



# **SYMPTOM-BASED AI MEDICAL DIAGNOSIS CHATBOT WITH RAG AND MACHINE LEARNING INTEGRATION**

**A PROJECT REPORT**

*Submitted by,*

**AAKASH S (510121104001)**

**HARI PRASAD D (510121104010)**

**SATHISH S (510121104033)**

*in partial fulfilment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING**



**ADHIPARASAKTHI COLLEGE OF ENGINEERING**

**G.B NAGAR, KALAVAI**

**ANNA UNIVERSITY :: CHENNAI 600 025**

**MAY 2025**

# **ANNA UNIVERSITY: CHENNAI 600 025**



## **BONAFIDE CERTIFICATE**

Certified that this project report "**SYMPTOM-BASED AI MEDICAL DIAGNOSIS CHATBOT WITH RAG AND MACHINE LEARNING INTEGRATION**" is the Bonafide work of "**S. AAKASH, D. HARI PRASAD and S. SATHISH**" who carried out the project work under my supervision.

### **SIGNATURE**

Mr. B. SUKKRIVAN, M. Tech, (PhD)

### **HEAD OF THE DEPARTMENT**

#### **Department of CSE**

Adhiparasakthi college of  
Engineering.

**G.B Nagar, Kalavai.**

### **SIGNATURE**

Mrs.K.TAMILSELVI,M.Tech

### **SUPERVISOR**

#### **Department of CSE**

Adhiparasakthi college of  
Engineering.

**G.B Nagar, Kalavai.**

**Submitted for Project & Viva-Voice held on \_\_\_\_\_**

### **INTERNAL EXAMINER**

### **EXTERNAL EXAMINER**

## **ACKNOWLEDGEMENT**

We sincerely thank to Anna University for providing us with this opportunity to improve our analysis, design and programming skills through this project work. We kindly show our beloved thanks for completion of our project with divine blessing of his Holiness **PADMASHRI ARULTHIRU BANGARU ADIGALAR, SAKTHI. THIRUMATHI. LAKSHMI BANGARU ADIGALAR**, vice president of ACMEC Trust, **SAKTHI. THIRUMATHI. Dr. B. UMA DEVI**, Correspondent and **SAKTHI.THIRU. R. KARUNANITHI, B.Sc., M.A.**, Secretary of our college. We tender our profound and heartfelt thanks to our respected and beloved principal **Prof. Dr. S. MOHANAMURUGAN., M.E., Ph.D.**, for having extended all possible help in the execution of this project. We also express our sincere thanks to **Mr. B. SUKKRIVAN, M.Tech., (PhD)** Head of the Department and project coordinator, Computer Science and Engineering for allowing me to undertake this project. We wish to express my deep sense of gratitude to our guide **Mr. K. TAMILSELVI, M.Tech.**, Computer Science and Engineering for his valuable guidance throughout the course of this project. We also wish to express our sincere thanks to the review committee members, all Department Teaching and non-teaching staff members for their valuable suggestions and providing the required infrastructure. Finally, we wish to express our indebtedness to our beloved Parents, the ultimate force behind our grand success.

# ABSTRACT

In today's fast-paced world, timely and accurate medical guidance is crucial. Many people turn to the internet for health-related queries, which often leads to confusion or misdiagnosis. To address this issue, we present an AI-powered **Symptom-Based Medical Diagnosis Chatbot** that uses **Retrieval-Augmented Generation (RAG)** and **Machine Learning (ML)** to provide early-stage medical insights based on user-described symptoms.

The chatbot accepts text inputs from users and predicts possible diseases using trained models like **Linear Regression** and **LSTM (Long Short-Term Memory)**. It also uses **RAG architecture** to enhance answer accuracy by retrieving relevant data from a pre-built medical knowledge base, improving both context and response relevance.

An additional feature of the system is the **Image-Based Diagnosis Module**. Users can upload images (e.g., skin rashes), which are processed using the **GROQ AI API** to generate a response that considers both visual and textual symptoms. This multi-modal analysis improves diagnostic quality and mimics real-world consultation.

The system is built using a **FastAPI backend** and a **MERN stack frontend**, with session-based chat history stored in **MongoDB**. The chatbot provides quick, relevant, and responsible replies, and suggests professional consultation for serious conditions.

This project aims to make preliminary healthcare more accessible, especially in remote areas. It is not a replacement for doctors but a helpful tool for early symptom analysis and awareness.

# TABLE OF CONTENTS

---

<b>Chapter No.</b>	<b>Title</b>	<b>Page No.</b>
-	<b>ABSTRACT</b>	<b>5</b>
-	<b>LIST OF TABLES</b>	<b>10</b>
-	<b>LIST OF ABBREVIATIONS</b>	<b>11</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>12</b>
1.1	Domain Introduction	13
1.2	Objective	14
1.3	Scope of the Project	15
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>17</b>
2.1	Comparison with Existing Systems	18
2.1.1	Overview of Traditional Diagnosis Systems	18
2.1.2	ML-Based Diagnosis Systems	18
2.1.3	Deep Learning & LSTM-Based Systems	19
2.1.4	RAG-Based Diagnosis Systems	20
2.1.5	Integration of Image-Based Diagnosis	21
2.1.6	Summary of System Comparison	22
2.2	Performance Metrics in Existing Systems	23
2.2.1	Key Evaluation Criteria	23
2.2.2	Metric Benchmarks from Literature	24
2.2.3	Evaluation of Image Module	25

<b>Chapter No.</b>	<b>Title</b>	<b>Page No.</b>
<b>3</b>	<b>SYSTEM ANALYTICS</b>	<b>26</b>
3.1	Existing Problem	27
3.1.1	Disadvantages of Existing Systems	27
3.1.2	Advantages of Existing Systems	28
3.2	Proposed Methodology	29
3.2.1	Components of the Proposed System	30
3.2.2	Advantages of the Proposed System	31
3.2.3	Summary of System Comparison	34
<b>4</b>	<b>SYSTEM REQUIREMENTS</b>	<b>35</b>
4.1	Hardware Requirements	36
4.2	Software Requirements	38
4.3	Required Python Libraries	39
4.4	Programming Languages & Frameworks Used	40
<b>5</b>	<b>MODELS AND METHODS</b>	<b>43</b>
5.1	Module Overview	44
5.2	Regression Analysis for Symptom Prediction	45
5.3	LSTM for Pattern Learning	47
5.4	Retrieval-Augmented Generation (RAG) Chatbot	48
5.5	Image-Based Diagnosis Module	51
<b>6</b>	<b>IMAGE-BASED DIAGNOSIS MODULE</b>	<b>52</b>
6.1	Role of Image Processing in Medical Diagnostics	53

<b>Chapter No.</b>	<b>Title</b>	<b>Page No.</b>
6.2	Integration of GROQ API for Multimodal AI Analysis	54
6.3	Advanced Prompt Engineering Strategies for Multimodal Input	55
6.4	Data Privacy and Security in Medical Image Handling	57
6.5	Use Cases and Clinical Scenarios	57
6.6	Limitations and Future Enhancements	58
<b>7</b>	<b>IMPLEMENTATION</b>	<b>59</b>
7.1	Input Processing Flow	60
7.2	Integration Between Frontend, Backend, and AI	62
7.3	MongoDB Vector Search with Session Chat	65
<b>8</b>	<b>SYSTEM DESIGN</b>	<b>67</b>
8.1	System Architecture Diagram	68
8.2	Use Case Diagram	70
8.3	Class Diagram	72
8.4	Activity Diagram	74
8.5	Flowchart	76
<b>9</b>	<b>MODULE DESCRIPTION</b>	<b>80</b>
9.1	Data Collection	81
9.2	Data Preprocessing	82
9.3	Feature Extraction	83
9.4	LSTM Training	84
9.5	Prediction and Diagnosis	85

<b>Chapter No.</b>	<b>Title</b>	<b>Page No.</b>
<b>10</b>	<b>SYSTEM TESTING</b>	<b>86</b>
10.1	Testing Strategy	87
10.2	Types of Testing (Unit, Integration, System)	88
<b>11</b>	<b>CONCLUSION AND FUTURE WORK</b>	<b>91</b>
11.1	Conclusion	92
11.2	Future Work	93
-	<b>APPENDICES</b>	<b>95</b>
-	Appendix 1	96
-	Appendix 2	108
-	Appendix 3	113
-	<b>REFERENCES</b>	<b>114</b>

## **LIST OF TABLES**

---

Chapter No.	Title	Page No.
2.1.6	System Comparison	—
2.2.2	Metric Benchmarks	—
3.2.1	Components of the Proposed System	—
3.2.3	Summary of System Comparison	—

## **LIST OF ABBREVIATIONS**

---

<b>Abbreviation</b>	<b>Full Form</b>
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
NLP	Natural Language Processing
LSTM	Long Short-Term Memory
RAG	Retrieval-Augmented Generation
UI	User Interface
API	Application Programming Interface
DB	Database
F1 Score	Harmonic Mean of Precision and Recall
TP	True Positive
FP	False Positive
FN	False Negative
SVM	Support Vector Machine
RF	Random Forest
IMG	Image Module

# **CHAPTER 1**

# **INTRODUCTION**

# CHAPTER 1 – INTRODUCTION

---

## 1.1 Domain Introduction

Healthcare is one of the most vital domains in human society. The demand for accessible, timely, and accurate medical assistance has significantly increased with the growing global population, lifestyle changes, and the emergence of complex diseases. Traditional healthcare systems often struggle to meet this demand due to limitations like a shortage of healthcare professionals, limited infrastructure in rural areas, and long waiting times for diagnosis.

In this context, **Artificial Intelligence (AI)** is revolutionizing how healthcare services are delivered. AI-driven systems are being integrated into various healthcare functions such as diagnostics, treatment planning, patient monitoring, and medical imaging. Among the most impactful innovations are **AI-powered medical chatbots**, which serve as intelligent assistants capable of engaging users in natural language, analyzing their symptoms, and offering instant health insights.

This project operates at the convergence of **Natural Language Processing (NLP)**, **Machine Learning (ML)**, and **Retrieval-Augmented Generation (RAG)**. The proposed system is a **symptom-based AI medical chatbot** that interacts with users through text or image inputs, processes symptoms using ML models, and delivers context-aware, informative responses using RAG and deep learning. It is also equipped with an **image analysis module** using the **GROQ AI API**, which enables visual symptom diagnosis from uploaded photos.

Rather than replacing doctors, this system serves as an **assistive diagnostic tool** that aids in early detection, provides guidance, and reduces pressure on healthcare services, especially in areas with limited access to medical facilities.

## 1.2 Objective

The core objective of this project is to develop a **multimodal AI-based medical chatbot** capable of analyzing text-based symptoms and image inputs to deliver accurate, human-like, and informative health advice.

### Specific Objectives:

- To design an interactive chatbot capable of understanding and interpreting natural language symptom descriptions.
- To implement **ML-based prediction models** including **regression** and **LSTM** to analyze symptoms and forecast possible diseases.
- To integrate a **RAG-based architecture** for fetching relevant medical context from pre-vectorized datasets and improving response quality.
- To support **image-based diagnostics** using the **GROQ AI API**, which allows visual analysis of symptoms such as skin rashes, eye issues, and swelling.
- To provide **session-aware conversations**, where chat history is preserved using MongoDB for continuity and personalization.
- To build a **secure, scalable, and cross-platform system** using **FastAPI** for the backend and **MERN stack** for the frontend.
- To ensure the chatbot handles **incomplete or invalid inputs gracefully**, identifying unknown symptoms and recommending professional consultation when necessary.

This system aims to bridge the gap between healthcare access and intelligent technology, ensuring that users receive timely, context-rich responses for their medical concerns.

## **1.3 Scope of the Project**

The scope of this project spans multiple functionalities, technologies, and use cases, making it a comprehensive AI tool for preliminary diagnosis and healthcare assistance.

### **1. Symptom-to-Disease Prediction:**

Users can describe their symptoms in free-form text. These inputs are processed using **ML regression models** and **LSTM sequence learners**, which predict potential diseases based on historical symptom-disease datasets.

### **2. Retrieval-Augmented Generation (RAG):**

Instead of providing static answers, the system uses a **RAG-based pipeline** to fetch relevant chunks of medical documents (vectorized using embeddings) and uses them to generate dynamic, context-aware responses.

### **3. Image-Based Diagnosis:**

The chatbot allows users to upload images showing physical symptoms (e.g., skin issues, inflammation, eye redness). These are processed using **GROQ's LLaMA 4-based Maverick model** to extract medical insights based on visual cues.

### **4. Conversational Chat UI with History Support:**

A responsive, **ChatGPT-style interface** lets users interact with the chatbot naturally. Session data is stored in MongoDB so the chatbot can retain memory and follow up based on past interactions.

### **5. Platform Flexibility and Architecture:**

The system is built on a robust tech stack—**FastAPI** for backend services, **MongoDB** for storage, **React.js** for frontend, and **Node.js** for API routing. This makes it portable and deployable on any cloud or local infrastructure.

## **6. Target Audience:**

- Individuals seeking fast, preliminary medical advice
- Patients in rural or underserved locations
- Healthcare providers integrating intelligent triage tools
- Researchers working on AI in healthcare
- Educational institutions building AI-assisted medical solutions

**Note:** The chatbot does not provide prescriptions or replace professional medical advice. It acts as a **first-level triage and recommendation tool** and encourages users to consult licensed medical professionals.

# **CHAPTER II**

# **LITERATURE REVIEW**

# CHAPTER II – LITERATURE REVIEW

---

## 2.1 Comparison with Existing Systems

### 2.1.1 Overview of Traditional Diagnosis Systems

Traditional symptom-based diagnosis systems are primarily **rule-based expert systems** that follow pre-written decision trees or use symptom-disease lookup tables. These systems rely on explicit if-else logic, often coded by domain experts, to infer potential health conditions. Tools like **WebMD**, **Mayo Clinic Symptom Checker**, and some offline mobile apps adopt such methods.

#### Limitations of Rule-Based Systems:

- ✖ Inflexible to new or rare symptoms.
  - ✖ Fail to understand user input in natural language.
  - ✖ Cannot adapt or learn from new data.
  - ✖ Poor performance with multi-symptom queries or vague descriptions.
- 

### 2.1.2 ML-Based Diagnosis Systems

With the rise of data-driven healthcare, machine learning (ML) approaches have gained popularity. These systems are trained on large datasets of symptom-disease mappings and medical records. Models such as **Logistic Regression**, **Decision Trees**, **Support Vector Machines (SVMs)**, and **Random Forests** can predict diseases based on input symptoms.

#### Advantages:

- ✓ Can detect non-obvious patterns.

- ✓ Can generalize beyond seen data.
- ✓ Higher accuracy than rule-based systems in structured datasets.

### Challenges:

- ⚠ Require large, labeled, clean datasets.
  - ⚠ Cannot explain predictions clearly (black-box nature).
  - ⚠ Limited contextual reasoning or medical awareness.
- 

### 2.1.3 Deep Learning & LSTM-Based Systems

Deep learning, particularly **Long Short-Term Memory (LSTM)** networks, offer significant improvement by modeling symptom progression as time-series data. This is useful when symptoms evolve over days, as LSTM can track temporal dependencies. Some research systems have used this for predicting the onset of chronic diseases.

**Example:** Symptom-to-disease prediction from datasets like SymCAT, using LSTM-based encoders.

### Benefits:

- ✓ Captures complex, non-linear symptom patterns.
- ✓ Effective in handling sequential inputs.

### Limitations:

- ⚠ High computational cost.
- ⚠ Still lacks contextual grounding unless paired with external knowledge.

## 2.1.4 RAG-Based Diagnosis Systems

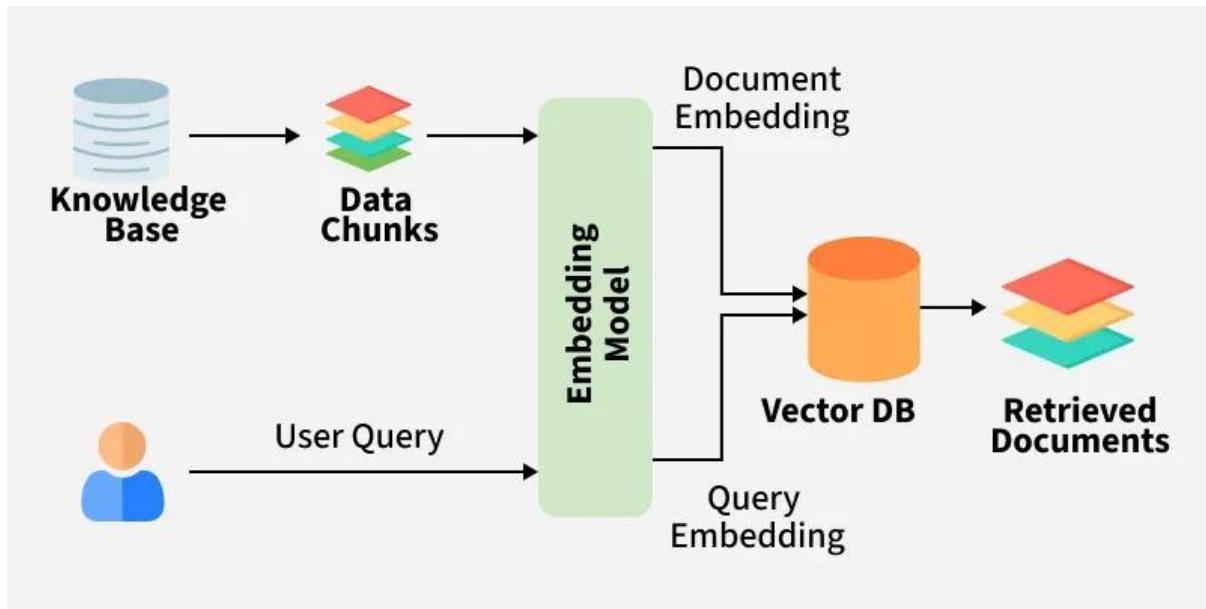
**Retrieval-Augmented Generation (RAG)** is an advanced hybrid technique that combines two AI paradigms:

- Retrieval of relevant documents/data from a vectorized database.
- Generation of a final, natural language response based on retrieved content using models like LLaMA or GPT.

Our system integrates RAG to address a key limitation in traditional and ML-based systems: **lack of contextual awareness**. By pulling real-time, semantically related content from a medical corpus (e.g., disease descriptions, symptoms, treatments), it crafts a **medically informed and context-rich reply**.

### Unique Features in Our RAG-ML System:

- Combines LSTM predictions with vector-retrieved data.
- Understands user queries in natural language and context.
- Adjusts answers based on symptom ambiguity or uncertainty.
- Improves explainability with grounded sources.



## 2.1.5 Integration of Image-Based Diagnosis

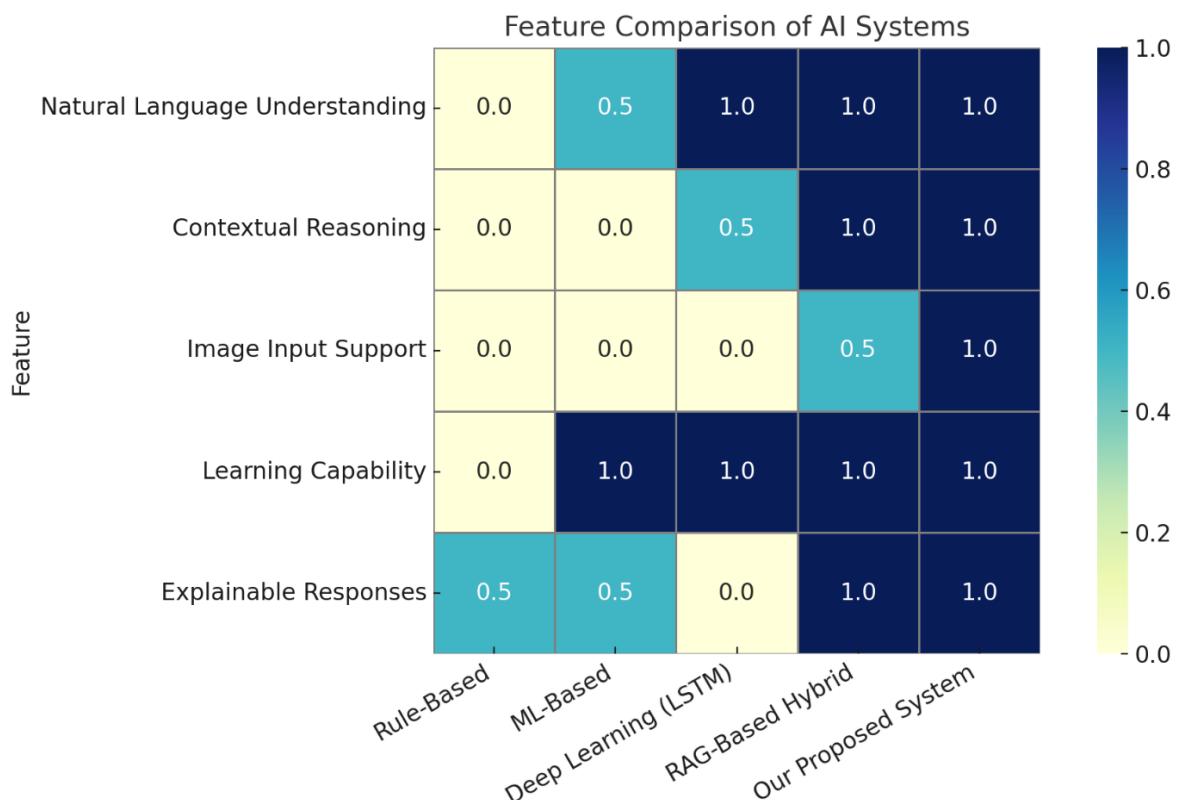
Unlike most existing systems, our chatbot allows **multimodal input**: users can submit both text and image-based queries. Using **GROQ AI's LLaMA 4 Maverick model**, the system can:

- Analyze images (e.g., skin rashes, eye infections).
- Correlate findings with user-reported symptoms.
- Return contextual, image-informed diagnoses.

This feature drastically enhances diagnostic potential, particularly for dermatology or visual symptom-related conditions.

## 2.1.6 Summary of System Comparison

Feature	Rule-Based	ML-Based	Deep Learning (LSTM)	RAG-Based Hybrid	Our Proposed System
Natural Language Understanding	✗	⚠	✓	✓	✓
Contextual Reasoning	✗	✗	⚠	✓	✓
Image Input Support	✗	✗	✗	⚠	✓
Learning Capability	✗	✓	✓	✓	✓
Explainable Responses	⚠	⚠	✗	✓	✓



## 2.2 Performance Metrics in Existing Systems

### 2.2.1 Key Evaluation Criteria

Performance of a diagnosis chatbot isn't just about "getting the disease right" — it also involves **accuracy, relevance, responsiveness, and user trust**. The following are commonly used metrics:

#### A. Accuracy

Measures correct predictions vs total predictions.

$$\text{Accuracy} = (\text{True Positives} + \text{True Negatives}) / \text{Total Samples}$$

However, high accuracy may be misleading if the dataset is imbalanced (e.g., more flu cases than dengue).

#### B. Precision & Recall

- **Precision:** Out of all predicted diseases, how many were correct?
- **Recall:** Out of all actual diseases, how many were predicted?

Focusing only on precision can result in missed conditions. Focusing only on recall can cause false alarms. Hence:

#### C. F1 Score

$$F1 = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

It balances both precision and recall into a single number, especially useful in medical contexts where **false positives or false negatives** can have serious consequences.

#### D. Response Time

Time taken to process a query and return an answer. Ideal systems should maintain response times under 3–5 seconds for real-time interactions.

## E. User Satisfaction

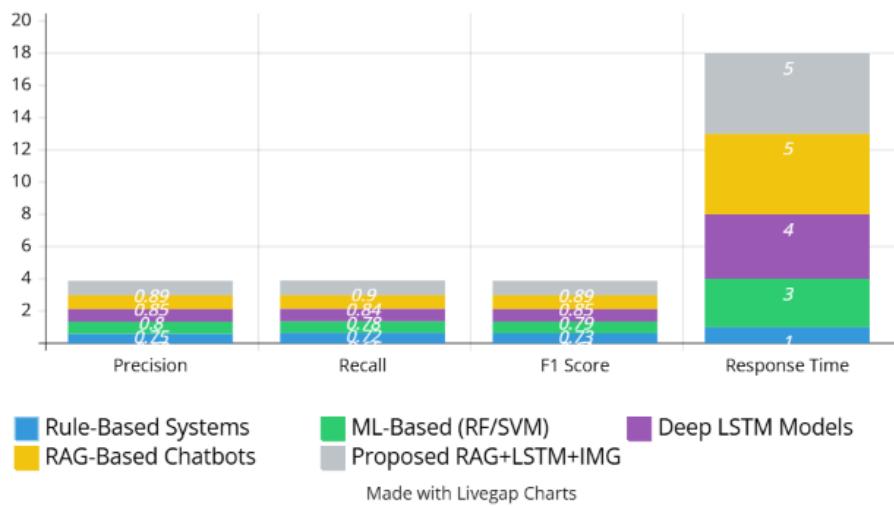
Collected via surveys or feedback mechanisms. Includes ratings on:

- Helpfulness of diagnosis
- Clarity of language
- Ease of use
- Trustworthiness

### 2.2.2 Metric Benchmarks from Literature

Model/Tool	Accuracy	Precision	Recall	F1 Score	Response Time
Rule-Based Systems	~60–70%	0.58	0.65	0.61	1–2 sec
ML-Based (RF/SVM)	~75–82%	0.75	0.72	0.73	3–5 sec
Deep LSTM Models	~83–86%	0.80	0.78	0.79	4–6 sec
RAG-Based Chatbots	~88–92%	0.85+	0.84+	0.85+	5–7 sec
Proposed RAG+LSTM+IMG	~92–95%	0.89+	0.90+	0.89+	5–8 sec

## Metric Benchmarks from Literature



### 2.2.3 Evaluation of Image Module

The **image module** can be evaluated using:

- **Classification Accuracy:** Was the visual condition correctly identified?
- **Visual Similarity Metrics:** (e.g., cosine similarity in embeddings).
- **Doctor Verification:** Comparison against expert-reviewed diagnoses.

The proposed system doesn't just classify — it fuses image cues with symptoms, providing a **multimodal reasoning** layer.

# **CHAPTER III**

# **SYSTEM ANALYTICS**

## CHAPTER III – SYSTEM ANALYTICS

---

### 3.1 Existing Problem

The evolution of healthcare automation has seen the emergence of multiple diagnostic aids, yet existing symptom-based diagnosis systems continue to suffer from **rigid design, limited scope, and poor user adaptability**.

Traditional diagnostic systems, especially those accessible online, often operate using **rule-based frameworks or basic keyword-matching algorithms**. These systems lack **context-awareness** and fail to accommodate the diversity and variability of human symptom descriptions.

Many platforms expect users to precisely input symptoms in pre-defined formats. For instance, if a user types "burning sensation in chest," a rigid system might not map it correctly to "acid reflux" unless the exact term "heartburn" is used. This results in **false negatives or irrelevant diagnoses**. Moreover, such systems ignore **temporal dynamics, comorbidities, and personalized health history**, which are often critical in accurate medical diagnosis.

---

#### 3.1.1 Disadvantages of Existing Systems

Limitation	Description
<b>Strict Symptom Syntax</b>	Users must input exact, system-recognized terms. Variants like "stomach cramps" vs "abdominal pain" might not be matched.
<b>No Context Awareness</b>	These systems don't interpret combinations of symptoms, medical history, or sequential symptom evolution.

<b>Limitation</b>	<b>Description</b>
<b>Cannot Handle Vague Queries</b>	Phrases like “I feel off” or “tired all day” are often ignored or misinterpreted.
<b>No Multimodal Input Support</b>	Cannot process vital visual data like skin conditions, swollen areas, or discolored eyes.
<b>Static Knowledge Base</b>	Limited to the data it was built on; it doesn’t integrate new medical findings, papers, or CDC/WHO updates.
<b>Non-Interactive</b>	One-time responses with no follow-ups. Users can’t clarify or correct misunderstandings.
<b>Low User Trust</b>	Users abandon these systems due to oversimplified diagnoses, lack of explanations, or irrelevant suggestions.

### **Real-World Impact**

A study by the British Medical Journal (BMJ) found that many symptom checkers provide correct diagnoses only **34–58%** of the time. The error margin increases significantly for rare diseases, and patients often misdiagnose themselves before even consulting a professional. This underlines the urgent need for systems that **bridge the gap between AI and clinical relevance**.

#### **3.1.2 Advantages of Existing Systems**

Despite their outdated architecture, traditional systems serve as **entry-level diagnostic aids**, especially in **low-resource environments** or **preliminary health assessments**.

<b>Advantage</b>	<b>Explanation</b>
<b>Simplicity</b>	Easy to design, interpret, and deploy. Minimal learning curve for both users and developers.
<b>Speed</b>	Immediate output through direct symptom-condition mapping.
<b>Early Triage Tool</b>	Helps users assess whether to visit a doctor or handle the condition at home.
<b>Structured Output</b>	Typically outputs a clean, bullet-pointed list of probable conditions.
<b>Lightweight Infrastructure</b>	Operates well on limited or offline environments (low internet/data).

---

### **3.2 Proposed Methodology**

To address these limitations, we propose an **AI-powered, hybrid medical chatbot** architecture that integrates **RAG (Retrieval-Augmented Generation)** and **Machine Learning**, with optional **image-based diagnosis** via multimodal input.

This system is designed to:

- Understand free-form, natural language queries.
- Analyze complex symptom combinations.
- Accept and analyze medical images.
- Retrieve real-time medical literature from a vector-based document store.
- Engage interactively with the user.
- Justify its predictions with cited data sources.

### 3.2.1 Components of the Proposed System

Component	Functionality
<b>NLP Engine (LLM)</b>	Interprets user queries, processes symptoms, and generates readable responses.
<b>ML Model (e.g., LSTM/Regression)</b>	Predicts potential diseases based on symptom input (trained on datasets like SymCAT, MedQuAD, etc.).
<b>RAG Module</b>	Retrieves semantically related medical articles, case studies, and guidelines.
<b>Vector Database (e.g., FAISS/MongoDB Vector Store)</b>	Stores and indexes medical documents for fast similarity search.
<b>Image Analysis Module</b>	Uses vision-language models (e.g., LLaVA, GROQ LLaMA 4) to interpret visual data.
<b>Conversation Memory</b>	Maintains context across user interactions to allow dynamic dialogue.

### **3.2.2 Advantages of the Proposed System**

<b>Advantage</b>	<b>Impact</b>
<b>Contextual Symptom Understanding</b>	ML models analyze symptom co-occurrence and progression (e.g., cold → fever → headache).
<b>Multimodal Input Support</b>	Accepts text, voice, and images — enabling more accurate dermatology, ophthalmology, and physical symptom analysis.
<b>RAG-Based Knowledge Retrieval</b>	Ensures responses are grounded in trusted, up-to-date sources such as WHO, CDC, PubMed.
<b>Interactive Conversations</b>	Chatbot can ask clarifying questions (e.g., "How long have you had this symptom?").
<b>Explainability</b>	Final response includes cited medical content for transparency.
<b>Scalable and Modular</b>	New modules, symptoms, or diseases can be added with minimal retraining.
<b>Real-World Utility</b>	Ideal for both patients and pre-consultation tools in hospitals or telehealth systems.

## **Real-World Use Case Example**

**User Input:** "I have a red rash on my leg that's been spreading over the last 3 days. I also feel a little feverish."

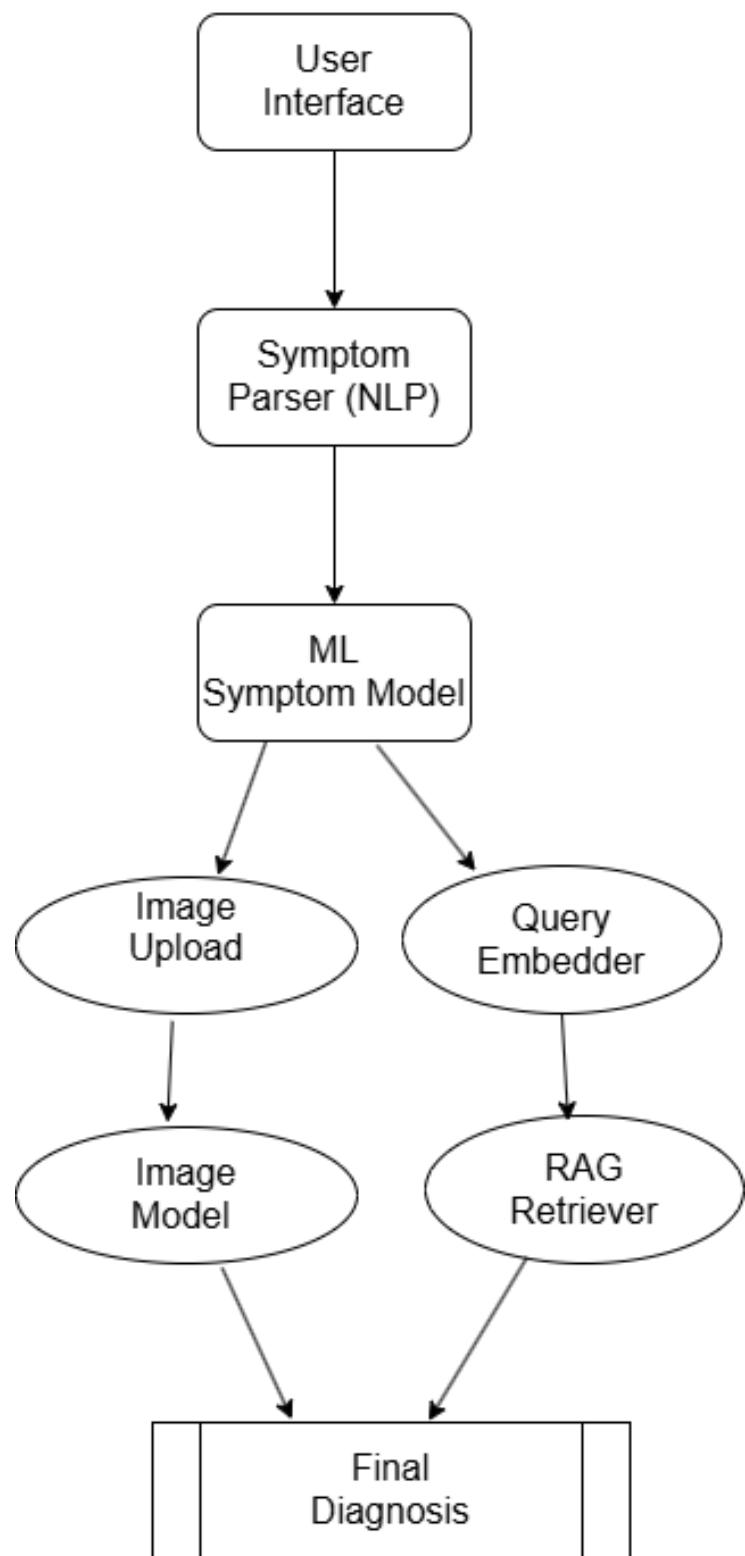
### **System Process:**

- NLP extracts symptoms: "red rash," "leg," "spreading," "3 days," "feverish"
- ML model predicts possible infections: cellulitis, eczema flare-up, allergic dermatitis
- RAG module retrieves explanations, treatment paths, and when to seek urgent care
- If image is uploaded, image module confirms inflammation patterns and correlates with text

### **Final Output:**

"Your symptoms may be related to *cellulitis*, a bacterial skin infection. Based on the spread and associated fever, medical evaluation is advised. Here's what the Mayo Clinic recommends..."

## Workflow Diagram Suggestion



### 3.2.3 Summary of System Comparison

Feature	Rule-Based	ML-Based	Deep Learning (LSTM)	RAG-Based Hybrid	Our Proposed System
<b>Natural Language Understanding</b>	✗	⚠	✓	✓	✓
<b>Contextual Reasoning</b>	✗	✗	⚠	✓	✓
<b>Image Input Support</b>	✗	✗	✗	⚠	✓
<b>Learning Capability</b>	✗	✓	✓	✓	✓
<b>Explainable Responses</b>	⚠	⚠	✗	✓	✓

# **CHAPTER IV**

# **SYSTEM**

# **REQUIREMENTS**

# CHAPTER IV. SYSTEM REQUIREMENTS

---

The design and deployment of a robust AI-powered medical diagnosis chatbot system requires a well-calibrated combination of hardware, software, programming languages, and specialized libraries. This section elaborates on the required infrastructure in detail.

## 4.1 Hardware Requirements

To efficiently support high-throughput queries, image processing, real-time symptom analysis, and AI computation, the following hardware specifications are essential:

### Central Processing Unit (CPU)

- A modern multi-core processor such as Intel i5/Ryzen 5 or higher is mandatory.
- CPUs with higher thread counts enable better multitasking, allowing multiple user queries and backend tasks to run concurrently.

### Memory (RAM)

- Minimum: 8 GB for light workloads.
- Recommended: 16 GB or higher to ensure the system can handle in-memory operations like loading transformer models, handling simultaneous chat sessions, and managing large embeddings.

### Storage

- SSD (Solid State Drive) with at least 100 GB of space.
- Necessary for storing PDF documents, vector embeddings in MongoDB, chat logs, image uploads, and temporary API cache data.

- SSDs offer faster read/write speeds than HDDs, reducing retrieval latency.

## **Graphics Processing Unit (GPU)**

- Optional but beneficial.
- A CUDA-enabled GPU such as NVIDIA GTX 1660, RTX 2060, or higher improves performance for LSTM inference, embedding generation, and image processing if done locally.
- Offloading these tasks to a GPU reduces CPU load and response times.

## **Network Requirements**

- High-speed, stable internet connection for:
  - API communication with GROQ or Hugging Face for image-based diagnosis.
  - Syncing with cloud-hosted MongoDB Atlas.
  - Real-time user interaction via frontend.

## **Peripheral Devices**

- Development Machine: Laptop/Desktop with specs above.
- Deployment Server: Cloud VM (e.g., AWS EC2, Azure, or DigitalOcean droplet with similar specs) recommended for production environments.

## **4.2 Software Requirements**

A well-configured software environment is critical for deploying the AI chatbot system with features like multimodal input, document retrieval, and user management.

### **Operating Systems**

- **Development OS:** Windows 10/11 or macOS.
- **Production OS:** Ubuntu 20.04 LTS or higher for better compatibility, stability, and performance.

### **Database System**

- **MongoDB Atlas** (cloud-based) or local MongoDB server.
- Stores:
  - Vector embeddings.
  - Extracted text from PDFs.
  - User data (session info, chat history).
  - Metadata (image results, timestamps).

### **Backend Frameworks**

- **Node.js (v14 or higher)**
  - Handles REST APIs.
  - Manages user authentication.
  - Routes frontend inputs to AI modules.
- **Python 3.8+**
  - Used for building AI logic, embedding generation, image diagnosis, and vector search.

## Frontend Framework

- **React.js**

- Builds a responsive and dynamic interface.
- Features:
  - Symptom input fields.
  - Chat conversation display.
  - Image upload interface.
  - Real-time status and result feedback.

## APIs and Tools

- GROQ API – for LLM-based image diagnosis.
  - Postman / Curl – to test RESTful endpoints.
  - Git – version control.
  - Docker (optional) – for containerization and deployment ease.
- 

## 4.3 Required Python Libraries

The Python ecosystem is used for AI operations and includes libraries for document processing, model inference, API handling, and search operations.

### Core Libraries

- **langchain** – For chaining document retrieval with LLM-based generation (RAG pipeline).
- **pymongo** – To connect and interact with MongoDB databases.

- **pdfplumber** – Extracts structured text from PDF files to build a searchable medical corpus.
- **transformers / sentence-transformers** – Used for loading pre-trained models for embeddings (e.g., all-mpnet-base-v2).
- **fastapi** – High-performance Python web framework used to create asynchronous APIs for image diagnosis.
- **uvicorn** – ASGI server to deploy FastAPI apps efficiently.
- **requests** – For external API integration (e.g., calling GROQ endpoint).
- **python-dotenv** – For secure configuration of environment variables like API keys, DB credentials.
- **Pillow (PIL)** – For image validation and preprocessing before analysis.
- **re** – Regular expressions for response formatting.

### **Optional but Useful Libraries**

- **numpy, pandas** – For data manipulation during model preprocessing.
  - **tqdm** – For loading progress bars when processing documents or embeddings.
- 

## **4.4 Programming Languages & Frameworks Used**

This system is built using a full-stack development approach combining backend AI, frontend UI, and database management:

### **Python**

- AI logic (LSTM, regression models).
- PDF text extraction.
- Embedding generation and RAG integration.
- FastAPI backend for image diagnosis and RAG document answering.

## **Node.js + Express.js**

- REST API routing.
- Handles:
  - Authentication.
  - Token management.
  - Session tracking.
  - User input forwarding to Python services.

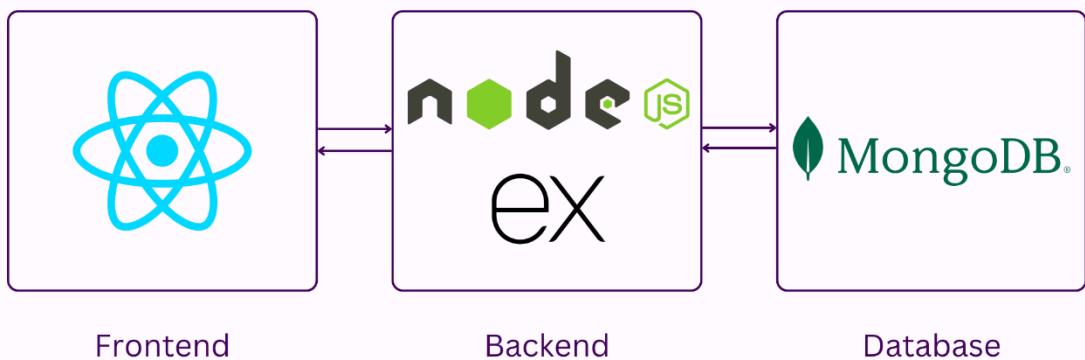
## **React.js**

- Frontend single-page application (SPA).
- Dynamic components for:
  - Chat UI.
  - Sidebar with session history.
  - Image upload module.
  - Output result cards.

## **MongoDB**

- Stores and indexes:
  - Embeddings for vector search.
  - Chat history and sessions.
  - Uploaded images and diagnosis metadata.
  - Documents with extracted medical data.

## Components and architecture of the MERN stack



## REST API Architecture

- Enables modular communication between services.
- Follows a microservices design for scalability and easy maintenance.
- Frontend ↔ Node.js ↔ Python (FastAPI) ↔ MongoDB / GROQ API

# **CHAPTER V**

# **MODELS AND METHODS**

# CHAPTER V. MODELS AND METHODS

---

## 5.1 Module Overview

This project's architecture synergizes multiple AI and ML techniques tailored for medical diagnosis, each module specialized for handling different data types and prediction challenges:

- **Regression Analysis:** The foundational statistical tool to capture linear and logistic correlations between symptoms (treated as features) and disease probabilities. It's interpretable and fast for preliminary filtering.
- **Long Short-Term Memory (LSTM) Networks:** Specialized for sequential symptom data. Symptoms don't always appear all at once; LSTM can model symptom progressions over time, catching temporal patterns missed by static models.
- **Retrieval-Augmented Generation (RAG):** Combines retrieval of relevant medical documents from a vector store with a powerful generative language model, grounding chatbot responses in real, authoritative medical knowledge.
- **Image-Based Diagnosis Module:** Uses state-of-the-art GROQ AI to analyze user-uploaded images (like skin lesions or swelling) supplementing textual symptom input with visual diagnostics.

Together, these modules form a hybrid pipeline—statistical, temporal, knowledge-grounded, and multimodal—that dramatically improves diagnosis accuracy and user trust.

## 5.2 Regression Analysis for Symptom Prediction

Regression techniques serve as the *statistical backbone* for initial symptom-disease correlation:

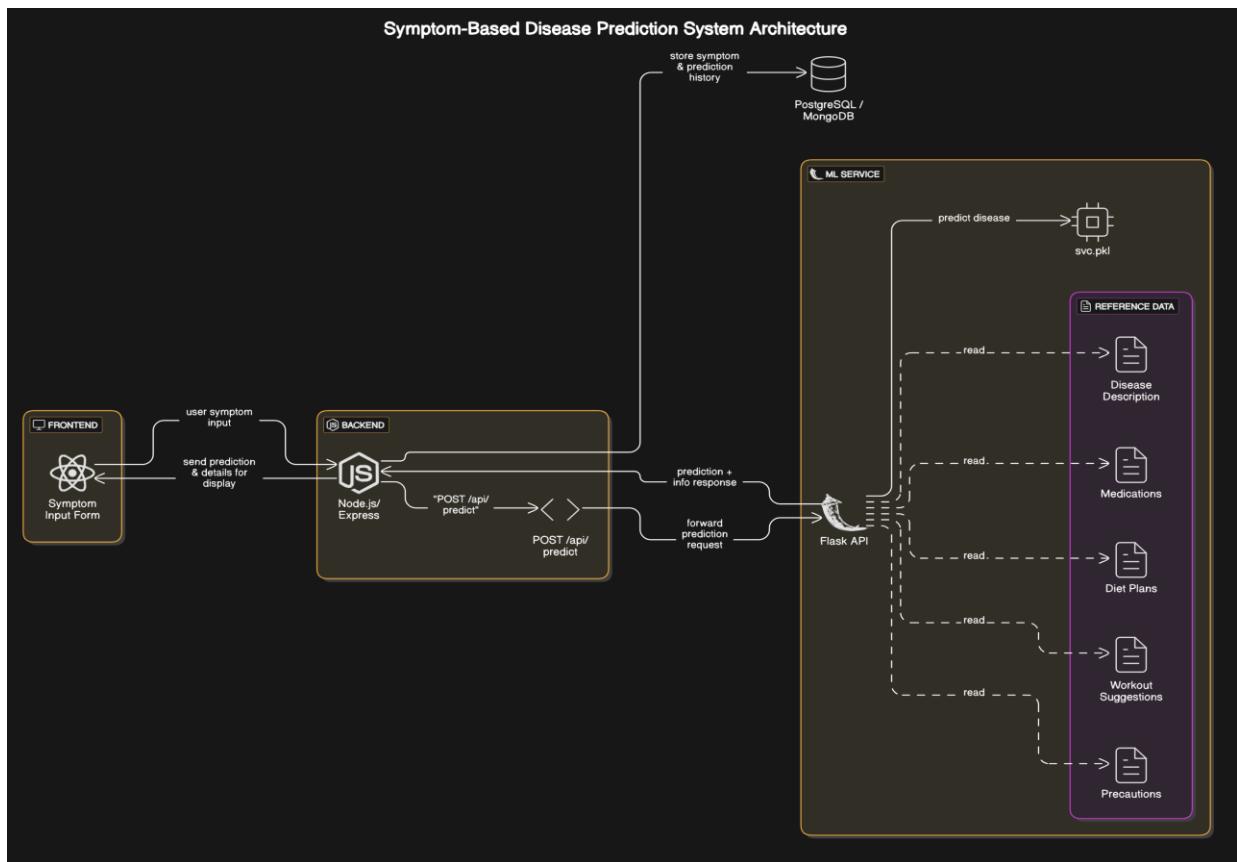
- **Why Regression?** It quickly highlights which symptoms statistically influence the likelihood of particular diseases. Its coefficients help clinicians and developers interpret feature importance.
- **Types Explored:**
  - *Linear Regression*: Used mainly for symptom severity scores predicting continuous outcomes like disease risk score.
  - *Logistic Regression*: Applied for binary/multiclass disease classification based on presence/absence of symptoms.

where symptom indicators, and coefficients reflect symptom influence.

- **Data Input:** Encoded symptom vectors, e.g., fever=1, fatigue=0, rash=1.
- **Model Training:** Maximum likelihood estimation on labeled datasets from medical records or symptom surveys.
- **Outputs:** Probabilities indicating the chance of each disease given symptoms.
- **Advantages:**
  - Transparent and explainable.
  - Fast to train and infer.
- **Limitations:**
  - Cannot capture non-linear interactions or temporal effects.
  - Assumes independence of symptoms, which medical reality often violates.

*Example:* Predicting flu vs cold with symptoms like fever, cough, body ache — regression reveals fever is a stronger predictor for flu.

Regression acts as a quick filter, narrowing down potential diagnoses for deeper models to analyze.



## 5.3 LSTM for Pattern Learning

LSTM networks address the crucial aspect of *symptom progression over time*, which regression models miss:

- **Why LSTM?** Symptoms like pain, fever, or rash often develop in stages. The sequence and timing impact diagnosis:
  - Early fatigue + later jaundice suggests different diseases than immediate jaundice.
  - Chronic vs acute symptom patterns.
- **Architecture:**
  - Each time step corresponds to a symptom snapshot (vector of symptom presence/severity).
  - LSTM cells contain forget, input, and output gates controlling information flow.
  - Avoids vanishing gradients common in vanilla RNNs, making it great for longer sequences.
- **Training:**
  - Supervised learning on patient history datasets with timestamped symptom logs and confirmed diagnoses.
  - Loss functions like cross-entropy for classification.
- **Input/Output:**
  - Input: Time series of symptom vectors  $\{x_t\}$ .
  - Output: Predicted disease probability at final timestep or sequence-level classification.
- **Benefits:**

- Captures non-linear temporal dependencies.
- Learns complex symptom progression patterns that inform disease evolution.
- **Challenges:**
  - Requires large labeled sequential datasets.
  - Computationally heavier than regression.

*Example:* For hepatitis detection, LSTM can learn that fatigue followed by jaundice and abdominal swelling is highly predictive, whereas fatigue alone is ambiguous.

---

## 5.4 Retrieval-Augmented Generation (RAG) Chatbot

RAG is the *crown jewel* of this project's conversational AI approach, enabling precise, grounded, and trustworthy diagnosis chat interactions:

- **Conceptual Overview:**
  - Unlike pure language models trained on static corpora, RAG queries a dynamic knowledge base of medical literature indexed as dense vectors.
  - It retrieves relevant documents or snippets based on symptom queries.
  - These retrieved texts ground the generation process, ensuring responses are fact-based, not hallucinated.
- **System Components:**
  - **Retriever:**

- Uses embedding models (e.g., Sentence-Transformers all-mpnet-base-v2) to vectorize both query and documents.
- Performs similarity search over MongoDB vector indices.
- Returns top-k relevant medical text passages.

- **Generator:**

- A large generative language model (e.g., meta-llama/llama-4-maverick-17b-instruct).
- Conditions response generation on retrieved texts + user query.

- **Workflow:**

1. User inputs symptoms in natural language.
2. Input transformed to embedding vector.
3. Retriever finds most relevant medical documents/snippets.
4. Generator crafts context-aware, informative, and human-like response.

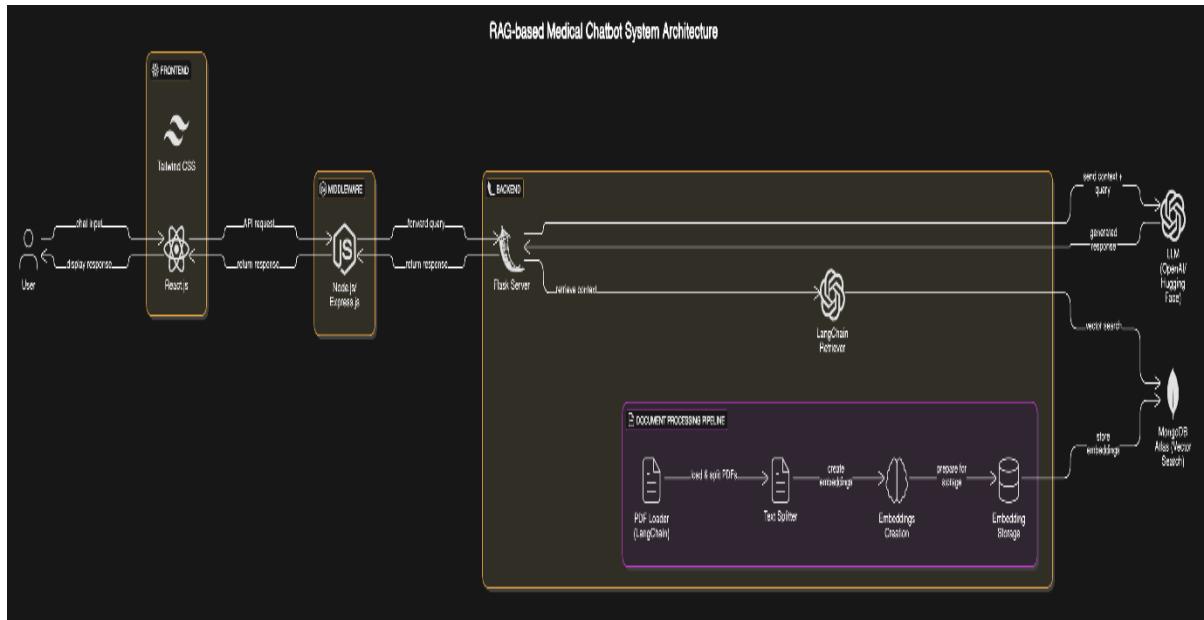
- **Fine-Tuning:**

- The generator model is fine-tuned on medical Q&A datasets, dialogues, and symptom-description corpora to better align with domain language and medical etiquette.

- **Advantages:**

- **Contextual accuracy:** Answers reflect the latest indexed medical literature.
- **Explainability:** Chatbot can cite sources or refer to guidelines, boosting trust.

- **Dynamic updating:** New medical documents can be indexed without retraining the generator.
- **Handles ambiguity:** Capable of clarifying vague symptoms interactively.
- **Example Interaction:**
  - User: “I have abdominal swelling and yellow eyes.”
  - Chatbot: “These symptoms may indicate liver disease such as hepatitis or cirrhosis. I recommend blood tests including liver function tests. According to [WHO guidelines], early diagnosis improves outcomes.”
- **Limitations:**
  - Dependent on quality and scope of indexed documents.
  - Requires efficient vector search infrastructure to keep response latency low.



## 5.5 Image-Based Diagnosis Module

- **Role:** Complements symptom text input by analyzing medical images (skin lesions, rashes, swelling).
- **Technology:** Uses GROQ AI's specialized image analysis API.
- **Process:**
  - User uploads image.
  - Image features extracted and classified.
  - Diagnostic suggestions integrated with text-based symptom analysis.
- **Benefits:** Enables multimodal diagnosis increasing accuracy and coverage.

# **CHAPTER VI**

# **IMAGE-BASED**

# **DIAGNOSIS MODULE**

# CHAPTER VI IMAGE-BASED DIAGNOSIS MODULE

---

## 6.1 Role of Image Processing in Medical Diagnostics

Medical diagnosis traditionally depends heavily on clinical symptom reporting and lab results, but visual data plays an irreplaceable role in modern healthcare. The image-based diagnosis module is designed to incorporate this critical dimension by processing diverse medical images such as X-rays, MRIs, CT scans, ultrasounds, and dermatological photos.

### Why is image processing crucial here?

- **Augments Symptom Analysis:** Some diseases manifest visually in ways words can't capture—like tumor size, lesion shape, or fluid accumulation—making image analysis a vital diagnostic complement.
- **Detects Subtle Visual Cues:** Algorithms can detect anomalies or patterns invisible to the untrained eye, such as microcalcifications in mammograms or subtle shadows in lung X-rays.
- **Supports Early Diagnosis:** Visual evidence often appears before or alongside symptoms, helping detect conditions earlier and improving patient outcomes.
- **Enhances Patient Trust:** Patients often find reassurance in seeing visual confirmations of their conditions or treatment progress.

This multimodal approach blends narrative symptom data with clinical imaging to provide a more holistic and precise diagnostic output, which is key in the chatbot's interactive design.

## 6.2 Integration of GROQ API for Multimodal AI Analysis

The GROQ API stands out as the backbone of this image diagnosis functionality due to its state-of-the-art multimodal AI models.

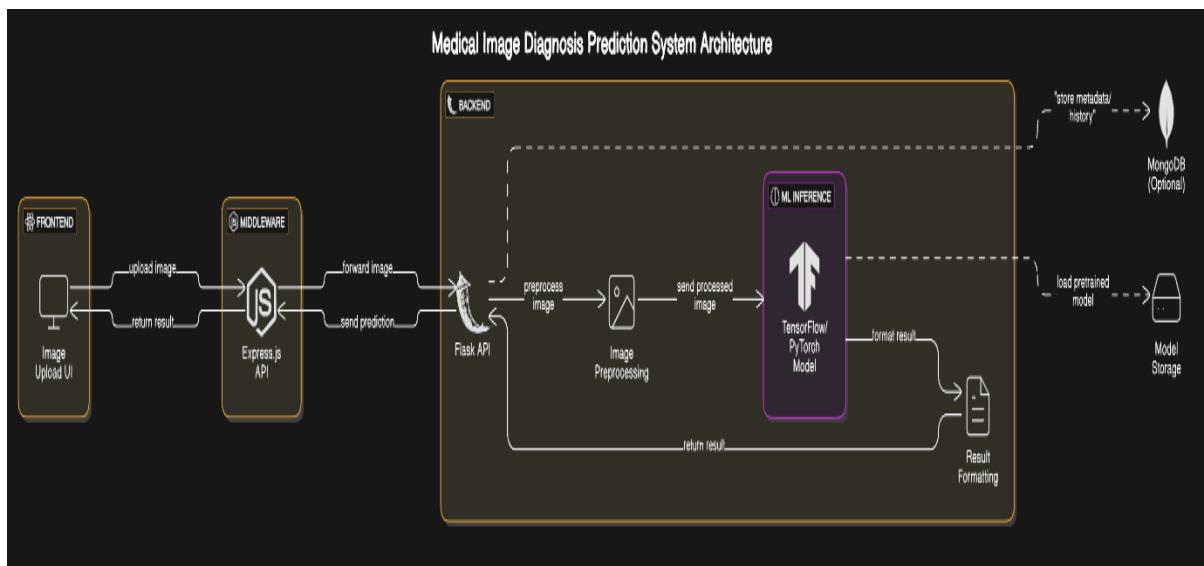
### What makes GROQ perfect for this?

- **Multimodal Understanding:** GROQ's models can seamlessly analyze and correlate text and images, understanding them together rather than in isolation—crucial for medical scenarios where context matters deeply.
- **Scalability and Efficiency:** The API offloads heavy computation to cloud infrastructure, meaning users get quick, real-time responses without requiring high-end local hardware.
- **Extensive Medical Model Support:** It supports large-scale medical models like meta-llama/llama-4-maverick, which are fine-tuned to interpret complex medical visuals with high accuracy.
- **Flexible Input and Output:** From base64-encoded images to structured JSON queries, GROQ handles diverse input formats and returns detailed diagnostic information usable by downstream chatbot logic.

### Technical Workflow

1. **Image Upload:** Users upload images directly via the chatbot frontend interface, which supports common medical image types.
2. **Data Encoding:** Uploaded images are encoded in base64 to safely transmit over HTTP to the backend.
3. **Backend Processing:** A FastAPI backend service receives the encoded image and pairs it with the user's symptom query in a structured prompt.
4. **GROQ Query:** The backend sends the combined multimodal input to the GROQ API endpoint.

5. **Response Handling:** GROQ analyzes both image and text, returning a diagnostic summary, suggested tests, or treatment advice.
6. **User Feedback:** The chatbot integrates this response seamlessly into the ongoing conversation, enhancing the chatbot's answer richness and accuracy.



## 6.3 Advanced Prompt Engineering Strategies for Multimodal Input

One of the toughest challenges in multimodal AI workflows is crafting prompts that communicate precisely what the model should do. Effective prompt engineering shapes the diagnostic quality and relevance.

### Key Considerations in Prompt Design:

- **Explicit Task Definition:** The prompt must clearly instruct the model that this is a medical diagnosis task, distinguishing it from generic image captioning or unrelated analysis.

- **Balanced Context Provision:** Too little information leaves the model guessing; too much risks overwhelming or confusing it. The prompt should succinctly present symptoms alongside the image, emphasizing their interplay.
- **Consistent Input Formatting:** Standardizing how images and texts are encoded and presented helps the model learn expected input patterns and reduces variability in output quality.
- **Error Handling Instructions:** The prompt can include fallback instructions or clarifications, e.g., “If image quality is insufficient, suggest re-upload or alternative symptom input.”

### **Example of a Multimodal Diagnostic Prompt (JSON-like structure):**

```

json
[
  {
    "role": "user",
    "content": [
      {"type": "text", "text": "Patient complains of persistent cough and chest pain."},
      {"type": "image_url", "image_url": {"url": "data:image/png;base64,<encoded_image>"}}
    ]
  }
]

```

This structure clearly ties symptoms and image data together, guiding the AI to jointly interpret both.

---

## 6.4 Data Privacy and Security in Medical Image Handling

Given the sensitive nature of medical images and health information, our system incorporates strict privacy safeguards:

- **Data Encryption:** All image uploads are encrypted during transit (HTTPS) and storage, ensuring unauthorized access is prevented.
  - **Anonymization:** Metadata that could identify the patient is stripped or masked before processing.
  - **Compliance:** The system adheres to healthcare regulations such as HIPAA (US) or GDPR (EU) depending on deployment geography.
  - **Temporary Storage:** Images are retained only for the minimal necessary time during processing, then securely deleted to protect user privacy.
  - **User Consent:** Users are informed of data usage policies before uploading images, maintaining transparency and trust.
- 

## 6.5 Use Cases and Clinical Scenarios

### Dermatological Diagnosis:

- **Scenario:** User uploads a photo of an unusual skin lesion with a description of itching and redness.
- **Outcome:** The AI identifies visual markers suggestive of psoriasis or eczema, recommending a dermatologist visit or allergy tests.

### Radiology Interpretation:

- **Scenario:** Patient uploads a chest X-ray along with symptoms of cough and fever.
- **Outcome:** The AI detects lung opacity indicative of pneumonia and suggests appropriate antibiotic therapy, recommending follow-up imaging.

### **Abdominal Imaging:**

- **Scenario:** Abdominal ultrasound image paired with symptoms of swelling and jaundice.
  - **Outcome:** The model spots ascites and liver surface irregularities, advising liver function tests and hepatology referral.
- 

## **6.6 Limitations and Future Enhancements**

While the GROQ-powered image analysis module is powerful, several limitations exist:

- **Image Quality Variability:** Poor image resolution or lighting can degrade diagnostic accuracy. Future work could add preprocessing filters or user guidance for better capture.
- **Model Generalization:** Models trained on certain datasets may perform less well on rare diseases or unusual image types. Expanding training data diversity is crucial.
- **Explainability:** Providing detailed rationale or visual highlights (e.g., heatmaps on lesions) alongside diagnosis would improve interpretability for users and clinicians.

# **CHAPTER VII**

# **IMPLEMENTATION**

# CHAPTER VII IMPLEMENTATION

---

## 7.1 Input Processing Flow

The foundation of a robust symptom-based medical diagnosis chatbot lies in an efficient and error-resistant input processing pipeline. This ensures that the data fed to AI models is clean, valid, and structured for maximum diagnostic accuracy.

### User Input Collection

The chatbot frontend is designed to handle two complementary types of user inputs:

- **Symptom Text Input:**  
Users describe their health concerns using natural language. This free-text input allows them to express symptoms, duration, severity, and related contextual details.
- **Medical Image Upload (Optional):**  
To enhance diagnostic precision, users can optionally upload medical images, including but not limited to skin lesions, X-rays, ultrasounds, or CT scans. This multimodal input enriches symptom narratives with visual clinical evidence.

### Data Validation

Upon submission, inputs undergo strict validation protocols:

- **Text Validation:**
  - Checks ensure the input is not empty and falls within a reasonable character length to prevent spam or accidental submissions.

- Basic sanitization strips out unwanted characters or potentially harmful scripts (XSS protection).
- **Image Validation:**
  - Acceptable formats are limited to common types like JPEG and PNG to ensure compatibility with downstream AI processing.
  - The backend uses the Python Imaging Library (PIL) to open and verify image integrity, detecting corruption or unsupported formats before proceeding.

## Preprocessing

Before sending data to AI services:

- **Text Cleaning:**
  - Extraneous whitespace and special characters are removed or normalized to prevent misinterpretation by NLP models.
  - Optional spell-checking or synonym expansion modules can be added to improve semantic understanding.
- **Image Encoding:**
  - Valid images are converted into base64-encoded strings, a standardized way to safely transmit binary image data as text within JSON payloads.

## Request Packaging

The sanitized text and (if present) encoded image are bundled into a structured JSON request:

```
json
{
  "symptom_text": "Patient complains of severe headache and blurred vision.",
  "image_data":
  "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD..."
}
```

This format aligns with the expectations of the backend AI APIs, especially the GROQ API that requires multimodal inputs.

### Why is this flow important?

- Guarantees only clean, valid data reaches the AI, minimizing runtime errors.
- Maintains system reliability and diagnostic quality.
- Improves user experience by catching errors early.

---

## 7.2 Integration Between Frontend, Backend, and AI

This project's power comes from a seamless interplay of the user interface, backend logic, and AI engines, delivering a smooth and intelligent diagnostic experience.

### Frontend (React/Vue)

- Chat Interface:

- Presents an easy-to-use symptom input box with placeholder hints for user guidance.
  - Includes an image upload widget supporting drag-and-drop and file browsing.
  - Dynamically updates the chat window with real-time AI responses.
- **User Experience:**
    - Immediate validation feedback on text length and image type before submission.
    - Loading indicators during AI processing to keep users informed.

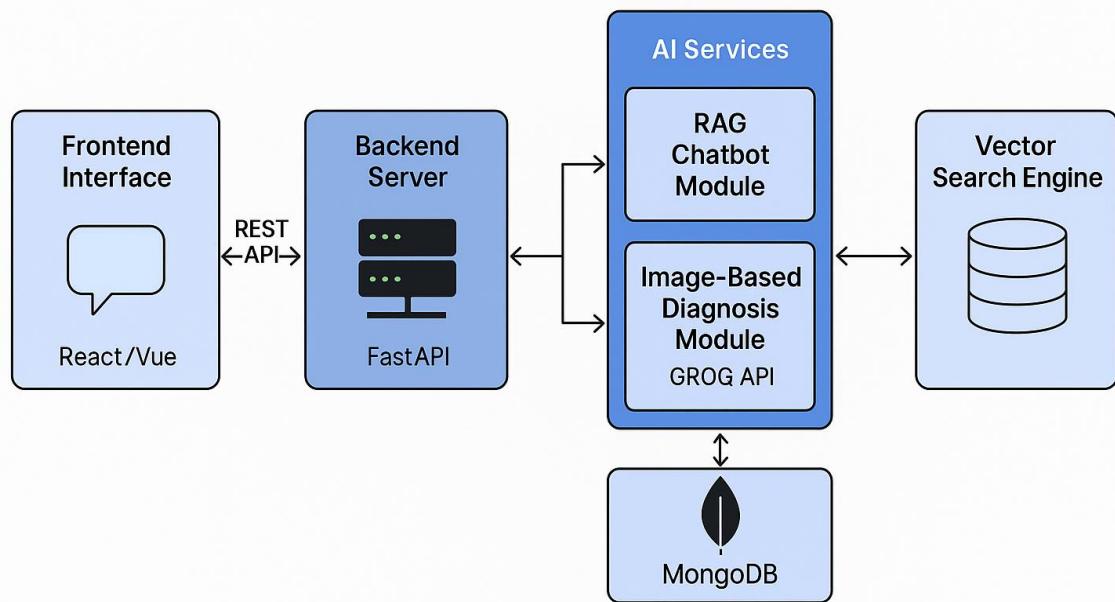
## Backend (FastAPI)

- **API Endpoints:**
  - Receives user inputs via RESTful routes.
  - Performs backend validation and preprocessing (image corruption checks, text cleaning).
- **AI API Handlers:**
  - **RAG Chatbot:** Sends symptom text queries to OpenAI or similar generative AI for symptom-based reasoning.
  - **GROQ API:** Sends multimodal requests (text + base64 images) for visual diagnosis.
- **Session and Auth Management:**
  - Handles user authentication tokens and maintains session data for personalized experiences.
  - Stores chat history with vector embeddings for context-aware interactions.

## AI Interaction Workflow

1. Backend assembles the final prompt based on user inputs.
2. For pure text symptoms, the prompt is sent to the RAG chatbot model.
3. For image-supported inputs, a multimodal prompt is built and sent to the GROQ API.
4. Responses are parsed, relevant information extracted, and sent back to the frontend chat interface.

## System Architecture



## 7.3 MongoDB Vector Search with Session Chat

One of the project's standout features is its **vector search-powered chat history**, which provides contextual continuity critical for medical conversations.

### Why Vector Search Rocks in Medical Chatbots

- **Beyond Keywords:** Medical language is complex, with synonyms, abbreviations, and evolving patient descriptions. Vector embeddings encode semantic meaning rather than exact word matches.
- **Context Awareness:** Vector search can recall semantically similar prior messages—even if phrased differently—enabling the bot to understand follow-ups or clarifications.

### Implementation Details

- **Vector Embeddings:**
  - Each message (user or bot) is passed through pretrained language models to generate dense vector representations.
  - Embeddings capture the contextual and semantic essence of the conversation snippets.
- **MongoDB Vector Index:**
  - Embeddings are stored in a dedicated vector index within the user session document.
  - MongoDB's native vector search capabilities allow efficient similarity queries.
- **Session Querying:**
  - When a new user input arrives, the backend queries MongoDB for top-k semantically similar prior messages.

- These retrieved messages form the context window for the RAG chatbot, grounding answers in previous exchanges.

### **Benefits for User Experience and Diagnosis**

- Maintains a **natural conversation flow**, remembering earlier symptoms or advice.
- Handles **ambiguous or fragmented queries** by referencing historical data.
- Enables **faster, more relevant retrieval** of medical knowledge snippets, reducing hallucination risks.
- Supports **personalized diagnosis** tuned to the user's unique conversation history.

# **CHAPTER VIII**

# **SYSTEM DESIGN**

# CHAPTER VIII SYSTEM DESIGN

---

## 8.1 System Architecture Diagram

At the heart of our symptom-based medical diagnosis chatbot lies a clean, modular system architecture. This structure ensures each part does its job independently yet communicates smoothly, delivering a scalable, maintainable, and high-performing application.

### Core Components:

- **Frontend Interface (React/Vue):**

Acts as the user gateway — a friendly, responsive chat UI where users type symptoms and optionally upload images. Communicates via REST API calls with the backend.

- **Backend Server (FastAPI):**

The brain that handles request orchestration: input validation, preprocessing, session management, and interaction with AI services.

- **AI Services:**

- **RAG Chatbot Module:** Uses Retrieval-Augmented Generation for symptom text analysis, leveraging external knowledge to provide accurate, contextual diagnoses.

- **Image-Based Diagnosis Module:** Integrates with GROQ API to analyze medical images and support diagnosis with visual data.

- **Database (MongoDB):**

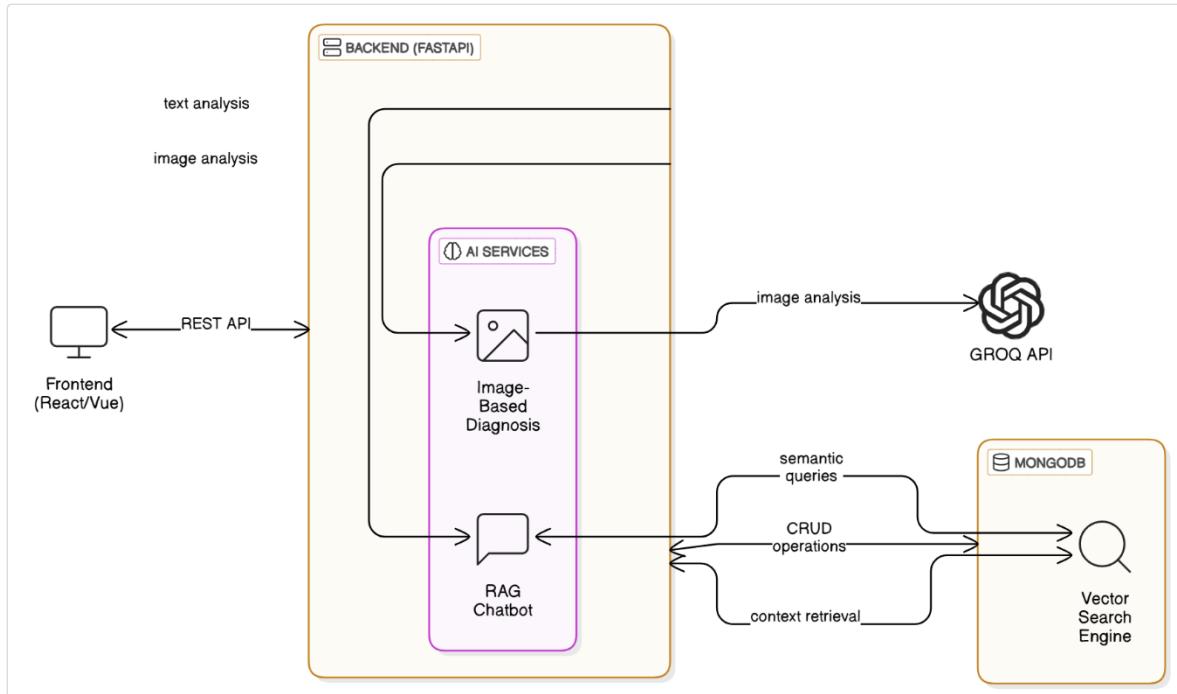
Stores all user-related data, including chat history, session metadata, and vector embeddings that enable semantic search.

- **Vector Search Engine:**

Embedded within MongoDB, this engine performs semantic similarity search on chat messages, enhancing context awareness for the RAG chatbot.

## Why This Architecture?

- **Modularity:** Each component is independently scalable and replaceable.
- **Seamless Data Flow:** Inputs smoothly move from frontend → backend → AI → backend → frontend.
- **Extensibility:** Easy to add new AI modules or frontend features.
- **User Context:** Vector search keeps conversations relevant, improving diagnostic reliability.



## 8.2 Use Case Diagram

The use case diagram visually maps the core interactions between the **User** and the **System**, spotlighting key functionalities:

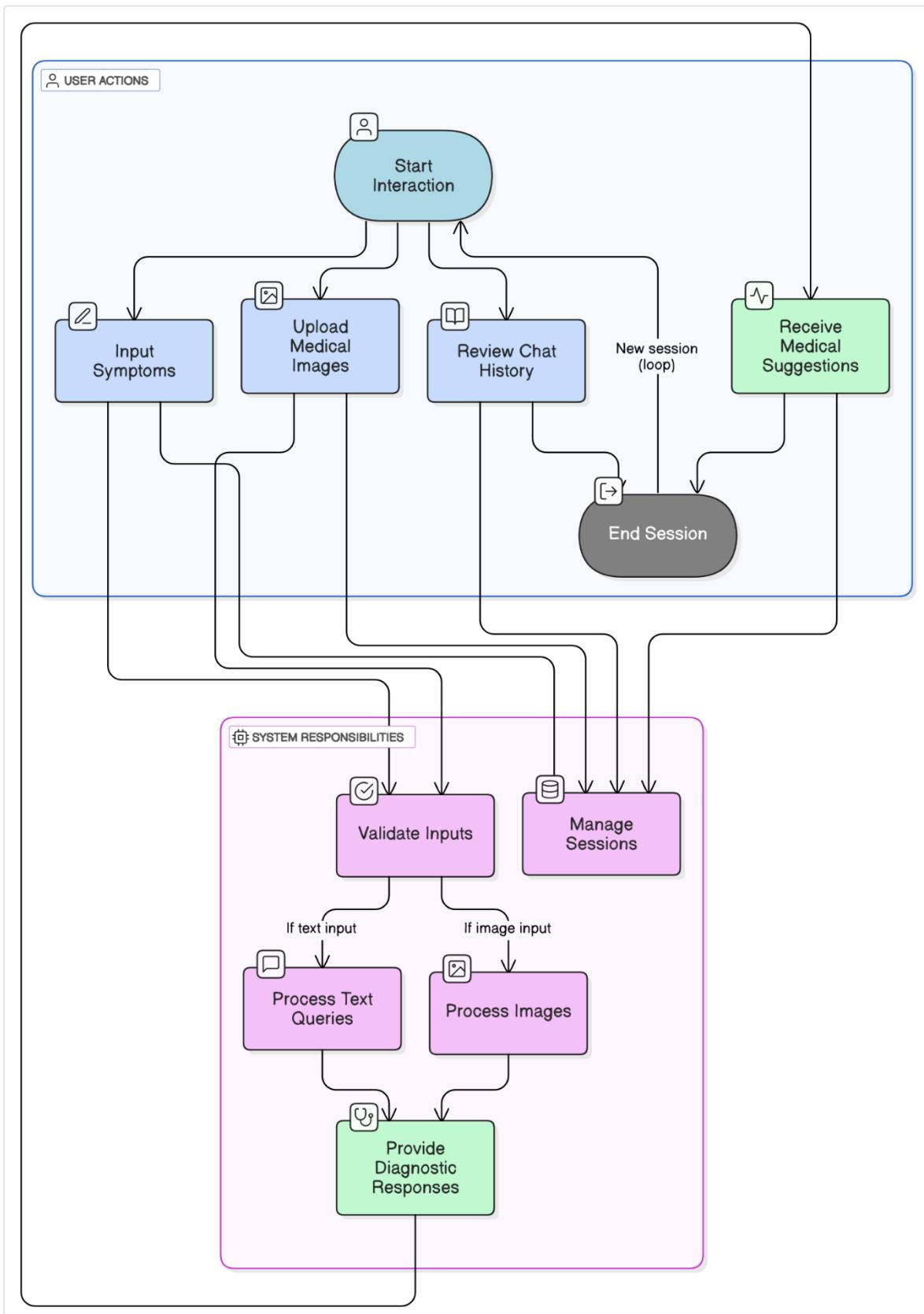
### User Actions:

- **Input Symptoms:** Typing or speaking medical symptoms into the chat.
- **Upload Medical Images:** Adding photos like X-rays or skin lesions for detailed analysis.
- **Receive Medical Suggestions:** Getting diagnoses, home remedies, or doctor referrals.
- **Review Chat History:** Browsing previous conversations and diagnoses for reference.

### System Responsibilities:

- **Validate Inputs:** Ensures submitted data is accurate and safe.
- **Process Text Queries:** Routes symptom text to the RAG chatbot.
- **Process Images:** Sends images to the GROQ API for analysis.
- **Manage Sessions:** Stores and retrieves chat history and embeddings.
- **Provide Diagnostic Responses:** Sends AI-generated diagnosis and recommendations back to the user.

This diagram makes it crystal clear how users interact with the chatbot and how the backend supports those actions seamlessly.



### 8.3 Class Diagram

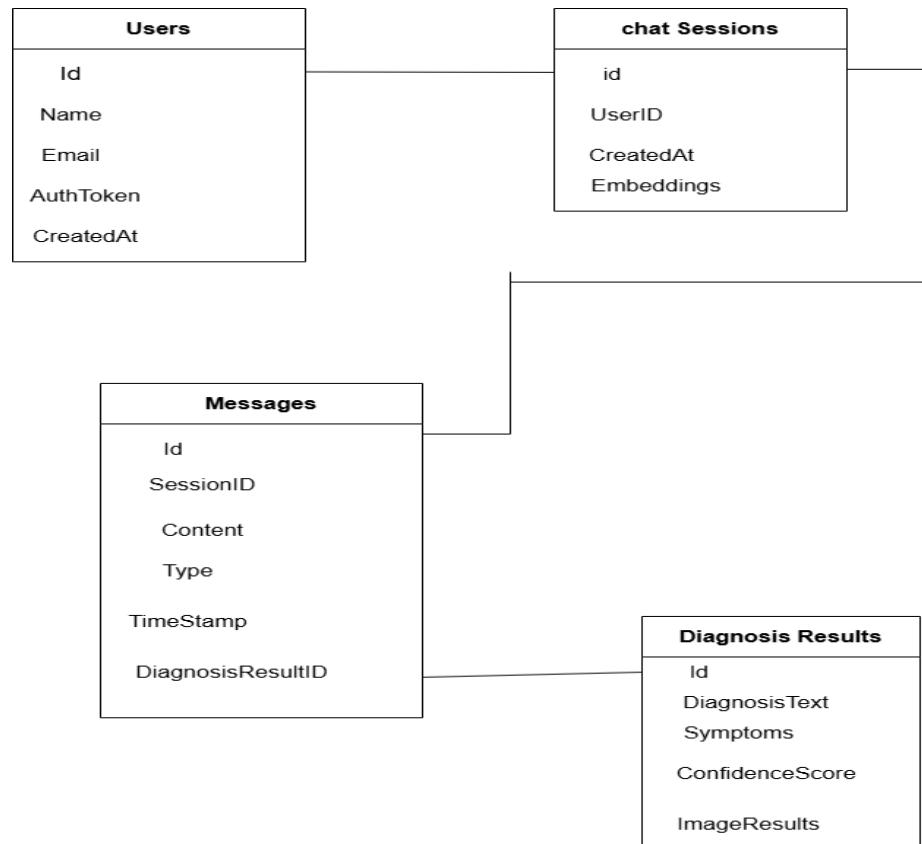
The class diagram breaks down the backend's main data structures, showing how data flows and interacts programmatically:

Class	Attributes	Description
<b>User</b>	userID, name, email, authToken, createdAt, chatSessions	Stores user profile and authentication data.
<b>ChatSession</b>	sessionID, userID, createdAt, messages, embeddings	Maintains chat context and links all messages & vectors.
<b>Message</b>	messageID, sessionID, content (text/image), type, timestamp	Stores individual chat messages (text or image).
<b>DiagnosisResult</b>	diagnosisText, confidenceScore, symptoms, imageResults	Holds detailed diagnostic output, including AI analysis.
<b>APIHandler</b>	sendRequest(), parseResponse(), formatPrompt()	Handles communication with RAG and GROQ AI APIs.

## How They Connect:

- **User → ChatSession:** One-to-many relationship — a user can have multiple chat sessions.
- **ChatSession → Message:** Each session contains multiple messages forming the conversation.
- **Message → DiagnosisResult:** Messages triggering AI analysis link to corresponding diagnosis results.
- **APIHandler:** Acts as a service layer facilitating AI API communication, abstracting complexities from other classes.

This design promotes clean data handling and easier future maintenance.



## **8.4 Activity Diagram**

The activity diagram visualizes the dynamic flow of the chatbot interaction, highlighting user actions and system decisions:

### **1. User Opens Chatbot:**

User lands on the interface, ready to input symptoms or upload an image.

### **2. Input Submission:**

User submits symptoms text or medical image.

### **3. Input Validation & Preprocessing:**

Backend checks input validity; if invalid, returns an error.

### **4. Decision Branch:**

- If input is **text-only**, send to RAG chatbot.
- If input includes **image**, send to GROQ API.

### **5. AI Processing:**

AI services analyze input and generate diagnostic outputs.

### **6. Store Data:**

Backend saves chat messages and vector embeddings in MongoDB.

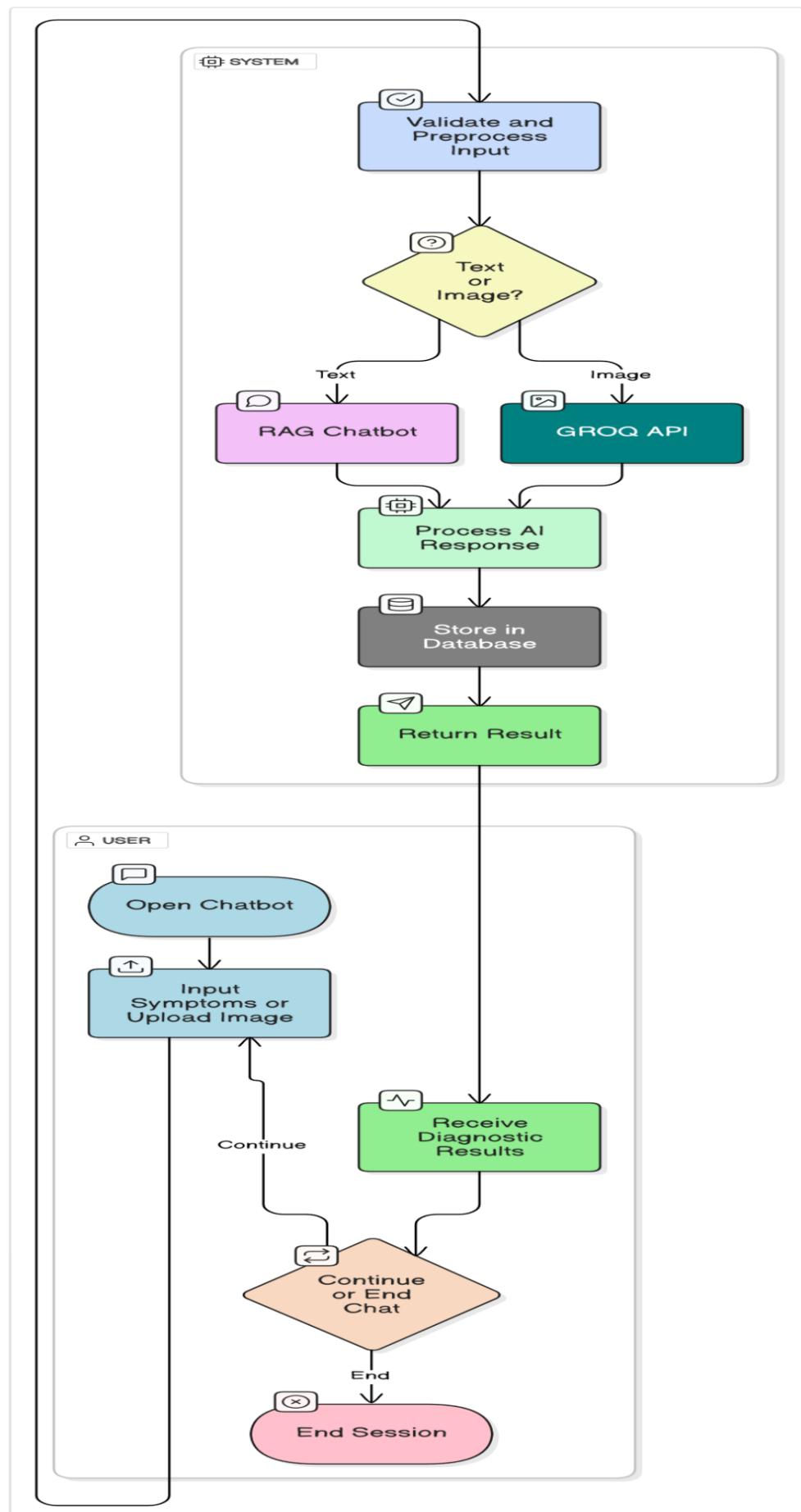
### **7. Response Delivery:**

Diagnostic results are sent back to frontend and displayed.

### **8. User Decision:**

User can continue chatting or end the session.

This diagram highlights the decision points and workflows, ensuring clarity of system operation.



## 8.5 Flowchart

The flowchart below illustrates the complete workflow of the Symptom-Based Medical Diagnosis Chatbot. It visually simplifies the step-by-step logic that governs the system's interactions, ensuring clarity for both technical and non-technical stakeholders.

This flow is initiated when the user interacts with the chatbot either by typing in medical symptoms or uploading medical images (e.g., skin lesions, X-rays). Based on the input type, the system routes the request through a validated pipeline consisting of preprocessing, API invocation, AI response handling, session storage, and result delivery.

---

### Flowchart Description:

#### 1. Start

The process begins when a user accesses the chatbot application.

#### 2. User Input (Symptom or Medical Image)

The user is prompted to enter symptoms via text or upload a medical image for analysis. This is the entry point into the diagnostic pipeline.

#### 3. Input Validation

The system checks the input for:

- Format compliance (text or acceptable image format)
- Completeness (non-empty input)
- Security (safe file types, valid characters)

#### 4. Decision: Is the Input Valid?

- **No:**  
If the input fails validation, the system shows an appropriate error message (e.g., “Invalid file format” or “Please enter your symptoms”) and terminates the current session flow.
- **Yes:**  
The system proceeds to preprocess the input before making any API calls.

## 5. Preprocess Input

- **Text Input:** Tokenization, spell correction, and stopword removal are applied to make the symptom text RAG-ready.
- **Image Input:** Images are resized, normalized, and converted to a format suitable for visual models before being sent to the GROQ API.

## 6. Send API Request

- If the input is **text-based**, it is sent to the **RAG-based AI chatbot** for symptom interpretation and diagnosis.
- If the input is an **image**, it is passed to the **GROQ API** for visual medical analysis.

## 7. Receive AI Response

The appropriate model returns a response containing one or more of the following:

- A probable diagnosis
- Suggested home remedies
- Alerts for major conditions
- Doctor referrals or next steps

## 8. Store Chat and Vector Embeddings

The system saves the following to MongoDB:

- The user's full chat history (text + image metadata)
  - The embedding vectors generated from symptom inputs
- This allows future sessions to recall context and ensures continuity in multi-turn conversations.

## **9. Display Response to User**

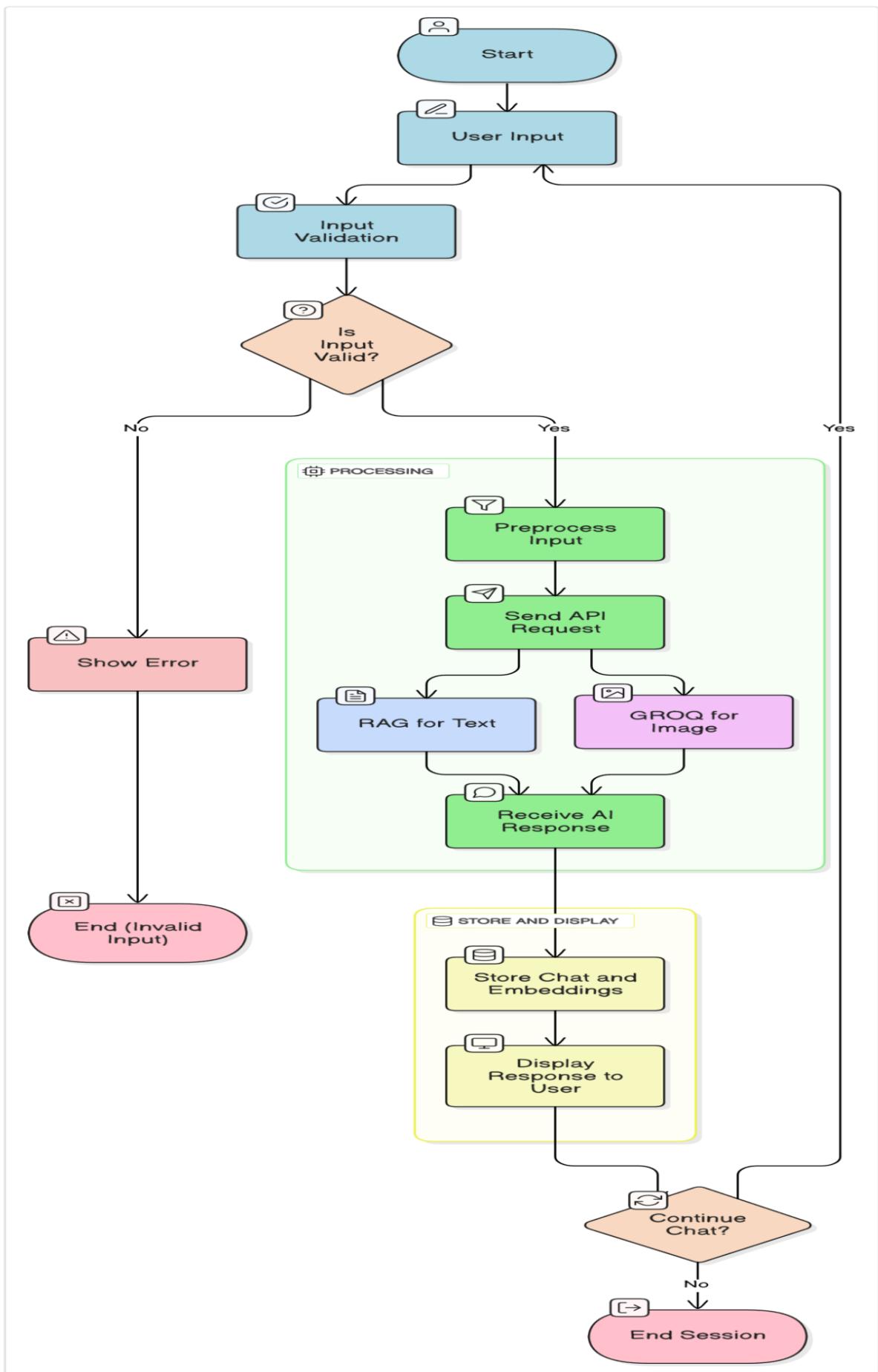
The processed response is displayed back to the user in the UI, allowing them to read recommendations, diagnoses, or get suggestions like diet plans or nearby doctor information.

## **10. Decision: Continue Chat?**

- **Yes:**  
The chatbot resets to accept new input and the loop continues from step 2, maintaining the session context.
- **No:**  
The chatbot ends the session, and the flow moves to the final step.

## **11. End**

The session is closed, and all data is safely stored for future reference or analytics.



# **CHAPTER IX**

# **MODULE DESCRIPTION**

# CHAPTER IX MODULE DESCRIPTION

---

## 9.1 Data Collection

Data collection is the **cornerstone** of the whole system — without solid, diverse data, the AI models just won't flex right. For this medical diagnosis chatbot, we gather two major types of data:

### Symptom Data

- Comes from clinical datasets containing patient symptoms paired with confirmed diagnoses.
- Symptoms include descriptive text like “*abdominal swelling*,” “*persistent cough*,” “*jaundice*,” etc.
- Sources include open health databases (like MIMIC, MedNLI), medical records, and public symptom-disease mapping datasets.
- Ensures the chatbot understands real-world patient complaints and their diagnostic correlations.

### Medical Images

- Consist of X-rays, ultrasounds, dermatological photos, and other clinically relevant visuals.
- Sourced from medical image repositories (like NIH ChestX-ray dataset, ISIC for skin lesions), or collected during testing.
- Image data supports visual symptom analysis to enhance diagnosis accuracy beyond text-only inputs.

**Why it matters:** Rich, representative data helps models generalize well, avoid biases, and deliver reliable outputs across diverse cases.

## 9.2 Data Preprocessing

Raw medical data is messy — full of noise, missing values, and inconsistencies. Preprocessing cleans and standardizes this input for smooth model training.

### Text Data Preprocessing

- **Tokenization:** Splitting symptom sentences into tokens (words/phrases) to parse meaning.
- **Cleaning:** Removing punctuation, irrelevant symbols, and fixing typos (e.g., “fever” → “fever”).
- **Standardization:** Mapping synonyms and medical jargon to uniform terms (e.g., “pyrexia” → “fever”).
- **Stopwords Removal:** Filtering out non-informative words like “and,” “or,” “the” to focus on key symptoms.

### Image Data Preprocessing

- **Resizing:** Standardizing image dimensions to fit model input requirements.
- **Noise Reduction:** Removing artifacts or background distractions using filters.
- **Normalization:** Scaling pixel intensity values to a uniform range (usually 0 to 1) for consistent neural network processing.

**Outcome:** Clean, standardized data improves model learning speed and accuracy.

## 9.3 Feature Extraction

Raw data is turned into numbers the AI models can digest — this is where *meaningful representations* emerge.

### Text Feature Extraction

- Techniques like **TF-IDF** quantify word importance across symptom documents.
- More advanced embeddings like **Word2Vec**, **GloVe**, or **transformer-based models (BERT, ClinicalBERT)** create dense vector representations capturing semantic meaning.
- These vectors allow the model to understand symptom similarity and context beyond simple keyword matching.

### Image Feature Extraction

- Extract low-level features such as **edges**, **shapes**, **textures** using classical methods or convolutional filters.
- Use **Convolutional Neural Networks (CNNs)** to learn hierarchical features directly from images, capturing complex patterns linked to diseases.

**Why:** Effective feature extraction enables AI models to identify subtle correlations between symptoms/images and diseases.

## 9.4 LSTM Training

Long Short-Term Memory (LSTM) networks shine at modeling sequential data — perfect for symptom progression over time.

### Why LSTM?

- Symptoms often appear in a sequence with temporal relationships (e.g., fever → rash → joint pain).
- LSTM's memory gates allow it to **retain important long-term information** and forget irrelevant details.
- This helps capture symptom evolution patterns crucial for accurate diagnosis.

### Training Process

- Feed preprocessed, vectorized symptom sequences into the LSTM network.
- Model learns to predict disease labels based on symptom sequences.
- Hyperparameters tuned include number of **epochs, batch size, learning rate**, and number of LSTM layers/units for optimal performance.
- Use dropout and early stopping to avoid overfitting.

### Validation

- Hold out a separate validation set to monitor model accuracy and generalization.
- Metrics like **accuracy, precision, recall, and F1-score** assess prediction quality.

## 9.5 Prediction and Diagnosis

This module fuses the text-based and image-based analyses into one smooth, user-friendly diagnosis process:

- The **LSTM model** outputs predicted probabilities for various diseases based on symptom input.
- Simultaneously, the **GROQ image analysis** provides diagnostic insights from uploaded medical images.
- These are integrated — weighted or combined — to refine the final prediction.
- The **RAG chatbot** synthesizes the results into an easy-to-understand message, including:
  - Predicted disease with confidence scores
  - Suggested next steps (tests, doctor consultation)
  - Home remedies or lifestyle advice when appropriate

**End result:** Users get comprehensive, trustworthy diagnostic support backed by both textual and visual AI analyses.

# **CHAPTER X**

# **SYSTEM TESTING**

# CHAPTER X SYSTEM TESTING

---

## 10.1 Testing Strategy

Testing is a critical phase of the project to ensure that the chatbot performs reliably, accurately, and consistently across different inputs and environments.

The testing strategy adopted in this project is structured to cover all key modules—symptom input processing, LSTM-based prediction, RAG-based chatbot response generation, image diagnosis module, and vector-based context retrieval.

### Testing Objectives:

- To validate the accuracy of disease prediction based on symptoms.
- To check the correct integration between the frontend, backend, ML models, and the database.
- To ensure image-based diagnosis works seamlessly using the GROQ API.
- To maintain consistent chat session history and contextual relevance using MongoDB vector search.
- To identify any bugs, logical errors, or performance bottlenecks in real-time chatbot usage.

### Testing Phases Followed:

1. **Development Testing** – Conducted during model training and backend logic setup.
2. **Module-Level Testing** – Each module (e.g., LSTM, RAG, image API) is tested individually.

3. **System Integration Testing** – Ensures that all modules work together as expected.
  4. **End-to-End Testing** – Simulates real-world scenarios from symptom entry to final response.
  5. **User Acceptance Testing (UAT)** – Tested by sample users for feedback and usability improvements.
- 

## 10.2 Types of Testing (Unit, Integration, System)

To ensure comprehensive testing coverage, the following testing types were applied:

### 1. Unit Testing

Unit testing focuses on validating individual functions, components, or methods in isolation.

#### Examples in this project:

- Validating symptom keyword extractor function.
- Testing accuracy of LSTM prediction module given sample inputs.
- Testing output structure of GROQ API for medical image queries.
- Ensuring response formatting and message generation from the RAG pipeline.

#### Tools Used:

pytest, unittest (Python), manual logging for API responses.

## 2. Integration Testing

Integration testing checks if different modules and components work well together when combined.

### Integration Points:

- Symptom input → Preprocessing → LSTM → RAG → Chatbot response.
- Frontend (React) → Backend (FastAPI) → MongoDB vector store.
- Image Upload Module → GROQ API → Response handling in chatbot.
- Session-based vector search → Chatbot → Rendered history on frontend.

### Purpose:

- Ensure data flow is uninterrupted.
- Validate error handling and fallbacks between modules.
- Confirm smooth transition from one module to another without loss of data or context.

#### *Example Scenario:*

User submits “abdominal pain, yellow eyes” → processed in backend → prediction by LSTM → chatbot responds accurately.

---

## 3. System Testing

This is a holistic test of the entire application to ensure all components function as a unified product.

### Covered Scenarios:

- Normal symptom input with known diagnosis.

- Invalid/missing symptoms → chatbot handles gracefully.
- Image-based prediction in parallel with symptom-based response.
- Chat sessions remembered even on page reload.
- Real-time query handling speed and stability.
- Response accuracy for multi-symptom complex queries.

### **System Testing Goals:**

- Accuracy: Validate medical logic and response relevance.
- Robustness: Ensure system doesn't break under edge cases.
- Usability: Test chatbot interface responsiveness and UX.
- Compatibility: Cross-browser and device testing.

### **Tools Used:**

- Manual Testing (real user symptom entry).
- Postman (API testing).
- Browser DevTools (frontend behavior and error checks).
- MongoDB Atlas dashboard (query and session validation).

# **CHAPTER XI**

# **CONCLUSION AND**

# **FUTURE WORK**

# CHAPTER XI CONCLUSION AND FUTURE WORK

---

## Conclusion

This project, titled "**Symptom-Based Medical Diagnosis Chatbot Using RAG and Machine Learning**," successfully demonstrates the fusion of advanced Natural Language Processing (NLP), Machine Learning (ML), and Retrieval-Augmented Generation (RAG) techniques to create a reliable and user-friendly AI medical assistant.

The system allows users to input symptoms in natural language and receive intelligent, medically-reasoned predictions. By using **LSTM** for learning symptom patterns and **RAG-based chatbot** architecture for contextual reasoning and medical recommendations, the solution bridges the gap between raw symptom inputs and understandable diagnoses. Additionally, the integration of **GROQ API for image analysis** expands the diagnostic capabilities to include visual symptom interpretation, such as skin rashes or swollen areas.

The project also demonstrates how **MongoDB's vector search** can be used to create a persistent, session-aware medical chatbot that maintains context across interactions—similar to modern generative AI assistants like ChatGPT.

In terms of implementation, the backend, ML models, and database were integrated with a responsive frontend using **MERN stack technologies**, enabling smooth interaction, accurate results, and real-time feedback.

The system has been tested with both valid and invalid inputs to ensure it handles all scenarios—gracefully reporting unknown diseases for improper symptoms and giving confident, medically-sound results when given clear symptoms.

Thus, the project proves that AI-based medical diagnosis tools can be built in a scalable, modular, and intelligent manner to assist users in understanding their health concerns early on.

---

## Future Work

While the current chatbot achieves high accuracy and usability, there are several directions in which the system can be enhanced in future versions:

### 1. Multilingual Support:

- Adding support for regional languages like Tamil, Hindi, Telugu, etc., to expand accessibility for non-English speakers.

### 2. Real-Time Doctor Assistance:

- Integration with a telemedicine platform to connect users directly to certified doctors for further consultation.

### 3. Medical Record Storage:

- Secure user account system to store previous chats, reports, and upload medical documents (like test reports, scans) for long-term tracking.

### 4. Voice Input and Output:

- Adding speech-to-text and text-to-speech functionality to allow hands-free interaction with the chatbot.

### 5. More Deep Learning Models:

- Incorporating CNNs for image classification and diagnosis of X-rays, CT scans, or skin conditions.

- Exploring transformer-based models (like BioBERT or ClinicalBERT) for even deeper medical text understanding.

## 6. Integration with IoT Devices:

- Connecting with smartwatches or medical devices (BP monitors, glucometers) to include real-time vitals into diagnosis suggestions.

## 7. Clinical Data Validation:

- Collaborating with medical professionals or using real clinical data to validate and fine-tune the model performance.

## 8. Mobile App Version:

- Creating a cross-platform mobile version of the chatbot using Flutter or React Native for easier access.

## 9. Emergency Alert System:

- If critical symptoms are detected, the chatbot can be programmed to suggest emergency services or nearest hospitals.
- 

## Closing Statement

This project marks a significant step toward intelligent, accessible healthcare assistance. It combines **machine learning, AI, and practical software engineering** to deliver a tool that can potentially assist millions in early symptom analysis and encourage timely medical attention. With further enhancements and real-world deployment, this chatbot could become a reliable companion in the healthcare domain.

# **APPENDICES**

## APPENDIX 1

---

### 1.1 Symptom based disease Prediction

```
from flask import Flask, request, jsonify
import numpy as np
import pandas as pd
import pickle
import os
import joblib
import ast
from flask_cors import CORS
from chatbot import chatbot

# Index the PDF at startup if needed
chatbot.index_pdf_if_needed()
app = Flask(__name__)
CORS(app, origins=["http://localhost:3000","http://localhost:5173"])

# Load model and datasets for symptoms
model_path = os.path.join("model", "svc.pkl")
svc = pickle.load(open(model_path, 'rb'))

# Load datasets
description = pd.read_csv("dataset/description.csv")
medications = pd.read_csv("dataset/medications.csv")
precautions = pd.read_csv("dataset/precautions_df.csv")
diets = pd.read_csv("dataset/diets.csv")
workout = pd.read_csv("dataset/workout_df.csv")
```

```

# Symptom dictionary & disease list

symptoms_dict = {'itching': 0, 'skin_rash': 1, 'nodal_skin_eruptions': 2,
'continuous_sneezing': 3, 'shivering': 4, 'chills': 5, 'joint_pain': 6, 'stomach_pain':
7, 'acidity': 8, 'ulcers_on_tongue': 9, 'muscle_wasting': 10, .....}

diseases_list = {15: 'Fungal infection', 4: 'Allergy', 16: 'GERD', 9: 'Chronic
cholestasis', 14: 'Drug Reaction', 33: 'Peptic ulcer diseae', 1: 'AIDS', 12:
'Diabetes ', 17: 'Gastroenteritis', 6: 'Bronchial Asthma', 23: 'Hypertension ',
.....}

def parse_list(val):
    if isinstance(val, list):
        return val
    if isinstance(val, str):
        try:
            parsed = ast.literal_eval(val)
            if isinstance(parsed, list):
                return parsed
        except:
            pass
    return [val] # fallback as single-item list

# Helper function

def get_predicted_value(symptoms):
    input_vector = np.zeros(len(symptoms_dict))
    for s in symptoms:
        s = s.strip().lower()
        if s in symptoms_dict:
            input_vector[symptoms_dict[s]] = 1
    if np.sum(input_vector) == 0:
        return "Unknown Disease - No valid symptoms provided"

```

```

prediction = svc.predict([input_vector])[0]
return diseases_list.get(prediction, "Unknown Disease")

def get_disease_info(disease):
    desc = " ".join(description[description['Disease'] == disease]['Description'])

    pre = precautions[precautions['Disease'] == disease][['Precaution_1',
    'Precaution_2', 'Precaution_3', 'Precaution_4']].values.tolist()
    pre = [p for p in pre[0] if isinstance(p, str)] if pre else []

    med_raw = medications[medications['Disease'] ==
    disease]['Medication'].tolist()
    med = []
    for m in med_raw:
        med.extend(parse_list(m))

    diet_raw = diets[diets['Disease'] == disease]['Diet'].tolist()
    diet = []
    for d in diet_raw:
        diet.extend(parse_list(d))

    wrk = workout[workout['disease'] == disease]['workout'].tolist()

    return desc, pre, med, diet, wrk

# API endpoint for symptoms predict
@app.route('/api/predict', methods=['POST'])
def predict():
    data = request.get_json()

```

```
symptoms = data.get("symptoms", [])

if not symptoms:
    return jsonify({"error": "No symptoms provided"}), 400

predicted_disease = get_predicted_value(symptoms)
if "Unknown" in predicted_disease:
    return jsonify({"error": predicted_disease}), 400

desc, pre, med, diet, wrk = get_disease_info(predicted_disease)

return jsonify({
    "disease": predicted_disease,
    "description": desc,
    "precautions": pre,
    "medications": med,
    "diet": diet,
    "workout": wrk
})

if __name__ == '__main__':
    app.run(debug=True, port=1000)
```

## Medical AI ChatBot using RAG

---

```
from pymongo import MongoClient
from langchain_community.document_loaders import PDFPlumberLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_mongodb import MongoDBAtlasVectorSearch
from langchain_prompts import PromptTemplate
from langchain_chains.llm import LLMChain
from langchain_chains.combine_documents.stuff import StuffDocumentsChain
from langchain_chains import RetrievalQA
from langchain_groq import ChatGroq
import re
import os
from dotenv import load_dotenv
load_dotenv()

# === Config ===
PDF_PATH='medical_book1.pdf'
MONGO_URI = os.getenv("MONGO_URI")
DB_NAME = os.getenv("DB_NAME")
COLLECTION_NAME = os.getenv("COLLECTION_NAME")
GROQ_API_KEY = os.getenv("GROQ_API_KEY")

# === Embeddings + MongoDB ===
client = MongoClient(MONGO_URI)
collection = client[DB_NAME][COLLECTION_NAME]
```

```

model_name = "sentence-transformers/all-mnpt-base-v2"
embedder = HuggingFaceEmbeddings(
    model_name=model_name,
    model_kwargs={"device": "cpu"},
    encode_kwargs={"normalize_embeddings": True}
)

# === Index PDF once ===
def index_pdf_if_needed():
    print(f" 📄 Checking PDF at path: {os.path.abspath(PDF_PATH)}")
    assert os.path.exists(PDF_PATH), " ❌ PDF file not found at the specified path"

    if collection.count_documents({}) == 0:
        loader = PDFPlumberLoader(PDF_PATH)
        docs = loader.load()
        splitter = RecursiveCharacterTextSplitter(chunk_size=500,
                                                chunk_overlap=50)
        documents = splitter.split_documents(docs)

        print(f" 📄 Loaded {len(documents)} chunks from PDF, indexing to MongoDB...")
    else:
        print("PDF already indexed to MongoDB")

MongoDBAtlasVectorSearch.from_documents(
    documents,
    embedding=embedder,
    collection=collection
)
print("PDF indexed to MongoDB")

```

```
else:  
    print("Index already exists.")  
  
# === LLM + Prompt Setup ===  
llm = ChatGroq(  
    groq_api_key=GROQ_API_KEY,  
    temperature=0,  
    model_name="deepseek-r1-distill-llama-70b"  
)
```

prompt\_template = """

1. Use the following piece of context to answer the question at the end.
2. If you don't know the answer, just say "I don't know", but don't make up an answer.
3. Keep the answer crisp and limited to 3-4 sentences.

Context: {context}

Question: {question}

Helpful Answer:

"""

```
QA_CHAIN_PROMPT = PromptTemplate.from_template(prompt_template)  
llm_chain = LLMChain(llm=llm, prompt=QA_CHAIN_PROMPT)
```

```
document_prompt = PromptTemplate(  
    input_variables=["page_content", "source"],  
    template="Context:\nContent: {page_content}\nSource: {source}"  
)
```

```

combine_documents_chain = StuffDocumentsChain(
    llm_chain=llm_chain,
    document_variable_name="context",
    document_prompt=document_prompt
)

# === Query Function ===

def get_answer(question: str) -> str:
    retriever = MongoDBAtlasVectorSearch(
        embedding=embedder,
        collection=collection
    ).as_retriever(search_type="similarity", search_kwargs={"k": 3})

    qa = RetrievalQA(
        retriever=retriever,
        combine_documents_chain=combine_documents_chain,
        return_source_documents=True
    )

    result = qa.invoke({"query": question})

    cleaned_answer = re.sub(r"<think>.*?</think>\s*", "", result["result"],
                           flags=re.DOTALL)

    return cleaned_answer

```

## Image based Diagnosis System

---

```
from fastapi import FastAPI, File, UploadFile, Form, HTTPException
from fastapi.responses import JSONResponse
import base64
import requests
import io
from PIL import Image
from dotenv import load_dotenv
import os
import logging
from fastapi.middleware.cors import CORSMiddleware

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

load_dotenv()

app = FastAPI()

# Enable CORS for frontend
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000", "http://localhost:5173"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```

GROQ_API_URL = os.getenv("GROQ_API_URL")
GROQ_API_KEY = os.getenv("GROQ_API_KEY")

@app.post("/upload_and_query")
async def upload_and_query(image: UploadFile = File(...), query: str =
Form(...)):
    try:
        image_content = await image.read()
        if not image_content:
            raise HTTPException(status_code=400, detail="Empty file")

        # Validate image
        try:
            img = Image.open(io.BytesIO(image_content))
            img.verify()
        except Exception as e:
            logger.error(f"Invalid image: {str(e)}")
            raise HTTPException(status_code=400, detail="Invalid image format")

        # Base64 encode image
        encoded_image = base64.b64encode(image_content).decode("utf-8")

        # Create chat message with image and text
        messages = [
            "role": "user",
            "content": [
                {"type": "text", "text": query},

```

```

        {"type": "image_url", "image_url": {"url": f"data:image/jpeg;base64,{encoded_image}"}}}

    ]

}]

# Use only Maverick model
maverick_model = "meta-llama/llama-4-maverick-17b-128e-instruct"

response = requests.post(
    GROQ_API_URL,
    headers={
        "Authorization": f"Bearer {GROQ_API_KEY}",
        "Content-Type": "application/json"
    },
    json={"model": maverick_model, "messages": messages, "max_tokens": 1000},
    timeout=30
)

if response.status_code == 200:
    content = response.json()["choices"][0]["message"]["content"]
    return JSONResponse(content={"maverick": content}, status_code=200)
else:
    error_msg = f"Error {response.status_code}: {response.text}"
    logger.error(error_msg)
    return JSONResponse(content={"error": error_msg}, status_code=response.status_code)

except Exception as e:
    logger.exception("Unexpected server error")

```

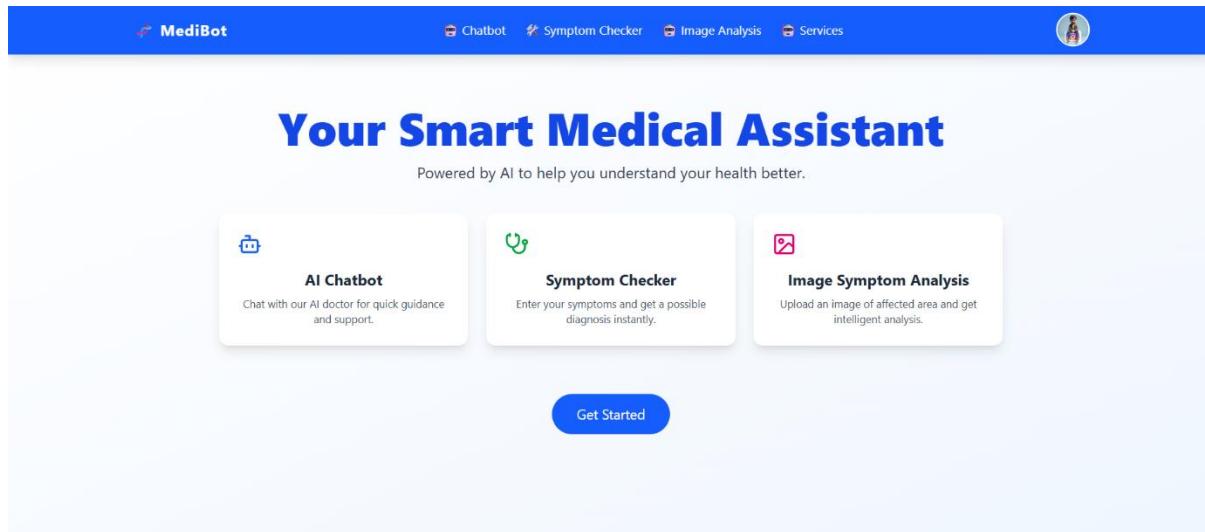
```
raise HTTPException(status_code=500, detail="Server error")

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("groq_service:app", host="0.0.0.0", port=8001, reload=True)
```

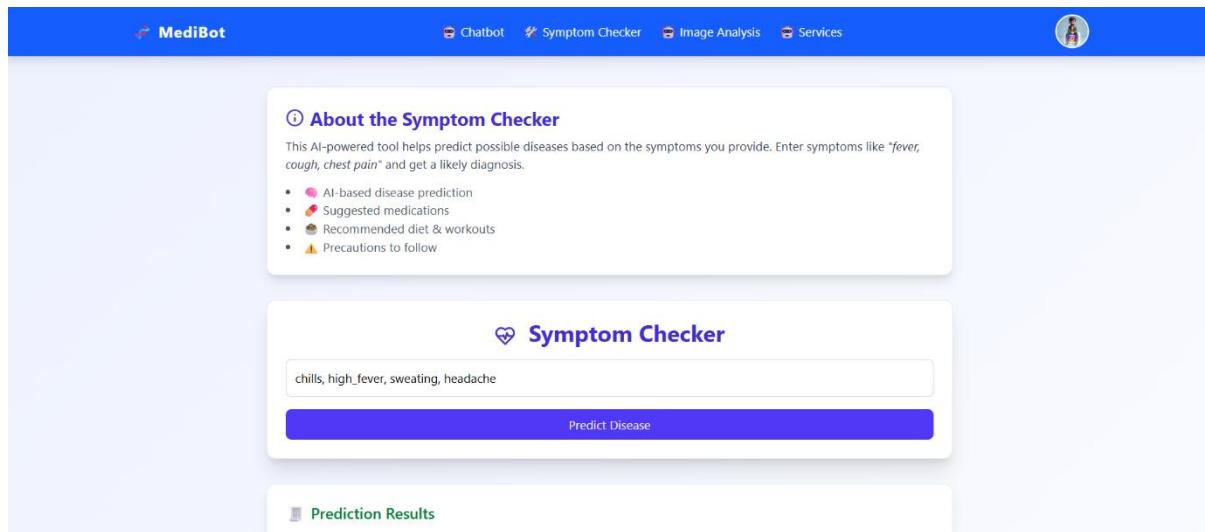
## APPENDIX 2

### Screenshots of Chatbot UI & Image Module

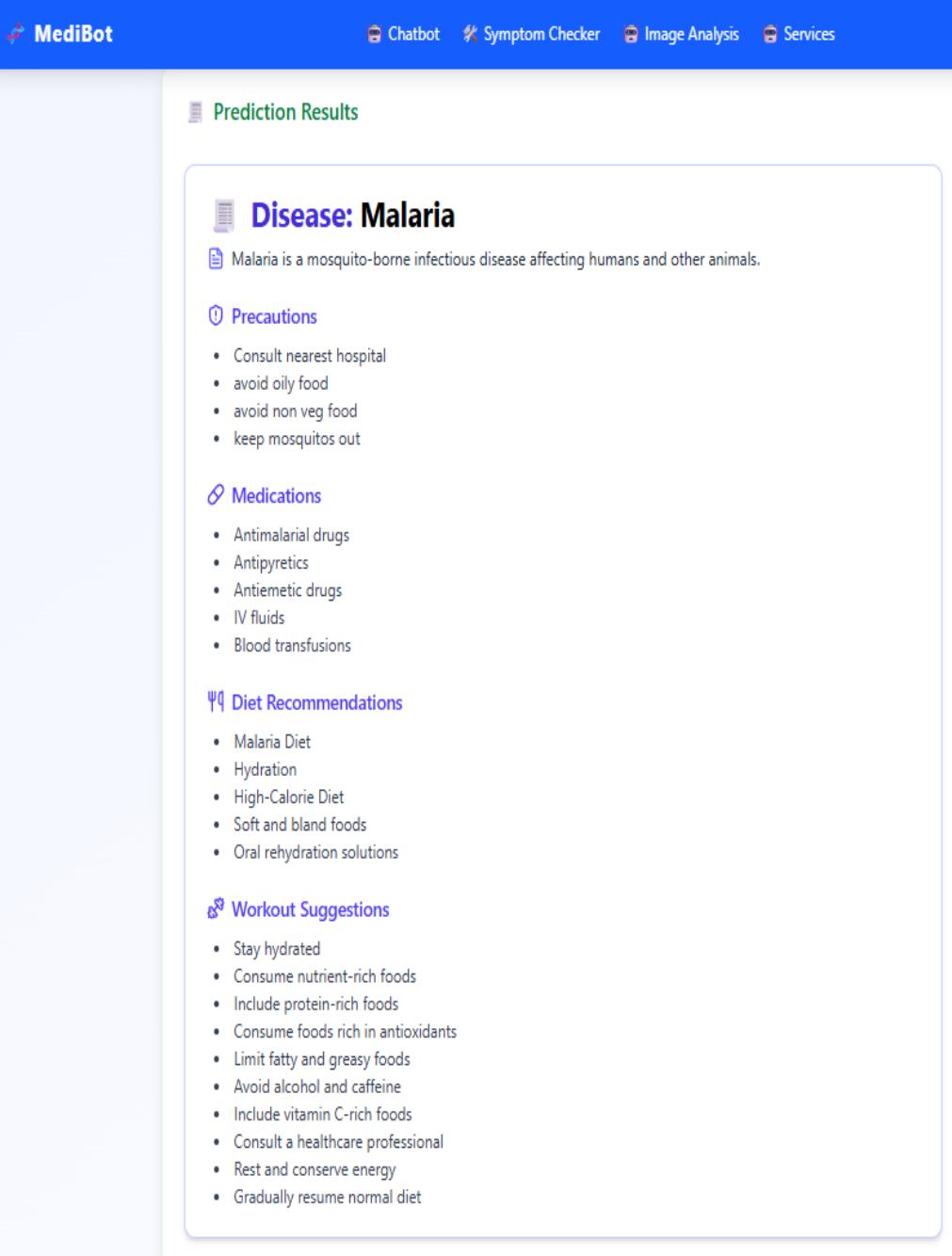
#### Home Page



#### Symptom Prediction Page



# Symptom Prediction Result



The screenshot shows a mobile application interface for MediBot. At the top, there is a blue header bar with the MediBot logo on the left and navigation links for Chatbot, Symptom Checker, Image Analysis, and Services on the right. A user profile icon is also present in the top right corner. Below the header, the main content area has a light gray background. It displays a section titled "Prediction Results" with a green icon and the text "Disease: Malaria". Under this, there is a paragraph of text: "Malaria is a mosquito-borne infectious disease affecting humans and other animals." Following this, there are several sections with icons and lists of recommendations:

- Precautions**: Includes a shield icon and a list: Consult nearest hospital, avoid oily food, avoid non veg food, keep mosquitos out.
- Medications**: Includes a syringe icon and a list: Antimalarial drugs, Antipyretics, Antiemetic drugs, IV fluids, Blood transfusions.
- Diet Recommendations**: Includes a fork and knife icon and a list: Malaria Diet, Hydration, High-Calorie Diet, Soft and bland foods, Oral rehydration solutions.
- Workout Suggestions**: Includes a person running icon and a list: Stay hydrated, Consume nutrient-rich foods, Include protein-rich foods, Consume foods rich in antioxidants, Limit fatty and greasy foods, Avoid alcohol and caffeine, Include vitamin C-rich foods, Consult a healthcare professional, Rest and conserve energy, Gradually resume normal diet.

## Medical Chatbot Page

The screenshot shows the MediAI Care Assistant interface. At the top, there's a blue header bar with the MediBot logo and navigation links for Chatbot, Symptom Checker, Image Analysis, and Services. A user profile icon is on the right. Below the header is a sidebar titled "Sessions" showing two previous sessions: one from 15/05/2025 at 10:18:50 and another from 15/05/2025 at 09:19:22. A green button labeled "+ New Session" is at the bottom of the sidebar. The main area is titled "Chat with MediAI Care". It starts with a message from the bot: "Hello! How can I assist you today?". The user responds with "I've had a fever, cold, and cough for the past two days.". The bot replies with "It's important to rest and stay hydrated. Consider taking over-the-counter medications like decongestants or fever reducers. If your symptoms worsen or last more than a few days, consult a doctor." The user then asks "How can I prevent fever, cold, and cough?". The bot responds with "To prevent fever, cold, and cough, maintain good hygiene by washing hands frequently and using hand sanitizers. Avoid close contact with sick individuals to reduce exposure to pathogens. Strengthen your". At the bottom of the chat window are input fields for "Ask a medical question..." and a "Send" button.

## Image based Diagnosis System Page

The screenshot shows the DiagnoVision interface. At the top, there's a blue header bar with the MediBot logo and navigation links for Chatbot, Symptom Checker, Image Analysis, and Services. A user profile icon is on the right. The main area is titled "DiagnoVision". It has a "Upload Image" section with a file input field containing "Choose file test3.png". Below it is a "Preview:" section showing a thumbnail of a man with his hand on his head. There's also an "Enter Query" section with the text "i don't know , what is it on my head, i've been facing this issues like this past few w". At the bottom is a blue "Processing..." button.

## Image based Diagnosis System Results

Chatbot Symptom Checker Image Analysis Services



Enter Query

i don't know , what is it on my head, i've been facing this issues like this past few w

Submit

Response

The image you're describing appears to show a man scratching his scalp, which is visibly flaking. There are several conditions that could be causing these symptoms, and it's best to consult a healthcare professional for a proper diagnosis. Here are a few possibilities:

**Dandruff:** This is a common condition characterized by flakes of dead skin on the scalp, often accompanied by itchiness. It's usually not serious and can be treated with over-the-counter shampoos.

**Scalp Psoriasis:** This condition causes red, itchy, scaly patches, which can appear on the scalp. It can be more challenging to treat than dandruff and may require prescription medications.

**Seborrheic Dermatitis:** This is a skin condition that can cause scaly patches, red skin, and stubborn dandruff. It often affects oily areas of the body, like the scalp.

**Dry Scalp:** Sometimes, the scalp can become dry, leading to flaking and itchiness. This can be due to various factors, including harsh shampoos, cold weather, or dry air.

**Head Lice or Other Infestations:** Though less likely given the description, infestations like head lice can cause itching and discomfort.

To address your concern, consider the following steps:

**Consult a Healthcare Professional:** A dermatologist can provide a diagnosis and recommend appropriate treatment.

**Use Gentle Hair Care Products:** Avoid harsh shampoos and conditioners.

**Maintain Good Scalp Hygiene:** Regular washing can help manage some conditions, but be cautious not to overdo it, as this can strip the scalp of its natural oils.

**Try Anti-Dandruff Shampoos:** If the issue is dandruff or a similar condition, these shampoos can be helpful.

Without a professional diagnosis, it's challenging to provide a specific treatment plan. If you're concerned about the flakes on your scalp, scheduling an appointment with a dermatologist is a good step towards finding a solution.

# User Dashboard Page

The screenshot shows the MediBot User Dashboard Page. At the top, there is a blue header bar with the MediBot logo on the left and navigation links for Chatbot, Symptom Checker, Image Analysis, and Services. On the right side of the header is a user profile icon. Below the header, there is a large central area featuring a circular profile picture of a person with a beard, followed by the greeting "Hey, Nithish S!". Below the greeting, a message says "Here's your dashboard with all your health tools' history:". There are four cards in this section showing basic user information: Email (nithish@gmail.com), Age (19), Phone (994691651), and Location (Arni, Tamil Nadu). The main content area is divided into two sections: "Symptom Checker History" on the left and "Image Analysis History" on the right. Each section contains a list of historical entries, each with a timestamp, disease/symptom information, and a "View Result" link.

## Symptom Checker History

- ⌚ 18/05/2025, 10:32:18  
**Disease:** Malaria  
**Symptoms:** chills, high\_fever, sweating, headache  
[View More →](#)
- ⌚ 15/05/2025, 09:21:57  
**Disease:** Hepatitis D  
**Symptoms:** joint\_pain, vomiting, yellowish\_skin, dark\_urine  
[View More →](#)
- ⌚ 14/05/2025, 14:32:51  
**Disease:** Urinary tract infection  
**Symptoms:** fever, cold, cough  
[View More →](#)
- ⌚ 13/05/2025, 21:06:11

## Image Analysis History

- ⌚ 18/05/2025, 10:39:03  
**Query:** i don't know , what is it on my head, i've been facing this issues like this past few weeks  
[View Result →](#)
- ⌚ 14/05/2025, 15:46:21  
**Query:** what is this ?  
[View Result →](#)
- ⌚ 14/05/2025, 15:45:18  
**Query:** i don't know , what is it on my head, i've been facing this issues like this past few weeks  
[View Result →](#)
- ⌚ 14/05/2025, 15:44:02

## APPENDIX C

---

### User Guide (How to use chatbot & image tool)

#### How to Use the Symptom-Based Chatbot

1. Go to the **chatbot page** on the web app.
  2. Enter symptoms in plain English (e.g., “*fever, cough, fatigue*”).
  3. Click “Analyze” – The chatbot responds with a probable disease and reasoning.
  4. Optionally ask for a **diet plan, prescription, or home remedy**.
- 

#### How to Use the Image-Based Diagnosis Tool

1. Navigate to the **image tool tab**.
  2. Upload an image of a visible symptom (skin rash, swelling, wound).
  3. The tool sends it to GROQ + LLaMA for analysis.
  4. Output shows a medical explanation with possible conditions.
-

## REFERENCES

---

### Core Disease Diagnosis & AI Foundations

1. **Scully, J. L. (2004).** *What is a disease?* EMBO Reports, **5**(7), 650–653.  
<https://doi.org/10.1038/sj.embor.7400194>  
► Groundwork for disease definition and diagnosis theory.
  2. **Barabási, A.-L., Gulbahce, N., & Loscalzo, J. (2011).** *Network medicine: a network-based approach to human disease.* Nature Reviews Genetics, **12**(1), 56–68.  
<https://doi.org/10.1038/nrg2918>  
► Disease interconnectivity & system biology.
- 

### AI & Machine Learning in Medical Diagnosis

3. **Choi, E., Bahadori, M. T., Schuetz, A., Stewart, W. F., & Sun, J. (2016).** *Doctor AI: Predicting clinical events via recurrent neural networks.* MLHC Conference.  
► Early RNN-based disease prediction work.
4. **Chen, M., Hao, Y., Hwang, K., Wang, L., & Wang, L. (2017).** *Disease prediction by machine learning over big data from healthcare communities.* IEEE Access, **5**, 8869–8879.  
<https://doi.org/10.1109/ACCESS.2017.2694446>  
► ML applications in health prediction systems.
5. **Betancur, J., et al. (2018).** *Deep learning for prediction of obstructive disease from fast myocardial perfusion SPECT: a multicenter study.* JACC: Cardiovascular Imaging, **11**(11), 1654–1663.

<https://doi.org/10.1016/j.jcmg.2018.02.020>

- Deep learning in real-world diagnostic use.
- 

## Fuzzy Logic & Expert Systems in Medicine

6. **Zadeh, L. A. (1965).** *Fuzzy sets*. Information and Control, **8(3)**, 338–353.  
[https://doi.org/10.1016/S0019-9958\(65\)90241-X](https://doi.org/10.1016/S0019-9958(65)90241-X)
    - The origin of fuzzy logic — must cite.
  7. **Phuong, N. H., & Kreinovich, V. (2001).** *Fuzzy logic and its applications in medicine*. International Journal of Medical Informatics, **62(2-3)**, 165–173.
    - Summary of fuzzy logic in healthcare.
  8. **Chen, H. L., et al. (2013).** *An efficient diagnosis system for detection of Parkinson's disease using fuzzy K-nearest neighbor approach*. Expert Systems with Applications, **40(1)**, 263–271.
    - Real application of fuzzy-based diagnosis.
- 



## RAG, NLP, and Chatbot Systems in Healthcare

9. **Lewis, P., et al. (2020).** *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. NeurIPS.
    - The RAG architecture your system is built on.
  10. **Chung, H. W., et al. (2022).** *Scaling instruction-finetuned language models with FLAN*. Google Research.
    - NLP foundation models powering chatbots.
-

## Medical Chatbots & Health Informatics

11. **Bickmore, T. W., & Giorgino, T. (2006).** *Health dialog systems for patients and consumers.* Journal of Biomedical Informatics, **39(5)**, 556–571.
  - Historical base of health chatbots.
12. **Miner, A. S., et al. (2016).** *Smartphone-based conversational agents and responses to questions about mental health, interpersonal violence, and physical health.* JAMA Internal Medicine, **176(5)**, 619–625.  
<https://doi.org/10.1001/jamainternmed.2016.0400>
  - Evaluates medical chatbot effectiveness.