AyurWell

Multimodal Conversational AI Chatbot for Ayurvedic Health Assistance

Complete Project Documentation

Project Name:	AyurWell - AI Health Assistant
Domain:	AI in Healthcare / Ayurveda
Technology Stack:	Python, Flask, LangChain, LangGraph, Pinecone
AI Models:	Google Gemini 2.0 Flash, LLaMA-4 Scout
Generated:	November 17, 2025

Developed by: Santosh, Viresh, and Vishwas

■ Table of Contents

1. ■ Project Overview

AyurWell is an advanced multimodal conversational AI chatbot designed to provide comprehensive health assistance through an integrated Agentic RAG (Retrieval-Augmented Generation) system. The system combines multiple AI techniques including Natural Language Processing, vector embeddings, and multi-agent orchestration to deliver accurate, contextual health information with a focus on Ayurvedic medicine and holistic wellness.

This project represents a production-level AI application that demonstrates how modern AI assistants are built for real-world healthcare applications. It showcases the integration of multiple AI services, intelligent routing, and failover mechanisms to ensure reliable responses.

2. ■ What Does This Project Do?

Capability	Description
Answer Health Questions	Responds to queries about symptoms, diseases, treatments, and Ayurvedic remedies
Image Understanding	Analyzes medical images, reports, rashes, and medicine packages using vision AI
Voice Input	Supports speech recognition for hands-free interaction
Text-to-Speech	Speaks answers in high-quality voice (English and Kannada)
Web Search Fallback	Searches the internet when local database doesn't have answers
Multilingual Support	Works in English and Kannada with automatic translation
Conversational Memory	Remembers chat history for context-aware responses
Smart Routing	Intelligently routes queries to appropriate handlers

3. ■■ System Architecture

The system follows a three-tier architecture:

3.1 Frontend Layer (User Interface)

Files: templates/ui.html, static/style.css
Technology: HTML5, CSS3, JavaScript (Vanilla)
Features:
• Modern, responsive web interface with dark mode support
• Real-time chat interface with message history
• Image upload functionality with preview
• Voice input using Web Speech API
• Audio playback for TTS responses
• Language switcher (English/Kannada)
• Material Design icons for better UX

3.2 Backend Layer (Application Server)

File: app.py
Technology: Flask (Python web framework)
Responsibilities:
• HTTP request handling and routing
• Session management and CORS handling
• Image upload processing and storage
• Integration with AI workflow orchestrator
• Translation services (Kannada ↔ English)
• Text-to-speech generation
• Error handling and logging

Key Endpoints:
• GET / - Serves the main UI
• POST /chat - Handles chat messages and images
• POST /tts - Generates speech audio
• GET /health - Health check endpoint

3.3 Intelligence Layer (AI Agents)

Files: workflow/graph.py, Agents/* folder
Technology: LangGraph, LangChain, Google Gemini
Components:
• Multi-agent orchestration system
• Vector database retrieval (Pinecone)
• LLM-based reasoning and generation
• Document grading and relevance scoring
• Web search integration (Tavily)
• Query enhancement and classification

4. ■ Multi-Agent Workflow

The heart of AyurWell is its sophisticated multi-agent system built with **LangGraph**. Instead of a single monolithic AI, the system uses specialized agents that work together like a team of medical specialists, each handling specific tasks.

4.1 Agent Flow Diagram

```
User Question
↓
1. Query Enhancer - Cleans and reformulates the question
↓
2. Query Classifier - Categorizes the query type
↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
↓ ↓ ↓ ↓
Health Query Greeting Off-Topic
↓ ↓ ↓
3. Retrieve Greeting Resp. Polite Reject
↓ ↓ ↓
4. Grade Docs → Generate Answer ←
↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
↓ ↓ ↓ ↓
Good Docs No Docs Tried 2x
↓ ↓ ↓
Generate Refine Q. Web Search
↓ ↓
→ Retrieve ←
```

4.2 Individual Agent Descriptions

Agent	File	Purpose	Example
Query Enhancer	query_processing.py	Reformulates questions for better retrieval	"My head hurts" → "What are causes and treatments for hea
Query Classifier	query_processing.py	Routes to health/greeting/off-topic handlers	"Hello" → greeting, "diabetes symptoms" → health
Retrieval Agent	retrieval.py	Searches Pinecone vector database for relevant documents	Retrieves top 5 similar chunks from Ayurvedic PDFs
Retrieval Grader	retrieval.py	Scores document relevance using LLM	Filters out off-topic documents, keeps only relevant ones
Query Refiner	query_processing.py	Rewrites query if no good documents found	Attempts up to 2 times before web fallback
Web Search Agent	retrieval.py	Searches internet using Tavily API	Used when local database has no relevant information
Response Generator	response_generation.py	Creates final answer using RAG	Combines context + history + query → coherent answer
Greeting Handler	response_generation.py	Responds to casual interactions	"Hello" → "Namaste! How can I help you today?"
Off-topic Handler	response_generation.py	Politely rejects non-health queries	"Weather today?" → "I specialize in health topics"

5. ■■ The Knowledge Base

The system's knowledge comes from curated Ayurvedic and medical texts stored in a **vector database**. This enables semantic search - finding information based on meaning rather than just keyword matching.

5.1 Vector Database - Pinecone

Technology: Pinecone (Managed vector database)
Setup File: Pinecone_load.py
Embedding Model: sentence-transformers/all-MiniLM-L6-v2 (384 dimensions)
Source Documents: Data/ folder containing Ayurvedic PDFs

How it Works:
1. **Document Loading:** PDF files are loaded from the Data/ directory
2. **Text Splitting:** Documents are chunked into 500-character segments with 20-char overlap
3. **Embedding Generation:** Each chunk is converted to a 384-dimensional vector
4. **Upload to Pinecone:** Vectors are stored with metadata in the cloud
5. **Query Processing:** User questions are embedded the same way
6. **Similarity Search:** Find top-k most similar vectors (cosine similarity)
7. **Return Documents:** Retrieve the original text chunks

5.2 Source Documents

The Data/ folder contains authoritative Ayurvedic texts:
• Evidence-based Ayurvedic Practice
• Ayurvedic Home Remedies (English)
• The Complete Book of Ayurvedic Home Remedies
• Charak Chikitsa Sthana (Classical text)
• Research papers from medical journals

These documents provide the foundation for accurate, evidence-based responses.

6. ■ AI Technologies Used

Technology	Model/Service	Purpose	File Reference
Large Language Model	Google Gemini 2.0 Flash	Question answering, classification, grading, query rewriting	chains/rag_chain.py
Embeddings	sentence-transformers/all-MiniLM-L6-v2	Convert text to 384-dim vectors for similarity search	Pinecone_upload.py, chains/rag_chain.py
Vision AI	LLaMA-4 Scout (via Groq)	Analyze medical images, describe visual findings	utils/image_desc.py
Text-to-Speech	Microsoft Edge TTS	High-quality voice synthesis (EN & KN)	edge_tts_helper.py
Translation	Google Translate API	Kannada ↔ English translation	app.py
Web Search	Tavily Search API	Real-time internet search fallback	Agents/retrieval.py
Agent Framework	LangGraph + LangChain	Multi-agent orchestration, state management, workflow	workflow/graph.py
Vector Database	Pinecone	Cloud-hosted similarity search	chains/rag_chain.py

7. ■ Key Files Explained

File Path	Lines	Purpose
app.py	583	Main Flask server - HTTP endpoints, request handling, TTS, translation
workflow/graph.py	42	LangGraph workflow definition - agent orchestration and routing
Agents/state.py	11	Shared state definition (TypedDict) passed between agents
Agents/routing.py	32	Conditional routing logic - decides next agent based on state
Agents/query_processing.py	144	Query enhancement, classification, and refinement
Agents/retrieval.py	110	Document retrieval, grading, and web search
Agents/response_generation.py	65	Final answer generation, greeting, off-topic handlers
chains/rag_chain.py	51	RAG setup - LLM, retriever, Tavily initialization
chains/prompt_templates.py	13	Prompt templates for LLM interactions
utils/image_desc.py	97	Image analysis using LLaMA vision model via Groq
edge_tts_helper.py	73	Text-to-speech helper using Microsoft Edge TTS
Pinecone_load.py	76	Vector database setup - loads PDFs into Pinecone
config/env.py	9	Environment variable loader
templates/ui.html	619	Frontend HTML - chat interface, voice input, image upload
static/style.css	~400	Styling for UI - dark mode, responsive design
requirements.txt	30	Python dependencies

8. ■ API Keys Required

The project requires API keys from several cloud services. Most offer generous free tiers suitable for development and testing.

Service	Purpose	Free Tier	How to Get
Pinecone	Vector database hosting	1 index, 100K vectors	pinecone.io → Sign up → Create API key
Google AI Studio	Gemini LLM access	60 requests/min	makersuite.google.com → API keys
Groq	LLaMA vision model	Generous limits	console.groq.com → Create API key
Tavily (Optional)	Web search	1000 searches/month	tavily.com → Sign up → API key

Environment Configuration (.env file):

Create a .env file in the project root with the following structure:

```
# API Keys
PINECONE_API_KEY=your_pinecone_api_key_here
GOOGLE_API_KEY=your_google_api_key_here
GROQ_API_KEY=your_groq_api_key_here
TAVILY_API_KEY=your_tavily_api_key_here

# Database Configuration
index_name=AyurWell

# Model Configuration
EMBEDDING_MODEL=sentence-transformers/all-MiniLM-L6-v2
model=gemini-2.0-flash-001
```

9. ■ How to Run the Project

Step 1: Clone or Navigate to Project

```
cd "D:\PES\sem-7\AI in HealthCare\Ayur"
```

Step 2: Create Virtual Environment (Recommended)

```
python -m venv .venv
.venv\Scripts\Activate.ps1 # On Windows PowerShell
# OR
source .venv/bin/activate # On Linux/Mac
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
```

This installs all required packages including Flask, LangChain, LangGraph, Pinecone, and AI model libraries. Installation may take 5-10 minutes depending on internet speed.

Step 4: Configure API Keys

Create a .env file in the project root directory and add your API keys (see Section 8).
Important: Never commit the .env file to version control (add to .gitignore).

Step 5: Initialize Vector Database (First Time Only)

```
python Pinecone_load.py
```

This script:
• Loads all PDF files from the Data/ directory
• Splits documents into chunks
• Generates embeddings using sentence-transformers
• Uploads vectors to Pinecone cloud

Time: 5-15 minutes depending on document size and internet speed.
Note: Only run this once. Re-running will duplicate data.

Step 6: Run the Application

```
python app.py
```

The Flask server will start on http://localhost:5000
You should see output like:

```
* Running on http://0.0.0.0:5000/
* Press CTRL+C to quit
Workflow built successfully
Pinecone retriever initialized with index: AyurWell
LLM initialized with model: gemini-2.0-flash-001
```

Step 7: Access the Application

Open your web browser and navigate to: **http://localhost:5000**

You should see the AyurWell chat interface. Try asking:
• "What are the symptoms of diabetes?"
• "How to improve digestion in Ayurveda?"
• Upload an image of a medical report
• Use voice input (click the microphone icon)

10. ■ Key Concepts for Beginners

10.1 RAG (Retrieval-Augmented Generation)

The Problem: Large Language Models (LLMs) like GPT or Gemini can "hallucinate" - they sometimes make up facts that sound convincing but are wrong.

The Solution: RAG combines retrieval and generation:
1. **Retrieve:** Search a knowledge base for relevant information
2. **Augment:** Add this information to the LLM's context
3. **Generate:** Let the LLM create an answer based on the retrieved facts

Benefit: Responses are grounded in actual documents, reducing hallucinations and ensuring accuracy. Perfect for healthcare where accuracy is critical.

10.2 Vector Embeddings

The Challenge: Computers don't understand meaning - they work with numbers.

The Solution: Vector embeddings convert text into arrays of numbers (vectors) that capture semantic meaning:
• "headache" → [0.23, -0.15, 0.87, ...] (384 numbers)
• "head pain" → [0.25, -0.14, 0.85, ...] (similar numbers!)

How it Works:
1. Pre-trained models learn to map words/sentences to vectors
2. Similar meanings = similar vectors (cosine similarity)
3. Search becomes a math problem: find nearest vectors

Advantage: Can find "diabetes treatment" even if document says "managing blood sugar" - semantic search instead of keyword matching.

10.3 Multi-Agent Systems

The Idea: Instead of one giant AI doing everything, break tasks into specialized agents.

Analogy: Like a hospital with specialists:
• Receptionist (Query Classifier) - routes patients
• Librarian (Retrieval Agent) - finds medical records
• Reviewer (Grader) - checks if records are relevant
• Doctor (Generator) - gives final diagnosis

Benefits:
• **Modularity:** Each agent is simple and testable
• **Debuggability:** Easy to see where things go wrong

• **Flexibility:** Can swap out individual agents
• **Scalability:** Add new agents without rewriting everything

10.4 LangGraph - Stateful Agent Orchestration

What is LangGraph? A framework for building complex AI workflows with state management.

Key Features:
• **State Management:** Shared data structure passed between agents
• **Conditional Routing:** "If documents are good, generate answer; else, refine query"
• **Cycles:** Can loop (e.g., retry retrieval with refined query)
• **Checkpointing:** Save/restore conversation state

In AyurWell:
• State contains: messages, documents, query, proceed flag, rephrase count
• Agents read state, do work, update state, pass to next agent
• Router functions decide which agent runs next based on state

10.5 Prompt Engineering

What is it? Crafting instructions to LLMs to get desired outputs.

Example from AyurWell (Query Classifier):

```
"You are a classifier that determines whether a user's question is about health topics.
Respond only with 'Yes' or 'No'. Be strict."
```

Best Practices:
• Be specific and clear
• Provide examples when possible
• Define output format explicitly
• Use system messages for role-setting

11. ■ Cool Features

Feature	Description	Technology
Multimodal Input	Accept text, voice, and images in a single query	Web Speech API, Flask file upload, LLaMA vision
Bilingual Support	Full English and Kannada interface with automatic translation	Google Translate API, i18n in frontend
Intelligent Fallback	Database → Web Search → Polite rejection (never fails)	Tavily API, conditional routing
Document Grading	LLM-based relevance scoring filters out wrong documents	Gemini structured output
Conversational Memory	Remembers chat history for context-aware responses	LangGraph state management
Dark Mode	Eye-friendly UI with toggle	CSS variables, JavaScript
Query Refinement	Automatically rewrites unclear questions (up to 2x)	Gemini LLM rephrasing
High-Quality TTS	Natural-sounding voices in multiple languages	Microsoft Edge TTS (neural voices)
Image Analysis	Describes medical images for context extraction	LLaMA-4 Scout via Groq
Responsive Design	Works on mobile, tablet, and desktop	CSS Grid, Flexbox, Media Queries

12. ■ Learning Path

To fully understand and contribute to this project, here's a recommended learning path:

Topic	Time	Resources	Why Important
Python Basics	1-2 weeks	Official tutorial, Codecademy	Project is entirely in Python
Flask Framework	3-5 days	Flask Mega-Tutorial	Backend web server
REST APIs	2-3 days	REST API tutorial	Communication between frontend/backend
HTML/CSS/JavaScript	1 week	MDN Web Docs	Frontend development
LLMs & Prompt Engineering	1 week	OpenAI cookbook, Prompt Engineering Guide	Core AI interaction
RAG Concepts	3-4 days	LangChain docs, RAG tutorials	Understanding retrieval-generation
LangChain Basics	1 week	LangChain documentation	Framework for LLM apps
LangGraph	1 week	LangGraph tutorials	Agent orchestration
Vector Databases	3-4 days	Pinecone docs, embedding tutorials	Semantic search
Embeddings & Similarity	2-3 days	Sentence Transformers docs	Text-to-vector conversion

Hands-On Practice:
1. Start with simple Flask apps
2. Build a basic chatbot with LangChain
3. Experiment with embeddings and similarity search
4. Create a simple RAG system
5. Try LangGraph tutorials
6. Finally, study this project's code

13. ■ Technical Implementation Details

13.1 Request Flow - Complete Journey

User asks: "What are symptoms of diabetes?"

1. Frontend (ui.html):
• User types question in input field
• Clicks send button
• JavaScript captures text and selected language
• Makes POST request to /chat endpoint

2. Backend - Flask (app.py):
• Receives POST request with message and lang parameters
• If image provided, saves to uploads/ and calls image_desc.py
• If lang=kn, translates Kannada → English using Google Translate
• Creates HumanMessage object with the query
• Calls chatbot.invoke() with question

3. LangGraph Workflow (graph.py):
• Starts at entry point: "query_enhancer"
• Passes through agent chain
• Returns final state with messages list

4. Agent Processing:
• **Query Enhancer:** Reformulates to "What are the symptoms and signs of diabetes mellitus?"
• **Query Classifier:** Determines it's a health query → routes to "retrieve"
• **Retrieval Agent:** Searches Pinecone, gets 5 document chunks
• **Retrieval Grader:** Scores each document, keeps 3 relevant ones
• **Response Generator:** Combines docs + query + history → answer via Gemini

5. Backend - Response Processing (app.py):
• Extracts final AIMessage from result["messages"][-1]
• If lang=kn, translates English → Kannada
• Returns JSON: {"reply": "diabetes symptoms are..."}

6. Frontend - Display:
• JavaScript receives response
• Creates bot message bubble with reply
• If TTS enabled, calls /tts endpoint
• Plays audio response

Total time: 2-5 seconds

13.2 State Management

AgentState (TypedDict):

```
{
"messages": List[BaseMessage], # Chat history
"documents": List[Document], # Retrieved docs
"on_topic": str, # "yes"/"no"/"greeting"
"enhanced_query": str, # Refined question
"proceed_to_generate": bool, # Has good docs?
"rephrase_count": int, # Retry counter
"question": HumanMessage # Current query
}
```

This state object is passed through all agents, allowing them to share information and make decisions based on previous agent outputs.

13.3 Error Handling & Fallbacks

The system has multiple layers of error handling:

1. LLM Failures: If Gemini API fails, return cached snippets from documents
2. Retrieval Failures: If Pinecone is down, skip to web search
3. Web Search Failures: If Tavily fails, return polite "try again" message
4. Translation Failures: If translation fails, return English response
5. TTS Failures: If Edge TTS fails, disable audio but continue with text
6. Image Analysis Failures: Return error message, allow text query

All errors are logged to console and logs/app.err for debugging.

■ Conclusion

AyurWell represents a sophisticated integration of modern AI technologies to create a practical, production-ready health assistance system. By combining RAG, multi-agent orchestration, multimodal inputs, and intelligent fallback mechanisms, the system provides reliable, accurate health information grounded in authoritative Ayurvedic texts.

Key Achievements:
• Multimodal interaction (text, voice, images)
• Bilingual support (English & Kannada)
• Intelligent query processing with refinement
• Reliable fallback mechanisms
• Evidence-based responses from curated knowledge base
• Modular, maintainable architecture

Future Enhancements:
• Support for more Indian languages (Hindi, Tamil, Telugu)
• Integration with telemedicine platforms
• Personalized health tracking and recommendations
• Voice-to-voice interaction (STT + TTS pipeline)
• Mobile app development (React Native / Flutter)
• Integration with wearable health devices
• Expanded knowledge base with more medical literature

This project demonstrates the practical application of cutting-edge AI technologies in the healthcare domain, providing a foundation for further research and development in AI-assisted medical guidance systems.

Contact Information:
Project Repository: [GitHub URL]
Developed by: Santosh, Viresh, and Vishwas
Institution: PES University, Semester 7
Course: AI in Healthcare