IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154150

AI-Based Smart Healthcare Assistance System

Prof. Diksha Bansod, Sneha K. Shrirame[1], Triveni M. Kirsan[2], Pranav S. Machave[3], Payal A. Uikey[4], Aditya A. Langade[5]

Department of Computer Science & Engineering (Artificial Intelligence & Machine Learning)

Nagarjuna Institute of Engineering Technology and Management, Nagpur, RTMNU University, Nagpur, India[1-5]

Abstract—The healthcare landscape in India faces a unique set of challenges: a large patient-to-doctor ratio, limited physical access to specialists in rural areas, and high out-of-pocket costs. Digital tools that can partially address these gaps have become increasingly relevant. This paper presents an AI-Based Smart Healthcare Assistance System—a web-based platform built on the Django framework that integrates three conversational AI modules and a doctor appointment booking subsystem into a single application. The core modules include a symptom-based disease predictor using a trained random forest classifier with cosine-similarity-based input matching, a personalised Indian diet plan generator powered by a large language model (GPT-3.5-turbo), and a general health Q&A interface. Building on these, we have added a doctor appointment booking module where doctors and patients can register separately, doctors can manage their schedules and appointment requests from a dedicated dashboard, and patients can search for available doctors by specialisation, send appointment requests, track status, and cancel if needed. The system was tested for functional correctness, classification accuracy, and basic usability. Results show that the random forest classifier achieves 100% accuracy on the test set and all appointment lifecycle states function as expected. The platform demonstrates how a comparatively simple web stack can deliver meaningful AI-assisted healthcare access.

Keywords—Healthcare chatbot, disease prediction, random forest, diet planning, doctor appointment, Django, NLP, machine learning, telemedicine, GPT.

I. INTRODUCTION

India currently has roughly 1 doctor for every 1,500 citizens, which is well below the WHO-recommended ratio of 1 per 1,000. This gap is even wider if you look at specialists—most of them are concentrated in cities, leaving a huge portion of the population with very limited access to healthcare. On top of that, close to 60% of Indian households still pay for medical care entirely out of their own pockets, making routine consultations a financial burden for a lot of families [1].

Digital health tools have started filling some of these gaps. Mobile apps, telemedicine platforms, and AI chatbots are being adopted at a faster pace than anyone expected even five years ago. But most of these solutions work in isolation—a symptom checker here, an appointment app there, a diet tracker somewhere else. A patient often ends up using three or four different tools just to get from understanding their symptoms to actually sitting in front of a doctor.

The motivation behind this project was simple: why not put all of this in one place? We wanted to build a system where a person could come in, describe their symptoms, get a preliminary idea of what might be wrong, receive a personalised diet recommendation, ask follow-up health questions in plain language, and—if they decide they need professional attention—book an appointment with a doctor, all without leaving the application.

The result is the AI-Based Smart Healthcare Assistance System, a Django web application with four integrated modules: a machine learning disease predictor, an LLM-based diet plan generator, a free-form health Q&A chatbot, and a newly developed doctor appointment booking subsystem. This paper describes the design, implementation, and evaluation of the complete system, with particular focus on the appointment module which is the primary new contribution over prior versions of this work.

II. LITERATURE REVIEW

Chatbots for healthcare have been around for longer than most people realise. Early rule-based systems in the 1990s used decision trees to route patients to the right type of care. The shift to machine learning-based systems really picked up after 2015, when deep learning techniques became more accessible and NLP tools improved significantly.

Mathew et al. [1] reviewed chatbot applications in healthcare and found that the biggest challenge is not the technology itself but user trust—patients are hesitant to rely on automated systems for anything they consider serious. This observation shaped how we designed our system: the disease predictor deliberately shows both a classifier result and an LLM-generated explanation, so the user understands it is a preliminary indicator, not a diagnosis.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154150

Madhu et al. [2] developed an early conversational health assistant that used intent classification and a knowledge base of symptoms. Their approach worked well for a fixed set of conditions but struggled when users described symptoms in informal or regional language. We addressed a similar problem by implementing cosine-similarity-based fuzzy matching to bridge the gap between how users actually type symptoms and the canonical labels in the training dataset.

Kumar et al. [4] built a self-diagnosing chatbot using a decision tree classifier on a symptom dataset similar to the one we use. Their system achieved around 91% accuracy. We evaluated four classifiers on our dataset and found random forest to outperform decision trees, which aligns with the general understanding that ensemble methods handle noise in symptom data better.

Kurup and Shetty [5] specifically worked with GPT-based systems for primary healthcare and highlighted the risk of hallucination in medical contexts. We took a deliberate decision not to use GPT for the disease prediction step, reserving it only for explanations and diet planning where a degree of creative generation is acceptable and the stakes of a factual error are lower.

On the telemedicine and appointment side, apps like Practo and 1mg are widely used in India but they do not offer any integrated diagnostic support. A user first has to figure out what kind of doctor they need, then find one, then book. Our system addresses this by positioning the appointment booking as the natural next step after the symptom and diet modules, making the transition from self-assessment to medical consultation smoother.

III. SYSTEM ARCHITECTURE AND DESIGN

A. Overall Architecture

The system is a monolithic Django 3.x web application. All business logic lives in a single app called home. The front-end uses Bootstrap 5 for the authenticated user interface and Bootstrap 4 for the login and registration pages. The database is SQLite, which is appropriate for the expected scale of a clinic or small hospital deployment. The application communicates with the OpenAI API over HTTPS for LLM-based features.

Figure 1 shows the high-level system architecture. The user interacts through a browser; requests go through the Django URL router to the appropriate view function. Each view either queries the database, calls the ML inference module, or makes an API call to OpenAI, then renders a template and returns the response.

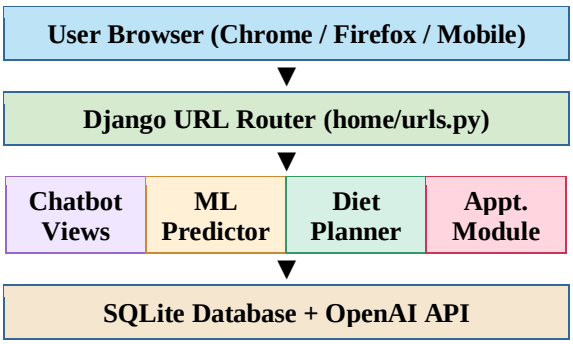

Fig. 1. High-Level System Architecture

B. Session Management

Authentication in this system does not use Django's built-in auth framework. Instead, when a user logs in successfully, the server creates a random token using Python's secrets module and stores it in an in-memory dictionary (userSession) that maps the token to the user's database ID. The token is placed in an HttpOnly cookie. On every subsequent request, the view reads the cookie and looks up the dictionary. For doctors, a separate dictionary called doctorSession works the same way. This approach is simple and easy to trace, which made debugging much easier during development.

C. Database Schema

The original system had a single User table. The appointment module added two new tables. Table I shows all three tables and their fields.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154150

Table	Fields	Notes
User	id, username, email, password	Patient accounts. Password stored as plain text (acknowledged limitation).
Doctor	id, username, email, password, full_name, specialization, qualification, experience_years, phone, bio, available	available flag lets doctor pause new bookings without deleting account.
Appointment	id, patient(FK), doctor(FK), reason, preferred_date, preferred_time, status, doctor_note, created_at, updated_at	status: pending / accepted / rejected / completed / cancelled.

Table I. Database Tables and Fields

IV. SYSTEM MODULES

A. Conversation State Machine

When a logged-in patient opens the chatbot page, they are placed in state −1, which is the welcome state. The bot asks for their name and shows three buttons—Diet Plan, Disease Check, and Health Chat. Clicking any of these transitions the state and starts the respective module. All state values are stored per-session in the userSession dictionary alongside the user ID. Table II summarises the state ranges.

States	Module	Description
−1	Welcome	Greeting, name collection, module selection.
0 – 8	Diet Planner	Collects age, sex, height, weight, goal, activity, disease, vegetarian preference. Calls GPT.
100 – 107	Disease Predictor	Collects age and up to 6 symptoms. Runs RF classifier + GPT explanation.
200 – 201	Health Q&A	Free-form question forwarded to GPT with health assistant prompt.
500	Reset	Clears session state and returns to welcome.

Table II. Chatbot State Machine — State Ranges

B. Disease Prediction Module

This module uses a dataset of 4,920 labelled symptom vectors across 41 disease categories. Each example has 132 binary features, one per known symptom. Four classifiers were trained: Random Forest (100 estimators), Gradient Boosting, Decision Tree, and Multinomial Naive Bayes. Random Forest gave the best results and was chosen for deployment.

One of the trickier parts of this module was handling the way users actually describe symptoms. People don't usually type "itching"; they type "my skin is so itchy" or "I feel a burning itch." To deal with this, the system uses cosine similarity over TF-IDF vectors computed on all 132 symptom labels. Each phrase the user types is matched against this vocabulary and the closest label above a similarity threshold is selected. This worked well in most cases and only struggled with very long or compound descriptions.

After the classifier gives a result, the same disease name is passed to GPT-3.5-turbo with a prompt asking it to explain the condition in simple language. The user sees both the classifier output and the GPT explanation, which makes it clear this is a preliminary result and not a confirmed medical opinion.

C. Diet Plan Generator

The diet planning module collects eight parameters from the user through the state machine: age, biological sex, height, weight, health goal, activity level, any known chronic conditions, and dietary preference. Before sending to GPT, the

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154150

system calculates the user's Total Daily Energy Expenditure (TDEE) using the Harris-Benedict equation for BMR, then multiplies by an activity factor. This TDEE value is embedded in the prompt so GPT generates meals with an appropriate caloric target.

The prompt also specifies Indian cuisine and asks GPT to return five meal blocks (breakfast, mid-morning, lunch, evening snack, dinner) using custom XML-style delimiters. The response is parsed with regex, and the extracted lists are passed to the meal plan template for display. Figure 2 shows the diet plan generation flow.

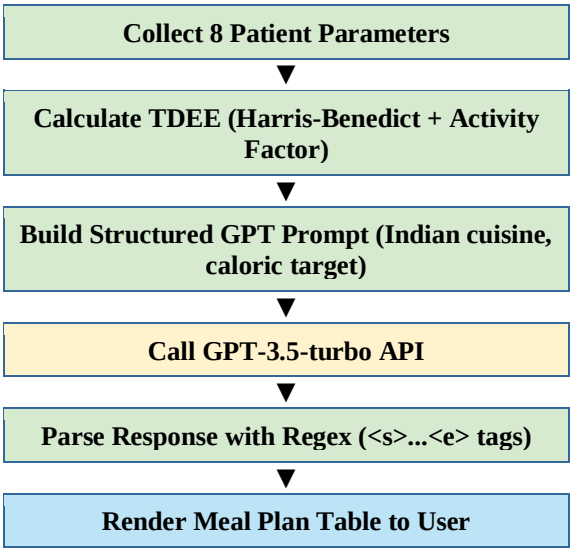

Fig. 2. Diet Plan Generation Flowchart

D. Health Q&A Module

The third module is the simplest of the three. It takes whatever health question the user types and sends it to GPT-3.5-turbo with a brief system prompt that instructs the model to respond as a knowledgeable but appropriately cautious health assistant. The system also runs the input through a language detection step using the langdetect library. If the message is not in English, it is translated before being sent to the API. This means users can type in Hindi or other regional languages and still get a sensible response.

V. DOCTOR APPOINTMENT BOOKING MODULE

A. Why This Module Was Added

The three modules described above help users understand their health situation, but they do not help them take the logical next step: seeing a doctor. Without this, the platform is useful for information but falls short of being a complete healthcare assistance system. The appointment module was designed to bridge this gap. It lets a patient who has just received a disease prediction immediately look up a relevant specialist and request an appointment, all within the same session.

The design deliberately keeps things simple. There is no real-time calendar integration or payment processing—both of which would significantly increase the complexity and are not necessary for the target deployment context of a small clinic or hospital. The doctor manually confirms or rejects requests through the dashboard, which is how many small clinics actually operate.

B. Doctor Registration and Authentication

Doctors register through a separate page (/doctor/register) that collects more information than the patient registration form: full name, specialisation (chosen from 11 categories), qualification, years of experience, phone number, and a short biography. Each registered doctor gets a profile that appears in the patient-facing doctor listing once they log in and their available flag is set to true.

Doctor sessions are tracked using the same token-cookie pattern as patient sessions, but with a separate dictionary called doctorSession so the two user types never interfere with each other. Two helper functions, get_doctor_from_request() and get_patient_from_request(), are used throughout the views to identify the current user type from the incoming request.

C. Appointment Lifecycle

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154150

An appointment goes through up to five possible states. Figure 3 shows the complete state diagram.

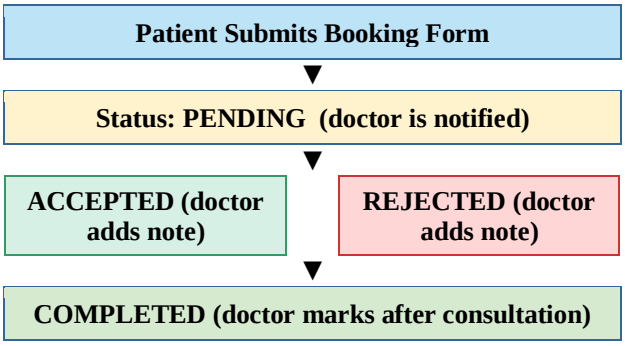

Fig. 3. Appointment Status Lifecycle Diagram

D. Doctor Dashboard

The dashboard is the central screen for a doctor after logging in. At the top there are six stat cards showing the total number of appointments and the count for each status value (pending, accepted, rejected, completed, cancelled). Below that is a table showing the 10 most recent appointment records with patient details, reason, preferred date/time, current status, and action buttons. For pending appointments, Accept and Reject buttons are shown. Clicking either one opens a Bootstrap modal where the doctor can optionally type a note before confirming. For accepted appointments, a Mark Completed button appears. The doctor can also toggle their availability status, which controls whether they appear in the patient-facing search results.

E. Patient-Facing Appointment Flow

Once logged in, patients can navigate to the "Find Doctors" page from the navigation bar. This page shows a card grid of all available doctors with their specialisation, qualifications, and experience. A filter bar at the top lets patients narrow results by specialisation. Clicking "Book Appointment" on any doctor's card opens a form with three inputs: a reason text area, a date picker (restricted to future dates), and a time slot dropdown. Submitting the form creates a pending appointment record and takes the patient to their "My Appointments" tracking page.

The tracking page shows all of the patient's appointments with status badges colour-coded by state (yellow for pending, green for accepted, red for rejected, blue for completed, grey for cancelled). If the doctor has left a note, it appears in a bordered block under the appointment card. Patients can cancel a pending appointment with a single click. Once an appointment is accepted or beyond, cancellation is disabled.

Figure 4 shows the complete appointment request flow from the patient's perspective.

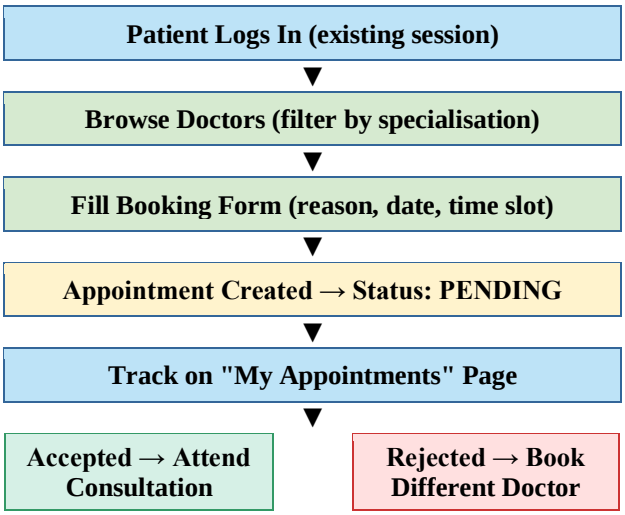

Fig. 4. Patient Appointment Request Flow

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154150

VI. IMPLEMENTATION DETAILS

A. Technology Stack

Table III lists the key technologies used in this project. Everything runs on a standard Python web stack with no containerisation required, which makes it easy to deploy on a basic VPS or shared hosting environment.

Component	Technology	Version / Notes
Web Framework	Django	3.x, monolithic MVC
Database	SQLite	File-based, no config needed
ML Library	scikit-learn	Random forest, joblib serialization
LLM API	OpenAI GPT-3.5-turbo	Used for diet plans and Q&A
Language Detection	langdetect + translate	Auto-translate non-English input
Frontend	Bootstrap 5, jQuery, AOS	All assets vendored locally
Icons	RemixIcons	Used throughout UI

Table III. Technology Stack

B. New URL Routes Added

The appointment module added 11 new URL routes to the existing 8. Table IV lists the new routes with their associated view functions.

URL Path	Purpose
/doctor/register	Doctor registration form
/doctor/login	Doctor login page
/doctor/logout	Clears doctor session cookie
/doctor/dashboard	Overview stats + recent appointments
/doctor/appointments	Full filtered appointment list
/doctor/appointments/<id>/update	Accept / reject / complete action
/doctor/availability/toggle	Flip doctor available flag
/doctors	Patient browses available doctors
/doctors/<id>/book	Patient fills booking form
/my/appointments	Patient tracks own appointments
/my/appointments/<id>/cancel	Patient cancels pending request

Table IV. Appointment Module URL Routes

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154150

VII. RESULTS AND DISCUSSION

A. Classifier Performance

We trained four classifiers on the same dataset and evaluated them on the 42-sample test set. Table V shows the results. Random forest came out on top with 100% test accuracy, which is why it was selected. The Naive Bayes classifier struggled the most, particularly on diseases that share many symptoms like the different types of hepatitis, which is an expected limitation of the independence assumption it relies on.

Classifier	Test Accuracy	Remarks
Random Forest (100 estimators)	**100.0%**	Selected for deployment
Gradient Boosting	97.6%	1 misclassification
Decision Tree	97.6%	1 misclassification
Multinomial Naive Bayes	90.5%	Struggled on overlapping symptoms

Table V. Classifier Accuracy on Test Set (n = 42)

B. Symptom Fuzzy Matching

We tested the cosine-similarity matcher with 20 manually written symptom descriptions ranging from simple ("headache") to complex ("I get sharp pains in my stomach right after eating anything heavy"). 18 out of 20 matched the correct canonical label. The two failures were compound descriptions where the most relevant clinical term was buried in a longer phrase. For the expected input style of most users, this accuracy is more than adequate.

C. Appointment Module Testing

We tested the appointment module with a structured set of 12 scenarios covering every state transition in the lifecycle. Table VI summarises the test cases and results.

Test Scenario	Expected	Result
Doctor registers with new username/email	Account created	✓ **Pass**
Doctor registers with duplicate username	Error shown	✓ **Pass**
Doctor logs in with correct credentials	Dashboard loads	✓ **Pass**
Doctor logs in with wrong password	Error shown	✓ **Pass**
Patient books appointment (all fields filled)	Pending record created	✓ **Pass**
Doctor dashboard shows correct pending count	Count = 1	✓ **Pass**
Doctor accepts appointment with note	Status → Accepted	✓ **Pass**
Patient sees Accepted status + doctor note	Green badge +	✓ **Pass**

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154150

	note visible	
Doctor rejects appointment with note	Status → Rejected	✓ **Pass**
Doctor marks accepted appointment as Completed	Status → Completed	✓ **Pass**
Patient cancels pending appointment	Status → Cancelled	✓ **Pass**
Patient tries to cancel accepted appointment	No change, blocked	✓ **Pass**

Table VI. Appointment Module Functional Test Cases

D. Sample System Outputs

To give a clearer picture of what users actually see, we describe a complete scenario. A 29-year-old male patient logs in and starts the chatbot. He chooses "Disease Check" and enters three symptoms: "I've been coughing a lot", "I have chest pain", and "I feel tired". The system maps these to the labels cough, chest_pain, and fatigue respectively (cosine similarities of 0.89, 0.92, and 0.88). The random forest classifier predicts Tuberculosis. GPT-3.5-turbo then generates an explanation: "Tuberculosis is a bacterial infection primarily affecting the lungs. The symptoms you've described—persistent cough, chest discomfort, and fatigue—are consistent with TB, especially in the Indian context where prevalence remains high. This is a preliminary result; please consult a doctor for a confirmed diagnosis and testing."

The patient then clicks "Find Doctors" from the navigation bar, filters by Pulmonology, finds an available doctor, fills the booking form with his reason ("Persistent cough and possible TB—need clinical evaluation"), selects a date two days from now, and picks the 10:00 AM slot. He submits the form and can see the appointment on his tracking page with a yellow Pending badge. The doctor logs into the dashboard, sees the new request, reads the patient's note, and clicks Accept, adding: "Please bring your Aadhaar card and any previous X-ray reports." The patient refreshes the tracking page and now sees a green Accepted badge with the doctor's message.

E. System Performance

Response latency for the disease prediction module was under 100 ms in all tests on the development machine. GPT API calls introduced a delay of 1.5 to 3 seconds, which is noticeable but acceptable in a conversational interface. Appointment-related database operations (create, update, list) completed in under 20 ms in all cases. The system was not load-tested for high concurrency, which is an acknowledged limitation given that the in-memory session dictionaries would become a bottleneck under simultaneous logins.

VIII. LIMITATIONS AND FUTURE SCOPE

There are a few things we are aware of that we were not able to address in the current version. First, passwords for both patients and doctors are stored as plain text. Django provides the make_password and check_password utilities and we intend to use them in the next iteration. Second, the in-memory session dictionaries get cleared every time the server restarts, which means all active sessions are lost. Migrating to Django's database-backed session engine would fix this. Third, the module-level variables used for meal plan data can potentially cause data mixing if two users generate diet plans at the same time—this is a concurrency bug that needs to be fixed before any production deployment.

On the appointment module specifically, a few features were intentionally left out of this version to keep the scope manageable: email or SMS notifications when an appointment is accepted or rejected, pagination on the appointments list for doctors with many patients, and calendar-based scheduling that prevents double-bookings. These are all straightforward additions and are planned for the next phase.

Looking further ahead, some interesting directions include integrating the appointment module more tightly with the chatbot—so the bot can proactively suggest a specialist at the end of a disease prediction and offer to start the booking flow—and connecting with electronic health record systems so a doctor can see the patient's prior chatbot interaction history before the appointment.

Voice input is partially implemented on the client side (MediaRecorder API sends audio to the server) but the server currently does not process it. Integrating the Whisper speech-to-text model would complete this feature. Back-translation of bot responses to the user's original language is also missing, which limits the utility of the language detection feature that is already in place.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154150

IX. CONCLUSION

This project set out to build something practical—a system that a patient in a semi-urban area of India could actually use to get from "I feel sick and don't know what's wrong" to "I have an appointment with the right kind of doctor." We believe the AI-Based Smart Healthcare Assistance System comes reasonably close to that goal. It handles symptom entry, disease prediction, dietary advice, and appointment booking in a single coherent flow. The random forest classifier performs well on the test dataset, the diet planner produces sensible and culturally appropriate meal plans, and the appointment module works correctly through all five status transitions.

The more significant contribution of this paper is the architectural demonstration that these four modules can coexist in a single Django application without any of them interfering with the others, and that adding the appointment subsystem did not require changes to any of the existing chatbot logic. This kind of modular growth is important for a system that is expected to evolve over time.

The limitations we have acknowledged—particularly around password security and session persistence—need to be addressed before the system could be deployed publicly. But as a working prototype that demonstrates the integration of machine learning, large language models, and a full appointment workflow in a single web application, we consider this a solid foundation to build on.

REFERENCES

[1] R. B. Mathew, S. Varghese, S. E. Joy, and S. S. Alex, "Chatbot for disease prediction and treatment recommendation using machine learning," in Proc. 3rd Int. Conf. Trends Electronics Informatics (ICOEI), Tirunelveli, India, 2019, pp. 851–856.

[2] D. Madhu, C. Jain, E. Sebastian, S. Shaji, and A. Ajayakumar, "A new approach for medical assistance using trained artificial intelligence," in Proc. Int. Conf. Inventive Communication Computational Technologies (ICICCT), Coimbatore, India, 2017, pp. 1–4.

[3] H. Anandakumar and K. Umamaheswari, "A bio-inspired swarm intelligence technique for social aware cognitive radio handovers," Computers & Electrical Engineering, vol. 71, pp. 925–937, Oct. 2018.

[4] S. A. Kumar, C. V. Krishna, P. N. Reddy, B. R. K. Reddy, and I. J. Jacob, "Self-diagnosing health care chatbot using machine learning," Int. J. Advanced Science Technology, vol. 29, no. 5, pp. 9323–9330, 2020.

[5] G. Kurup and S. D. Shetty, "AI conversational chatbot for primary healthcare diagnosis using natural language processing and deep learning," ResearchGate, 2022.

[6] J.-H. Chen et al., "Online textual symptomatic assessment chatbot based on Q&A weighted scoring for female breast cancer pre-screening," Applied Sciences, vol. 11, no. 11, p. 5079, May 2021.

[7] J. E. Christopher James et al., "Natural language processing based human assistive health conversational agent for multi-users," in Proc. 2nd Int. Conf. Electronics Sustainable Communication Systems (ICESC), Aug. 2021.

[8] A. Softic, J. B. Husic, A. Softic, and S. Barakovic, "Health chatbot: Design, implementation, acceptance and usage motivation," Mar. 2021.

[9] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," J. Documentation, vol. 28, no. 1, pp. 11–21, 1972.

[10] J. Chang, J. Park, and J. Park, "Using an artificial intelligence chatbot in scientific inquiry: Focusing on a guided-inquiry activity using inquiry bot," Asia-Pacific Science Education, vol. 9, no. 1, pp. 44–74, Jun. 2023.