization

1. Application starts and initializes Flask web server

2. Checks for GPU availability and configures PyTorch accordingly

3. Loads required AI models (embedding model and language model)

4. Looks for healthcare PDF documents to process

2. Document Processing (Offline Phase)

1. Loads medical documents (PDF format) using PyPDF

2. Extracts text content from PDF pages

3. Splits text into chunks of 500 characters with 50-character overlap

4. Converts each chunk to vector embeddings using Sentence Transformers

5. Stores embeddings in FAISS vector database for efficient retrieval

3. User Query Processing (Online Phase)

1. User submits a healthcare question through the web interface

2. Frontend sends the question to backend via HTTP POST request

3. Flask API receives and validates the request

4. Question is converted to vector embeddings using the same embedding model

5. FAISS performs similarity search to find the most relevant document chunks

6. Top 4 most relevant documents are retrieved as context

4. Response Generation

1. Retrieved documents are combined to form context for the language model

2. A prompt is constructed containing the context and original question

3. DialoGPT language model generates a response based on the prompt

4. Response is formatted and sent back to the user through the web interface

5. Display to User

1. Frontend receives the response via HTTP

2. Displays the answer in the chat interface with typing animation

3. User can ask follow-up questions or close the session

Key Components

1. PDF Document Loader

Responsible for parsing and extracting text content from medical PDF documents. Uses PyPDF library for

2. Text Chunking Module

Splits large documents into smaller chunks to enable granular retrieval. Uses 500-character chunks with 5

3. Embedding Model

Converts text (both documents and queries) into numerical vector representations. Uses all-MiniLM-L6-v2

4. FAISS Vector Store

Stores document embeddings and performs fast similarity searches. Uses IndexFlatL2 for exact nearest n

5. Language Model

Generates natural language responses based on retrieved context. Uses DialoGPT-small for conversation

6. Flask Web Server

Handles HTTP requests, routes API endpoints, and serves web pages. Provides REST API for chat functi

7. React Frontend

Provides modern, responsive user interface with real-time chat experience. Includes features like dark/ligh

File Structure

Installation Guide

Prerequisites

- Windows 10 or later

- Python 3.8+

- pip (Python package manager)

Step-by-Step Installation

Step 1: Clone the Repository

Step 2: Create Virtual Environment

Step 3: Activate Virtual Environment

Step 4: Install Dependencies

Step 5: First Run Setup

> ? Note: The application will automatically download required AI models on first run. This may take 2-5 mi

Usage Instructions

Starting the Application

Accessing the Interface

Open your browser and navigate to:

Using the Chatbot

1. Homepage: Click "Start Consultation" button

2. Ask Questions: Type your health-related query in the input field

3. Submit: Press Enter or click the send button

4. Get Answers: Receive AI-generated responses instantly with typing animation

5. Continue: Ask follow-up questions or switch to dark/light mode using the theme toggle

Example Questions

- "What are the symptoms of diabetes?"

- "How to perform CPR on an adult?"

- "What's the difference between flu and common cold?"

- "How to maintain heart health?"

Customization Options

Adding More Medical Documents

1. Replace the existing health.pdf with your own medical documents

2. The system will automatically process the new documents on next startup

3. Supports PDF format with text content

Changing AI Models

1. Modify the model names in app.py:

   - Update model_name for the language model

   - Update the embedding model in SentenceTransformer() initialization

2. Restart the application to load new models

Adjusting Retrieval Parameters

1. Modify chunk_size and overlap in the split_text() function

2. Change the number of retrieved documents by adjusting the k parameter in retrieve_documents()

Customizing the Interface

1. Modify index.html for landing page changes

2. Modify bot.html for chat interface customization

3. Update CSS styles in the <style> sections of HTML files

Future Improvements

Model Enhancements

- Fine-tune models on medical domain data for better accuracy

- Implement more advanced RAG techniques like HyDE or query expansion

- Add multi-document reasoning capabilities

System Scalability

- Support multiple concurrent users with proper request queuing

- Implement document versioning and management system

- Add user authentication and personalized experiences

Feature Extensions

- Multi-language support for global accessibility

- Voice input/output capabilities for hands-free interaction

- Integration with electronic health records (EHR) systems

- Personalized health recommendations based on user history

Performance Improvements

- Implement caching for frequent queries to reduce processing time
- Add asynchronous processing for better responsiveness
- Apply model quantization techniques for better GPU utilization
- Optimize FAISS indexing for larger document collections

Contributing

We welcome contributions to improve the Healthcare RAG Chatbot!

Steps to Contribute:

1. ? Fork the repository
2. ? Create your feature branch (git checkout -b feature/AmazingFeature)
3. ? Commit your changes (git commit -m 'Add some AmazingFeature')
4. ? Push to the branch (git push origin feature/AmazingFeature)
5. ? Open a Pull Request

Areas for Contribution:

- Improving medical document processing
- Enhancing the user interface
- Optimizing AI model performance
- Adding new features and capabilities
- Expanding documentation and examples

License

This project is licensed under the MIT License - see the LICENSE file for details.

The MIT License is a permissive open-source license that allows for commercial use, modification, distribu

This documentation provides a comprehensive overview of the Healthcare RAG Chatbot project, covering i