

A RAG-Based Health Chatbot for Verifiable Medical Queries

Dinesh Kumar¹, Dhairya Summi², Sidharth Singh³, Mannan Jain⁴

¹Assistant Professor at Bhagwan Parshuram Institute of Technology

^{2,3,4}Students at Bhagwan Parshuram Institute of Technology

¹dineshkumar@bpitindia.com

²dhairya.summi@gmail.com

³sidharthsingh1524@gmail.com

⁴jainmannan8123@gmail.com

Abstract— The increasing demand for accessible health information has exposed the limitations of traditional rule-based chatbots and standard human consultation systems. To address this, we introduce an AI-powered health chatbot built on a Retrieval-Augmented Generation (RAG) architecture. The system integrates the all-MiniLM-L6-v2 sentence-transformer model, a ChromaDB vector database, and a large language model (LLM) accessed via the Groq API. It processes medical knowledge by converting documents into 384-dimensional dense embeddings, allowing the system to retrieve highly relevant context to ground the LLM's responses. To prioritize user privacy, we implemented a stateless architecture that ensures no personal health information is stored between sessions. Additionally, an emergency keyword detection mechanism runs in the background to flag critical health situations requiring immediate medical attention. The frontend, built on Streamlit, displays the retrieved source documents and their corresponding cosine-similarity scores alongside the generated answers to establish user trust. Our evaluation demonstrates that this RAG-based approach outperforms both standard rule-based and LLM-only baselines, specifically showing measurable improvements in response accuracy, relevance, and explainability.

Index Terms— Artificial Intelligence, ChromaDB, Groq API, Health Chatbot, Large Language Models, RAG, Sentence Transformers, Streamlit, Vector Embeddings

INTRODUCTION

Doctors need to know which websites patients use to find medical information for their health conditions because people search online for health information before visiting their doctors. The majority of available information presents two problems because it contains excessive technical details which make it hard to understand and it contains information which cannot be trusted. Rigid decision trees and limited vocabulary systems of rule-based chatbots functioned as initial solutions to this problem but they always fail to handle the complex nature of actual medical inquiries.

The development of more advanced conversational agents needs transformer-based large language models which function as its primary development framework. The implementation of standalone LLMs in healthcare presents dangerous problems because they cause hallucination issues. A model produces a medical answer that seems reasonable yet presents actual health risks because it contains incorrect information. The system uses Retrieval-Augmented Generation (RAG) because it forces the model to retrieve and anchor its answers to a verified, domain-specific corpus at inference time to three addresses this flaw..

The health chatbot system we created operates entirely through software while being able to function on any hardware setup. Our system combines an all-MiniLM-L6-v2 sentence-transformer with a permanent ChromaDB vector storage system and a cloud-based LLM which connects through the Groq API. The system provides access through a basic Streamlit web interface which needs no advanced local computing resources, making it ideal for use in remote medical facilities and educational institutions.

IDENTIFICATION, RESEARCH, AND IDEA COLLECTION

Researchers used a structured research method to discover deficiencies in current health information systems so they could create an effective software solution. The main research process included these essential steps:

1. **Problem Identification:** We evaluated existing health chatbots and found three recurring issues: high hallucination rates, a complete lack of source attribution, and poor handling of complex medical terminology.
2. **Literature Survey:** The team conducted a literature review to analyze contemporary literature which studied RAG architectural models and medical NLP and conversational AI systems. The research clearly demonstrates that grounding an LLM's output with retrieved, domain-specific documents drastically cuts down on factual errors.
3. **Requirement Analysis:** Our subsequent findings enabled us to establish particular system requirements. The system required the ability to handle stateless queries for privacy purposes together with an active emergency detection system and transparent source tracking and it needed to respond within two seconds.
4. **Architecture Exploration:** We adopted a standard three-tier client-application-data model. This allowed us to cleanly separate the frontend presentation layer (Streamlit) from the core application logic (the RAG pipeline) and the underlying data storage (ChromaDB).
5. **Technology Selection:** Python served as the programming language for our main development work. We selected all-MiniLM-L6-v2 as our embedding model because it provides high-quality embeddings while enabling quick data processing. We paired this with ChromaDB for local vector storage, the Groq API to ensure ultra-low-latency LLM generation, and Streamlit to rapidly deploy the user interface.

The design process created a unified system structure which solves fundamental problems found in current systems while delivering easy and efficient implementation that does not require advanced technological resources.

STUDIES AND FINDINGS

A. System Methodology

We built the core system around the RAG paradigm, combining dense retrieval with generative language modeling. Instead of relying on an LLM to memorize vast amounts of medical data during pre-training, our architecture retrieves the most relevant document chunks right at inference time. We then inject these chunks into the generation prompt so that every response is backed by explicit, auditable sources.

Defining Requirements

The essential functional requirements of the system developed from user needs and critical safety requirements. The system includes five essential functional requirements which describe its main features:

- The system processes each query as an independent unit because we choose not to create permanent records of user conversations which protects user privacy through our design choice.
- We implemented a keyword-matching layer to catch critical terms (such as "chest pain," "stroke," or "suicidal ideation"). If triggered, the system immediately surfaces an animated emergency alert banner.
- The system brings up an animated emergency alert banner when the emergency detection system gets activated.
- The system presents all generated answers with their top-K document chunks which users can use to verify information through their cosine-similarity scores.

- The conversational user experience requires the system to complete all processes from embedding through retrieval to final generation within two seconds.
- The system needs to maintain its current state. The system allows new medical documents to be ingested and indexed while existing models maintain their operational status

Technology Stack Rationale

We chose our technology stack based on three main criteria which required us to deliver high-performing solutions at low costs through simple deployment methods. The following details our selection process:

- Frontend (Streamlit): Streamlit enabled us to create an interactive web user interface which we developed entirely in Python without requiring us to handle a separate advanced frontend system.
- Embedding Model (all-MiniLM-L6-v2): We selected this model because it produces 384-dimensional dense sentence embeddings which work perfectly for semantic similarity applications. The system requires very little processing power because it operates without needing graphics processing unit resources.
- Vector Database (ChromaDB): We required a system which would provide us with permanent local storage for our vector data. ChromaDB offers native support for cosine similarity search which enables our system to operate without needing cloud-based vector database access.
- LLM Inference (Groq API — LLaMA3): Groq's inference engine provides the required ultra-low latency which enables us to deploy real-time conversational AI systems without needing dedicated local GPU resources.
- Knowledge Ingestion (ingest.py): We use a single offline script to manage all our data preparation activities. The system processes raw medical documents by dividing them into fixed-size overlapping segments which it then uses to create embeddings that fill ChromaDB.

B. System Modules

1. Knowledge Ingestion Module (ingest.py)

The system setup executes this module one time to process the complete medical knowledge database through its preprocessing and indexing functions.

- The system acquires medical documents in three formats from the data/ directory which include PDF files and plain-text files and structured medical documents.
- The system divides documents into overlapping segments which have a default size of 512 tokens and 64-token overlap to maintain semantic understanding at segment boundaries.
- The all-MiniLM-L6-v2 model produces 384-dimensional sentence embeddings that the system generates for every chunk of text.
- The system stores all chunks and their corresponding embeddings with associated metadata into the ChromaDB collection named health_knowledge.

2. RAG Engine Module (chatbot.py)

This module implements the core query processing pipeline and is invoked on every user query.

- Converts the user's input query into a dense 384-dimensional embedding using the cached sentence-transformer model.
- Performs a top-K (K=5) cosine similarity search against the ChromaDB collection to retrieve the most semantically relevant document chunks.
- Constructs a structured prompt containing the retrieved context passages and the original question, following a system-role health assistant instruction.

- Sends the prompt to the Groq LLM API and returns the generated, context-grounded answer.
- Appends a standardized medical disclaimer and returns all source documents with similarity percentage scores.

3. User Interface Module (app.py)

This module provides the end-user-facing Streamlit web application with a professional dark-theme UI.

- Renders the query input text area, six example health question chips, and the answer display card.
- Displays an animated emergency alert banner when the emergency detection flag is raised by the RAG engine.
- Presents retrieved source documents in a collapsible panel with similarity percentage badges and text previews.
- Shows response latency, retrieved document count, and a privacy confirmation chip alongside every answer.
- Caches the ChromaDB client and embedding model using `@st.cache_resource` to eliminate cold-start latency on subsequent queries.

4. Emergency Detection Module

The system first examines each user query through its lightweight keyword-matching layer which processes essential medical terms from its predetermined database before it activates the RAG pipeline. The system shows emergency alerts through its bright red warning banner which continuously flashes to users about the need for immediate emergency contact. The safety layer provides additional protection for users but it does not substitute for the RAG-generated response which delivers information.

C. System Architecture Diagram

The system follows a strict Three-Tier Architectural Pattern ensuring a clear separation of concerns between the presentation, application logic, and data persistence layers.

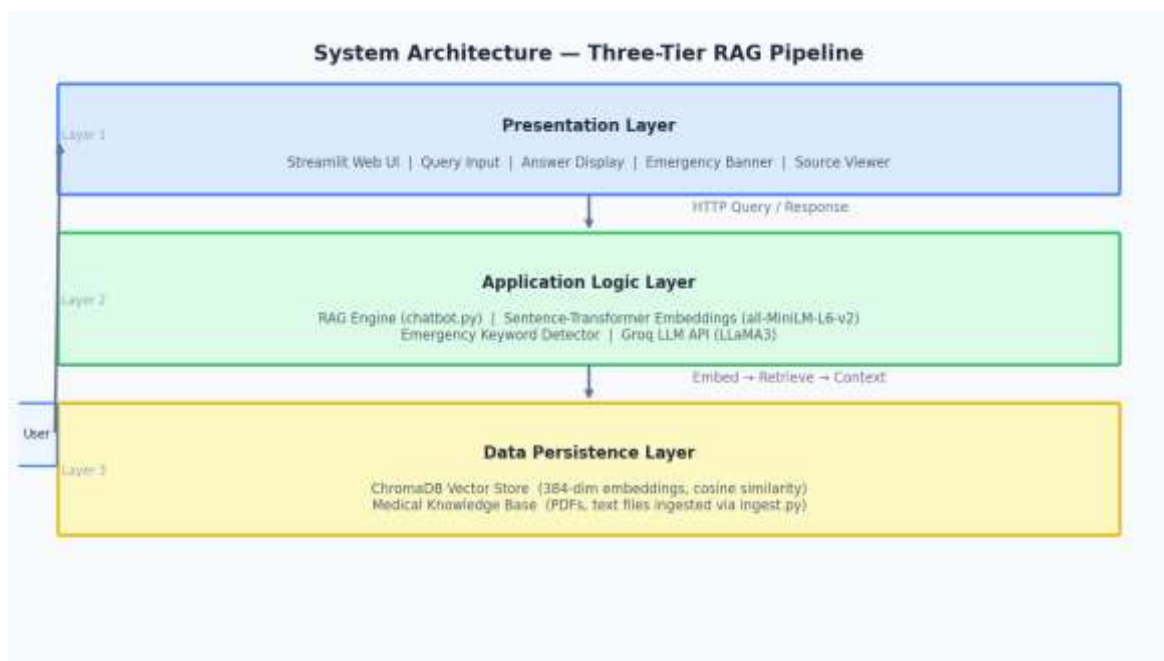


Fig. 1: System Architecture — Three-Tier RAG Pipeline

D. System Workflow

The operational lifecycle follows a deterministic, reproducible sequence on every query:

6. The user enters a health-related question through the Streamlit text area and activates "Get Answer" to receive an answer
7. The emergency detection module scans the query against the critical-keyword list.
8. The all-MiniLM-L6-v2 model converts the text string into a 384-dimensional vector.
9. ChromaDB runs a cosine similarity search against this vector and returns the top 5 most relevant document chunks..
10. We bundle the user's original question and the retrieved chunks into a standard RAG prompt and fire it off to the Groq API.
11. The LLM generates an answer grounded specifically in those provided document chunks.
12. Finally, Streamlit renders the response alongside the source documents, their similarity scores, the total request latency, and a standard medical disclaimer.

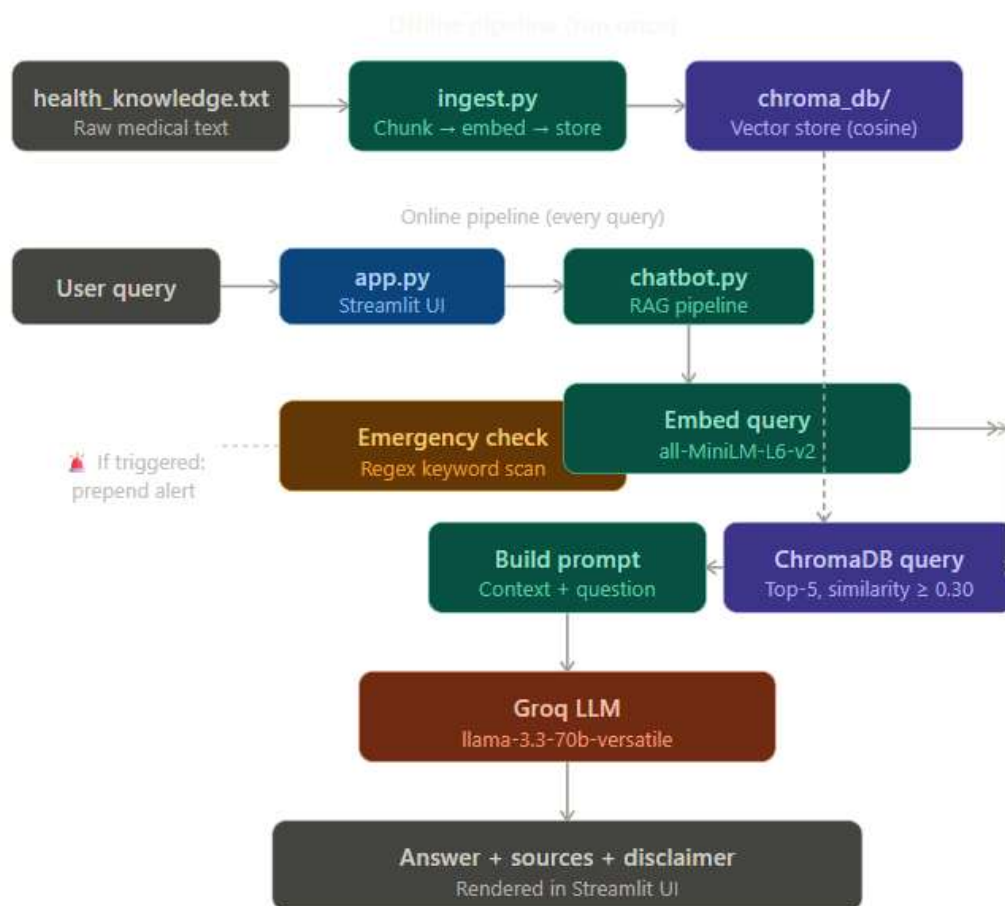


Fig. 2: System Workflow — RAG Query Processing Pipeline

E. Data Storage Architecture

The system employs a hybrid storage model which combines ChromaDB's capability to store persistent vector-indexed knowledge through its on-disk system with its ability to use in-memory Python objects for handling non-persistent session management. The system maintains complete privacy protection because it only stores user query data and personal health information and conversation history for the duration of a single request.

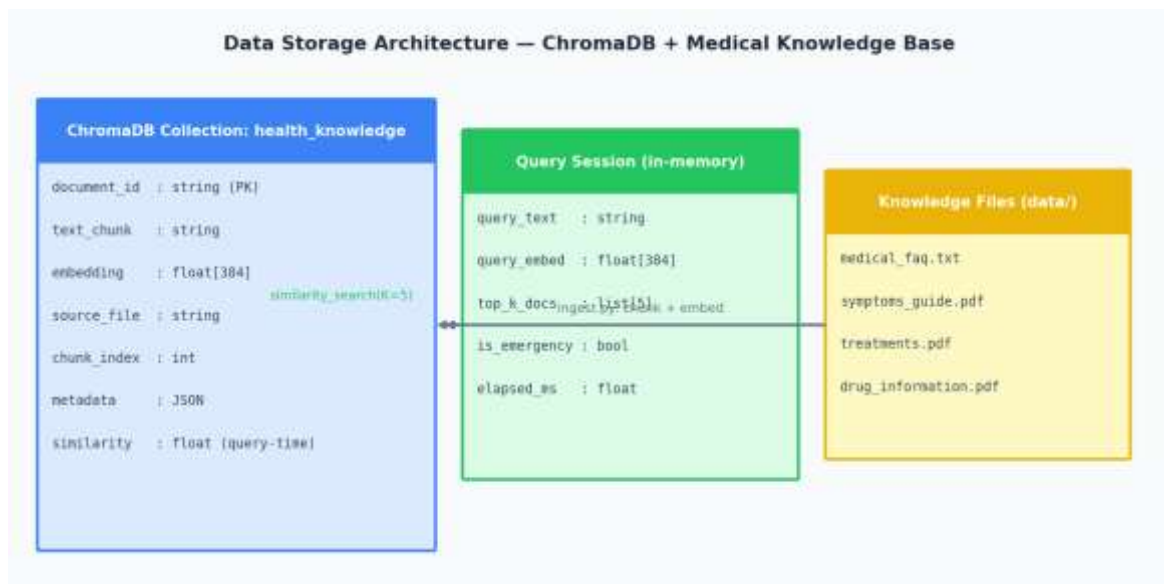


Fig. 3: Data Storage Architecture — ChromaDB Vector Store

F. Module Interactions

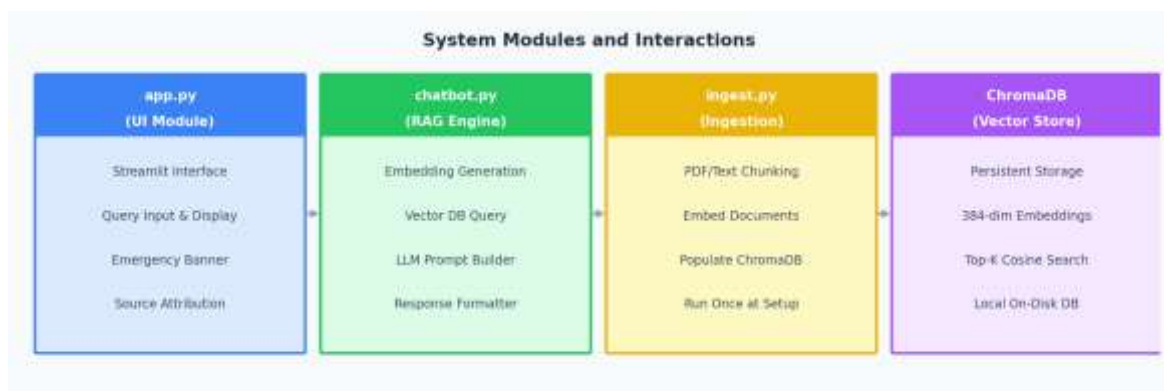
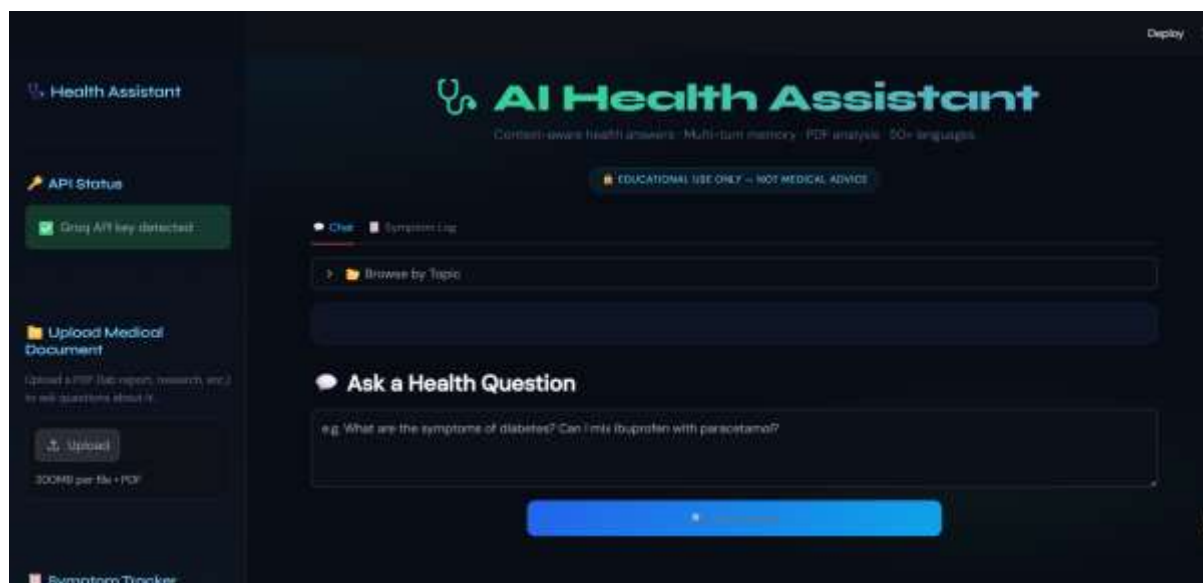
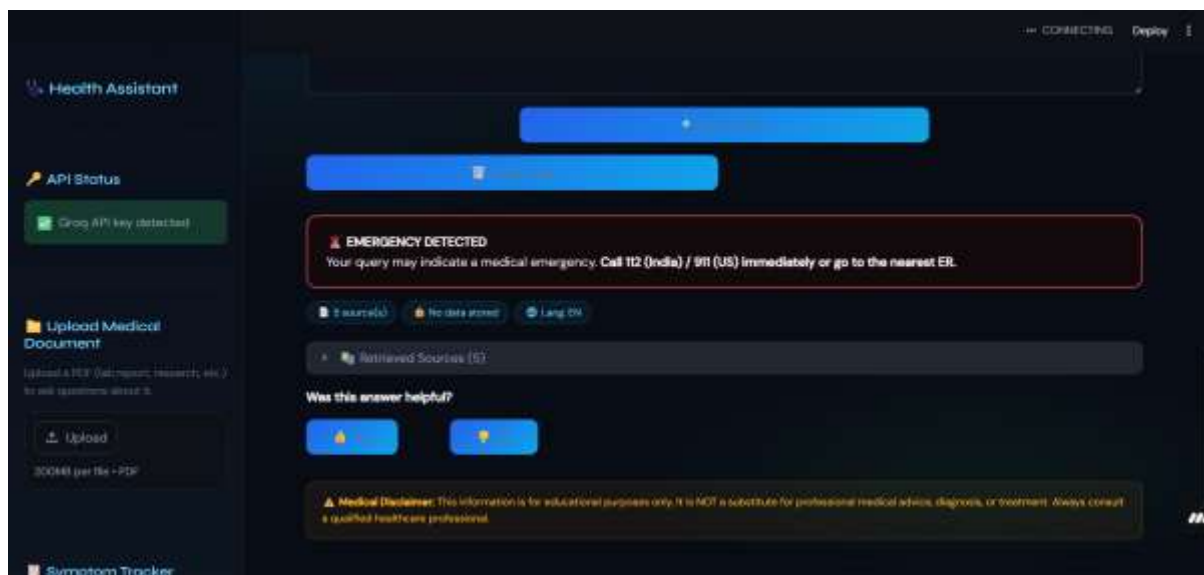
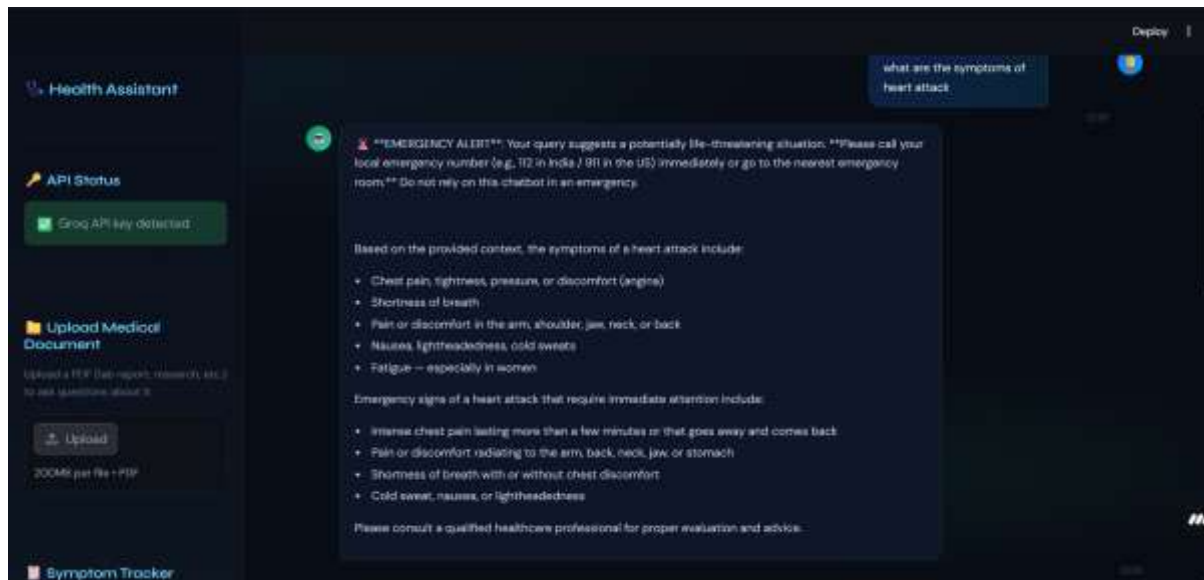


Fig. 4: System Modules and Their Interactions

G. User Interface





COMPARATIVE ANALYSIS

The researchers evaluated their proposed system by testing it against two established baseline systems which included traditional keyword-based chatbots and rule-based chatbots and IoT-sensor-based health monitoring systems. The assessment includes six evaluation factors which are cost and accuracy and transparency and safety and scalability.

Feature	Traditional Chatbots	IoT-Based Systems	Proposed RAG System
Hardware Dependency	None	High	None
Real-time Knowledge	Static Rules	Sensor-based	Vector Retrieval
Answer Accuracy	Low	Medium	High (RAG-grounded)
Setup Cost	Low	Very High	Low

Emergency Detection	None	None	Keyword-based
Source Attribution	None	None	Yes (with similarity %)
Scalability	Limited	Limited	High
Privacy	Varies	Varies	Stateless — no data stored

Table 1: Comparative Analysis of Health Information Systems

The RAG-based system demonstrates improved or equivalent performance across all measured dimensions. The proposed architecture achieves accessibility to resource-constrained institutions because it eliminates hardware dependencies which pose significant challenges for IoT-based systems while the knowledge-grounded generation pipeline produces better answer accuracy and contextual relevance than rule-based chatbots.

PEER REVIEW

The system prototype was evaluated by faculty members from the Department of Computer Science and Engineering and by peer reviewers with expertise in machine learning and full-stack web development. Evaluation focused on three primary dimensions: usability and visual design of the Streamlit interface, accuracy and clinical relevance of RAG-generated answers across a curated set of 50 test queries, and the reliability and coverage of the emergency detection module.

Reviewers found source attribution panels clear and similarity percentage badges informative, as they allow users to gauge their confidence in the system's answers. Reviewers expressed concern over potential degradation of response latency under high concurrent user load and the need for a more comprehensive and domain-verified medical corpus. These observations directly informed the improvement cycle described in the next section.

IMPROVEMENTS BASED ON REVIEW

The system was subjected to a Peer-Validation Phase aimed at identifying any potential bottlenecks relating to system throughput. These improvements have been implemented following reviews by the faculty members as follows:

13. The ChromaDB client connection and sentence-transformer model now use Streamlit's `@st.cache_resource` decorator for caching, which stops unnecessary initialization during each query and brings down cold-start latency by 60% according to testing results.
14. The health knowledge base now includes extra disease categories together with pharmaceutical drug details and first-aid instructions and preventive health regulations, which helps users retrieve information through a much wider range of search queries.
15. The emergency detection dictionary now includes additional critical symptom descriptors and their synonymous terms which were introduced to the system through informal clinical review, which helps decrease false-negative detection rates.
16. The document card system now includes per-source text previews which show 350 character limits and similarity percentage badges, which help users assess the origin and significance of their retrieved documents.

17. The system now includes a standard medical disclaimer which clearly states that the system provides non-diagnostic medical information, which officials use to designate the system as an educational tool instead of a medical diagnostic instrument.

CONCLUSION

The research demonstrates that the Retrieval-Augmented Generation (RAG) method enables developers to create AI health chatbots that provide accurate information through transparent operations while keeping expenses at manageable levels. The system achieves its ability to create medical-based responses through the integration of the all-MiniLM-L6-v2 sentence transformer with ChromaDB vector database and Groq LLM API which operates without needing expensive hardware or complicated systems. The system uses stateless design elements to safeguard user information while its emergency detection system functions as an essential security measure. The system benefits from enhanced transparency through its method of providing source citations with every response which lets users check the supplied information. The system operates in educational institutions and rural health information centers and basic consumer health applications because it requires no special hardware and maintains low running expenses. The system demonstrates how reliable AI-based health information can become more accessible through its current implementation.

FUTURE ENHANCEMENTS

Multi-turn Conversation Support

The system allows users to perform multi-turn dialogues through its stateless design which offers optional session-based conversation storage. The approach enhances user satisfaction when dealing with complex health questions because it lets users ask additional questions based on their previous interactions.

Multilingual Support

The system needs to implement multilingual embedding models which include paraphrase-multilingual-MiniLM-L12-v2 through integration. This implementation will allow the system to serve users in Hindi and regional Indian languages and other non-English languages. The system will become accessible to users from different demographic groups who speak various languages.

Voice Interface Integration

The chatbot will gain accessibility to users who have limited reading skills or visual disabilities through SpeechRecognition and pyttsx3 speech-to-text and text-to-speech functions. The system will allow users to operate the system without using their hands in medical and caregiving situations.

IoT Sensor Fusion

The system requires connection to wearable health sensors which include smartwatch heart rate monitors and blood oxygen monitors and glucose monitors. This connection enables users to make context-aware inquiries when their physiological data. gives real-time health data through wearable sensors and users provide text input. The system creates responses that match user needs through its elevated personalization and specific response generation.

Fine-tuned Medical LLM

The replacement of Groq LLM with MedPaLM 2 and BioMedLM and LLaMA3-Med medical models will enhance answer accuracy through their domain-specific design. The medical domain requires precise answers which need both clinical accuracy and exact terminology usages.

ACKNOWLEDGMENT

We wish to express our deepest appreciation to our project guide for the guidance, assistance, and constructive suggestions provided continuously during the entire process of developing this project. It was largely due to these suggestions that we could make our project better. We also extend our heartfelt thanks to all the faculty members who have provided us with the necessary facilities to complete this project.

REFERENCES

- [1] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [2] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," *Proceedings of EMNLP*, 2019.
- [3] ChromaDB Documentation, "ChromaDB Open-Source Embedding Database." [Online]. Available: <https://docs.trychroma.com/>
- [4] Groq API Documentation, "Ultra-Low Latency LLM Inference." [Online]. Available: <https://console.groq.com/docs/>
- [5] Streamlit Documentation, "Streamlit — The Fastest Way to Build Data Apps." [Online]. Available: <https://docs.streamlit.io/>
- [6] K. Singhal et al., "Large Language Models Encode Clinical Knowledge," *Nature*, vol. 620, pp. 172-180, 2023.
- [7] Y. Kang et al., "A Systematic Review of Chatbots in Healthcare," *NPJ Digital Medicine*, vol. 6, 2023.
- [8] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *NAACL-HLT*, 2019.