Medical Wellness Assistant

Advanced NLP-based Medical Question Answering System

Project Team

Aman Verma (MT2024020)
Deomani Singh (MT2024040)
Tushar Dubey (MT2024164)

MTech Program
Advanced Natural Language Project

November 2025

Table of Contents

1. Project Overview

In this project, we have developed a **Medical Question-Answering system** that addresses one of the most critical challenges in healthcare technology: providing accurate, reliable, and safe medical information to users. A person can ask any medical question, and our model delivers a clear and helpful answer based on verified medical knowledge.

Because the system focuses exclusively on medical topics, it provides better responses and significantly reduces wrong or fabricated information. In other words, **the system does not hallucinate** - a major problem in current AI systems.

This system functions like a smart medical assistant that uses modern Natural Language Processing (NLP) techniques to understand questions and generate accurate responses.

1.1 Core Components

1. **RAG (Retrieval-Augmented Generation):** The model answers using proper medical information from authoritative medical textbooks instead of guessing or generating random responses.

2. **Safety Checks:** Multiple layers of verification ensure that whatever answer is given is safe, reliable, and not harmful. The system performs four independent safety checks before providing any answer.

3. **Mistral-7B Model:** This is the main language model that generates the answers. We use it in zero-shot mode (without fine-tuning) combined with RAG context for medical domain adaptation.

4. **Simple Web Interface:** Users can easily type their questions in a modern, user-friendly interface and receive answers with proper citations.

1.2 Why This Matters

Medical misinformation is very common today, and wrong medical advice can be dangerous or even life-threatening. According to various studies, incorrect medical information on the internet has led to serious health consequences.

Our system prioritizes **safety over speed**, making sure that the information shared is correct and trustworthy. The best feature of our system is that if it is not fully confident or if the question is unsafe, it will simply abstain from answering instead of giving a wrong reply.

This approach makes our system particularly useful for real-life medical platforms where people depend on accurate guidance. It can be deployed in hospitals, clinics, health websites, or telemedicine platforms.

1.3 Key Innovation

Unlike traditional medical chatbots that often provide generic responses, our system combines **Retrieval-Augmented Generation (RAG)** with a **multi-layered safety pipeline**. This unique combination ensures:

- **Grounded Responses:** Every answer is backed by verified medical literature
- **Source Attribution:** Users can see which medical textbook and page the information comes from
- **Safe Abstention:** System refuses to answer when confidence is low
- **No Hallucination:** Multiple checks prevent fabricated medical information

2. Dataset Processing Pipeline

The foundation of any good AI system is high-quality training data. For our medical QA system, we carefully processed and merged multiple authoritative medical datasets to create a comprehensive knowledge base.

2.1 Raw Dataset Sources

We used three major medical QA datasets, each bringing unique strengths:

Dataset	Description	Format	Samples
MedDialog	Medical conversations between patients and doctors	JSON	~200,000
MedQuAD	Medical questions from NIH websites	CSV	~16,000
PubMedQA	Research-based medical question-answer pairs	CSV	~12,000

2.2 Dataset Merging Process

All three datasets were loaded, processed, and merged into a single unified format. The merging was done using a Jupyter notebook (*dataset_scripts/dataset_creation.ipynb*).

Standardized Schema: Each Q&A; pair was converted to a common format containing:

```
{
    "id": "unique_identifier",
    "question": "user question text",
    "answer": "medical answer text",
    "source": "dataset_name",
    "focus_area": "optional_category"
}
```

Final Output: A merged dataset file *datasets/final_dataset/merged_data.jsonl* containing **228,295 medical Q&A; pairs**.

2.3 Data Cleaning and Preprocessing

Before using the data, we performed extensive cleaning to ensure quality:

• **Duplicate Removal:** Identified and removed duplicate questions and answers

• **Length Filtering:** Removed answers shorter than 20 characters (too brief to be useful)

• **Empty Entry Removal:** Eliminated entries with missing or invalid data

• **Text Normalization:** Standardized text encoding to UTF-8

• **Whitespace Cleaning:** Removed excessive whitespace and formatting issues

```
def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = re.sub(r"\s+", " ", text) # Remove extra whitespace
    return text.strip()
```

2.4 Train/Validation/Test Split

The cleaned dataset was split into three parts for potential future model training and evaluation:

• **Training Set:** 80% of data (182,567 samples)

• **Validation Set:** 10% of data (22,821 samples)

• **Test Set:** 10% of data (22,821 samples)

• **Random State:** 77 (ensures reproducibility)

The split was performed using scikit-learn's *train_test_split* function, which ensures random distribution while maintaining reproducibility.

2.5 Instruction Format Conversion

To prepare the dataset for potential future fine-tuning (though the current system uses zero-shot inference), we converted Q&A; pairs into an instruction-following format. This format is commonly used for training instruction-tuned language models.

```
# Input Format
{"question": "What are symptoms of flu?", "answer": "Fever, cough..."}

# Output Format (Instruction Format)
{
    "instruction": "What are symptoms of flu?",
    "input": "",
    "output": "Fever, cough, body aches..."
}
```

The conversion script filters out any entries with answers shorter than 20 characters and saves the processed data as *train_inst.jsonl*, *val_inst.jsonl*, and *test_inst.jsonl*.

3. PDF Processing and RAG Construction

Retrieval-Augmented Generation (RAG) is the core technology that allows our system to provide accurate, grounded medical responses. Instead of relying solely on the language model's internal knowledge, we retrieve relevant information from authoritative medical textbooks and use that context to generate answers.

3.1 Medical Textbook Collection

We selected four authoritative medical textbooks covering different specialties:

Book Title	Specialty	Edition	Pages Used
Ramdas Nayak Pathology Book	Pathology	-	Skip first 27, last 41
Gale Encyclopedia of Medicine	General Medicine	3rd	Skip first 30, last 440
Oxford Handbook of Clinical Medicine	Clinical Medicine	10th	Skip first 14, last 47
KD Tripathi Pharmacology	Pharmacology	-	Skip first 17, last 76

We skip introductory pages (cover, table of contents) and end pages (references, indexes) to focus on the core medical content.

3.2 PDF Text Extraction and Cleaning

Extracting clean text from medical PDFs is challenging because they contain headers, footers, watermarks, tables, figures, and complex formatting. Our preprocessing pipeline handles all these issues.

Step 1: Text Extraction

We use the *pdfplumber* library to extract text page by page. This library handles complex PDF structures better than simple text extraction methods.

```
import pdfplumber

with pdfplumber.open(pdf_path) as pdf:
    for page_num in range(start_page, end_page):
        page = pdf.pages[page_num]
        raw_text =
        page.extract_text() # Process
        raw_text...
```

Step 2: Text Cleaning

We remove non-content elements that would confuse the retrieval system:

• **Headers & Footers:** Pattern matching for "OXFORD HANDBOOK", "Page 123", etc.

• **Watermarks:** "Downloaded from...", website URLs

• **Table Artifacts:** Lines with excessive whitespace (4+ consecutive spaces)

• **Figure Labels:** Lines starting with "Fig", "Figure", "Plate", "Table"

• **Edition Info:** "3rd Edition", "10th Edition", etc.

```
HEADER_FOOTER_PATTERNS = [
    r"OXFORD HANDBOOK.*",
    r"Downloaded\sfrom.*",
    r"Tripathi.*Pharmacology.*",
    r"Page\s*\d+",
    r"www\..*"
]

def clean_line(line):
    for pattern in HEADER_FOOTER_PATTERNS:
        if re.search(pattern, line,
            re.IGNORECASE): return "" # Remove
            this line
    return line.strip()
```

Step 3: Text Normalization

After cleaning, we normalize the text:

• Merge broken lines into continuous paragraphs

• Remove excessive whitespace

• Filter out pages with less than 200 characters (mostly non-content)

• Stop processing when "REFERENCES" or "Bibliography" sections appear

Step 4: Chunking Strategy

Medical textbooks can be thousands of pages long. We need to break them into smaller, manageable pieces (chunks) for efficient retrieval. Our chunking strategy:

• **Chunk Size:** 250 words per chunk

• **Overlap:** 25 words overlap between consecutive chunks (10% overlap)

• **Reason for Overlap:** Preserves context across chunk boundaries

• **Minimum Chunk Size:** 20 words (discard smaller chunks)

```
def chunk_text(words, chunk_size=250,
    overlap=25): chunks = []
    step = chunk_size - overlap

    for i in range(0, len(words), step):
        chunk = words[i:i+chunk_size]
        if len(chunk) > 20:
            chunks.append("
            ".join(chunk))
    return chunks
```

3.3 Embedding Generation

Once we have clean text chunks, we need to convert them into numerical vectors (embeddings) that capture their semantic meaning. This allows us to find similar chunks when a user asks a question.

Embedding Model: We use **BAAI/bge-large-en-v1.5**, which is:

• A state-of-the-art sentence transformer model

• Produces 1024-dimensional embeddings

• Trained on diverse text including scientific literature

• Excellent performance on semantic similarity tasks

```
from sentence_transformers import SentenceTransformer

embedder = SentenceTransformer("BAAI/bge-large-en-v1.5")

# Embed all chunks
embeddings = embedder.encode(
    all_texts,
    batch_size=4, # 4 for GPU, 16 for CPU
    normalize_embeddings=True, # L2 normalization
    show_progress_bar=True
)
```

3.4 FAISS Index Construction

FAISS (Facebook AI Similarity Search) is a library for efficient similarity search. We use it to quickly find the most relevant chunks for any query.

Index Type: IndexFlatIP (Inner Product)

• Works with normalized embeddings for cosine similarity

• Exact search (no approximation)

• Fast retrieval even with 15,000+ chunks

```
import faiss

dim = embeddings.shape[1] # 1024
```

```
index = faiss.IndexFlatIP(dim)
index.add(embeddings.astype("float32"))

# Save for later use
faiss.write_index(index,
"faiss_index.bin")
```

3.5 Metadata Mapping

For each chunk in the FAISS index, we store metadata to track its source:

• Book name

• Page number

• Preview of the text (first 300 characters)

This mapping allows us to cite sources when providing answers to users.

3.6 RAG Query Engine

The RAG Query Engine ties everything together. When a user asks a question:

Step	Action
1	Embed user question using BGE model
2	Search FAISS index for top-5 similar chunks
3	Retrieve chunk metadata (book, page, text)
4	Build context by combining retrieved chunks
5	Create prompt with question + context
6	Pass to language model for answer generation

4. Safety Pipeline Architecture

The safety pipeline is the most critical component of our system. It ensures that only medically accurate, well-supported answers reach the user. If any check fails, the system abstains from answering.

Core Philosophy: It is better to not answer than to give wrong medical information.

4.1 Safety Check Overview

The system performs **four sequential safety checks**:

Check	Purpose	Threshold
1. Retrieval Confidence	Are retrieved chunks relevant to query?	Top-1 ≥ 0.55 Mean-3 ≥ 0.50
2. Consistency	Does model give same answer?	Similarity ≥ 0.75
3. Model Confidence	Is model confident in its word choices?	Log-prob ≥ -2.5
4. Entailment	Is answer supported by context?	≥ 60% sentences entailed

4.2 Check 1: Retrieval Confidence

Purpose: Verifies that the retrieved medical text chunks are actually relevant to the user's question.

How it works:

• Calculates similarity score between query and each retrieved chunk

• Checks if top-1 score ≥ 0.55 (best match is relevant)

• Checks if mean of top-3 scores ≥ 0.50 (overall relevance)

Abstain if: Either threshold is not met

Reason: If we cannot find relevant medical information, we should not guess

```
def check_retrieval_confidence(retrieved, top1_thr=0.55, mean3_thr=0.50):
    scores = [chunk['score'] for chunk in retrieved]
    top1 = scores[0]
    mean3 = np.mean(scores[:3])

    if top1 < top1_thr or mean3 < mean3_thr:
        return False, "Low retrieval confidence"
    return True, "Retrieval confident"
```

4.3 Check 2: Consistency Check

Purpose: Ensures the model generates consistent answers across multiple runs.

How it works:

• Generate the same answer 2-3 times with different random seeds

• Use same prompt but vary temperature slightly

• Compute semantic similarity between all answer pairs

• Calculate mean pairwise similarity

Abstain if: Mean similarity < 0.75

Reason: If the model gives different answers to the same question, it indicates uncertainty about the correct response

```
def check_consistency(generate_fn, prompt, n=3, threshold=0.75):
    samples = []
    for i in range(n):
        answer = generate_fn(prompt, seed=1000+i)
        samples.append(answer)

    # Compute pairwise similarity
    embeddings = embedder.encode(samples)
    similarities = []
    for i in range(len(samples)):
        for j in range(i+1, len(samples)):
            sim = cosine_similarity(embeddings[i], embeddings[j])
            similarities.append(sim)

    mean_sim = np.mean(similarities)
    return mean_sim >= threshold
```

4.4 Check 3: Model Confidence (Log Probability)

Purpose: Measures how confident the language model is in its generated tokens.

How it works:

• During generation, extract log-probability for each token

• Log-probability indicates model's confidence (higher = more confident)

• Calculate average log-probability across all tokens

Abstain if: Average log-probability < -2.5

Reason: Low log-probability means the model is uncertain about its word choices, suggesting the answer may be unreliable

```
def compute_avg_logprob(generate_output):
    token_logprobs = []

    for logits, token_id in zip(generate_output.scores,
        token_ids): logprob = log_softmax(logits)[token_id]
        token_logprobs.append(logprob)

    return np.mean(token_logprobs)
```

4.5 Check 4: Entailment Verification

Purpose: Verifies that the generated answer is logically supported by the retrieved medical context. This is the most important check for preventing hallucination.

How it works:

• Use a specialized biomedical NLI model: **PubMedBERT-MNLI-MedNLI**

• Split generated answer into individual sentences

• For each sentence, check if it is entailed by any retrieved chunk

• Calculate percentage of sentences that are entailed

Abstain if: Less than 60% of sentences are entailed

Reason: If most of the answer cannot be verified against the source material, it likely contains hallucinated information

```
def entailment_check(hypotheses,
    retrieved_texts): entailed_count = 0

    for hypothesis in hypotheses: # Each sentence
        best_entailment_prob = 0

        for premise in retrieved_texts: # Each chunk
            # NLI model: Does premise entail hypothesis?
            prob = nli_model.predict(premise, hypothesis)
            best_entailment_prob = max(best_entailment_prob, prob)

        if best_entailment_prob >= 0.6:
            entailed_count += 1

    entailment_percentage = entailed_count / len(hypotheses)
    return entailment_percentage >= 0.60
```

4.6 Complete Safety Pipeline

All four checks are performed sequentially. If any check fails, the system immediately abstains without proceeding to subsequent checks.

Step	Action	On Failure
1	Retrieve top-5 chunks	Continue
2	Check retrieval confidence	ABSTAIN
3	Generate answer (consistency check)	ABSTAIN
4	Generate with log-probabilities	ABSTAIN
5	Check entailment	ABSTAIN
6	All checks passed	ACCEPT & RETURN

Abstain Message: When the system abstains, it displays: *"I'm not confident enough to answer safely."*

This transparency builds trust with users, who appreciate honesty about the system's limitations.

5. Model Integration

Our system uses **Mistral-7B-Instruct**, a powerful open-source language model, to generate medical responses. We use it in zero-shot mode combined with RAG context.

5.1 Why Mistral-7B?

We chose Mistral-7B for several reasons:

• **Instruction Following:** Excellent at following prompts and instructions

• **Model Size:** 7 billion parameters - good balance of performance and efficiency

• **GGUF Format:** Quantized format allows running on consumer hardware

• **Open Source:** Can be deployed without API costs

• **Context Window:** 4096 tokens - sufficient for RAG context

5.2 Zero-Shot vs Fine-Tuning

Current Approach: Zero-Shot

We use the base Mistral-7B model without fine-tuning. The model relies on:

• RAG-provided medical context for domain knowledge

• Carefully crafted prompts

• Its pre-trained instruction-following capabilities

Why Zero-Shot?

• **No GPU Requirements:** Fine-tuning requires expensive GPU resources

• **Flexibility:** Can update medical knowledge by updating PDF books

• **Proven Effectiveness:** RAG provides sufficient medical grounding

Future Enhancement: A fine-tuning script exists in the codebase (*training_scripts/mistral_finetune_qLoRA.py*) that could be used with GPU resources.

5.3 Inference Setup

We use **llama-cpp-python** library for efficient inference with GGUF models.

```
from llama_cpp import Llama

llm = Llama(
    model_path="models/mistral-7b-instruct.gguf
    ", n_ctx=4096,       # Context window
    n_gpu_layers=35,     # GPU acceleration
    n_threads=6,         # CPU threads
    logits_all=True,     # For log-probability
    extraction verbose=False
)
```

5.4 Text Generation

For the safety pipeline, we need generation with metadata (log-probabilities, token information).

```
def mistral_generate_with_meta(prompt, seed=0,
    temperature=0.2): out = llm.create_completion(
        prompt=prompt,
        max_tokens=256,
        temperature=temperature,
        logprobs=1, # Enable log-probabilities
        stop=["</s>", "###"]
    )

    text = out["choices"][0]["text"]
    logprobs =
    out["choices"][0]["logprobs"]
    avg_logprob = np.mean(logprobs["token_logprobs"])

    return {
        "text": text.strip(),
        "avg_logprob": avg_logprob,
        "tokens": logprobs["tokens"]
    }
```

5.5 RAG + Safety + Model Integration

The complete flow integrates all components:

```
def ask(query):
    # Step 1: Retrieve relevant chunks
    retrieved = rag.retrieve(query, k=5)

    # Step 2: Build RAG prompt
    prompt = build_prompt(query, retrieved)

    # Step 3: Run safety pipeline
    decision = safety_check_and_answer(
        query, retrieved, prompt,
        generator_fn=mistral_generate_with_met
        a, nli_model="PubMedBERT-MNLI-MedNLI"
    )

    # Step 4: Return result
    if decision["status"] ==
        "accept": return
        decision["answer"]
    else:
        return "I'm not confident enough to answer safely."
```

6. API and Web Application

Our system provides both a REST API and a modern web interface for users.

6.1 FastAPI Backend

The backend is built with **FastAPI**, a modern Python web framework.

Main Endpoints:

• **GET /health:** Health check

• **POST /new_session:** Create new chat session

• **POST /chat:** Send question and receive answer

• **POST /clear_memory:** Clear session history

6.2 Express.js Middleware

A TypeScript Express server acts as a proxy between the React frontend and FastAPI backend, providing request validation and error handling.

6.3 React Frontend

The user interface is built with **React 18 + TypeScript** and includes:

• **Modern Chat Interface:** Clean, intuitive design

• **Session Management:** Maintains conversation history

• **Developer Mode:** Toggle to view safety metrics

• **Responsive Design:** Works on desktop and mobile

• **Source Citations:** Shows which book and page information comes from

7. Complete System Flow

Let's walk through what happens when a user asks: *"What are the symptoms of diabetes?"*

#	Component	Action
1	User Interface	User types question and clicks Send
2	React Frontend	Sends POST request to Express server
3	Express Server	Validates request, forwards to FastAPI
4	FastAPI Backend	Calls rag_query_engine_safe.ask()
5	RAG Engine	Embeds query, searches FAISS index
6	Safety Check 1	Verifies retrieval confidence
7	Prompt Builder	Combines query + retrieved context
8	Safety Check 2	Generates multiple answers, checks consistency
9	Mistral-7B	Generates primary answer with log-probabilities
10	Safety Check 3	Verifies model confidence
11	Safety Check 4	Checks entailment with NLI model
12	Decision	Accept if all checks pass, else Abstain
13	Response	Returns answer + metadata to frontend
14	UI Display	Shows answer with source citations

7.1 Abstain Scenarios

The system abstains (refuses to answer) in these situations:

• **Low Retrieval Confidence:** Retrieved medical texts don't match the question well

• **Inconsistent Answers:** Model gives different responses to the same question

• **Low Model Confidence:** Model is uncertain about its word choices

• **Insufficient Evidence:** Generated answer cannot be verified against source material

8. Installation & Setup

Setting up the system involves both backend and frontend configuration.

8.1 Prerequisites

- Python 3.8 or higher
- Node.js 18 or higher
- GPU with CUDA (optional, for faster inference)
- At least 8GB RAM (16GB recommended)

8.2 Backend Setup

Step 1: Install Python dependencies

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

Step 2: Download Mistral-7B GGUF model from HuggingFace

Step 3: Process datasets (run Jupyter notebooks)

Step 4: Process PDFs and build FAISS index

Step 5: Start FastAPI server

```
uvicorn api:app --host 127.0.0.1 --port 8000
```

8.3 Frontend Setup

Step 1: Install Node.js dependencies

```
npm install
```

Step 2: Start development server

```
npm run dev
```

9. Usage

The system is designed to be simple and intuitive.

9.1 Web Interface

1. Open browser and navigate to the application

2. Type your medical question in the input box

3. Press Enter or click Send

4. View the answer with source citations

5. Enable Developer Mode to see safety metrics

9.2 API Usage

You can also interact with the system programmatically:

```
import requests

response = requests.post("http://127.0.0.1:8000/chat", json={
    "message": "What are the symptoms of diabetes?",
    "session_id": null,
    "developer_mode": true
})

data = response.json()
print(data["answer"])
print(data["meta"])
```

10. File Structure

The project is organized into clear, modular components.

```
Medical QA/
●  dataset_scripts/       # Dataset processing
      ○  dataset_creation.ipynb
      ○  eda&data_split.ipynb
      ○  convert_to_instruction.py
      ○  pdf_preprocess_and_chunk.py
●  rag/                   # RAG system
      ○  embed_and_build_faiss.py
      ○  rag_query_engine.py
      ○  rag_query_engine_safe.py
●  safety_scripts/        # Safety checks
      ○  safety_pipeline.py
      ○  safety_retrieval.py
      ○  safety_consistency.py
      ○  safety_logprob.py
      ○  safety_entailment.py
●  inference_scripts/     # Model inference
      ○  mistral_inference.py
●  api.py                 # FastAPI backend
●  requirements.txt


MediChatUI/
●  client/                # React frontend
      ○  src/
      ○  components/
      ○  pages/
      ○  hooks/
●  server/                # Express middleware
●  routes.ts
```

11. Key Design Decisions

Several important design decisions shaped this project.

11.1 Why RAG Instead of Fine-Tuning?

Advantages of RAG:

• Easily update medical knowledge by adding new PDFs

• No expensive GPU training required

• Provides source citations for transparency

• Prevents outdated medical information

Fine-tuning would require:

• Expensive GPU resources (A100/H100)

• Days/weeks of training time

• Regular retraining as medical knowledge updates

11.2 Why Multiple Safety Checks?

Each safety check catches different types of errors:

• **Retrieval check:** Catches irrelevant questions

• **Consistency check:** Catches model uncertainty

• **Log-probability check:** Catches low-confidence generations

• **Entailment check:** Catches hallucinations

Using all four provides comprehensive safety coverage.

11.3 Why Abstain Instead of Always Answering?

In medical applications, **safety trumps completeness**. Wrong medical advice can cause harm. Users actually trust the system more when it admits limitations rather than always providing an answer.

11.4 Why Zero-Shot Mistral-7B?

• **Accessibility:** Can run on consumer hardware

• **Open Source:** No API costs or data privacy concerns

• **Good Performance:** Combined with RAG, provides excellent results

• **Instruction Following:** Mistral-7B follows prompts very well

11.5 Why FAISS for Vector Search?

• **Speed:** Very fast similarity search

• **Scalability:** Handles 15,000+ chunks easily

• **Accuracy:** Exact search (no approximation needed at this scale)

• **Industry Standard:** Widely used and well-tested

11.6 Conclusion and Screenshots

This Medical Wellness Assistant project demonstrates how modern NLP techniques can be combined to create a safe, reliable medical question-answering system. By integrating **Retrieval-Augmented Generation** with a **multi-layered safety pipeline**, we have created a system that:

• Provides accurate medical information grounded in authoritative sources

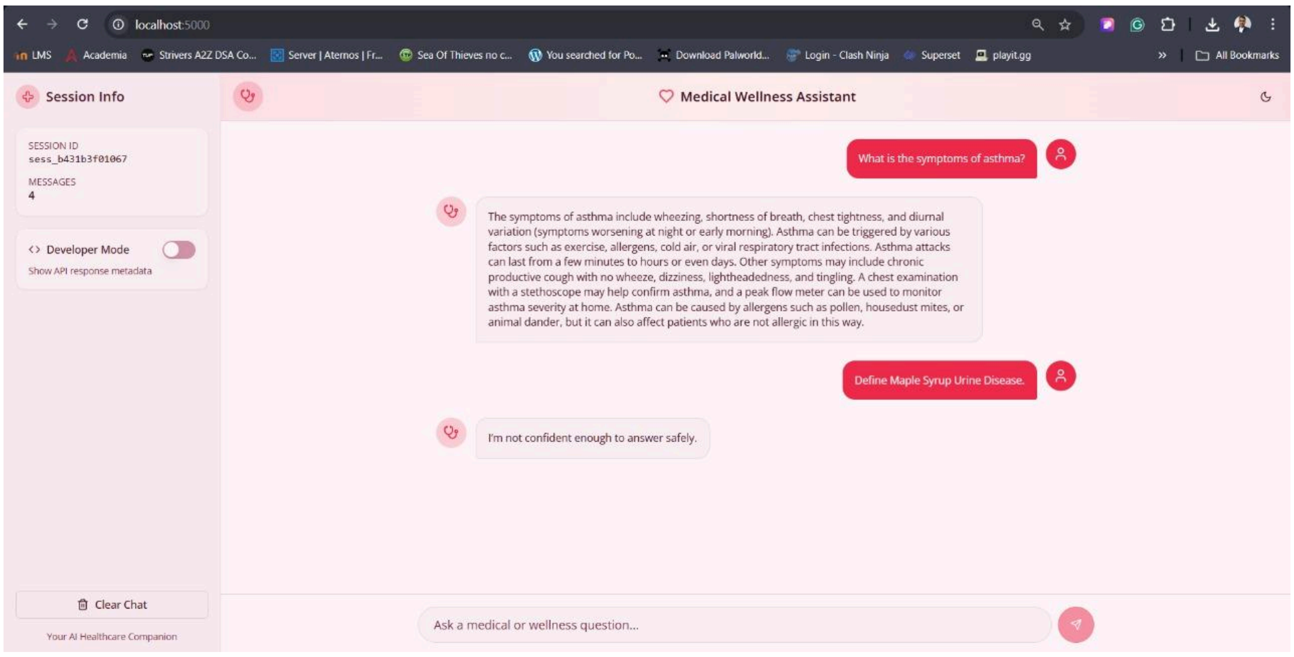
• Refuses to answer when confidence is insufficient

• Cites sources for transparency

• Can be deployed on consumer hardware

• Maintains user trust through honest communication about limitations

The project successfully addresses the critical challenge of preventing medical misinformation while maintaining practical usability. Future enhancements could include fine-tuning with GPU resources, expanding the medical textbook collection, and adding multi-language support.


localhost:5000
Session Info
Medical Wellness Assistant
Developer Mode
Show API response metadata
Clear Chat
Your AI Healthcare Companion
Ask a medical or wellness question...

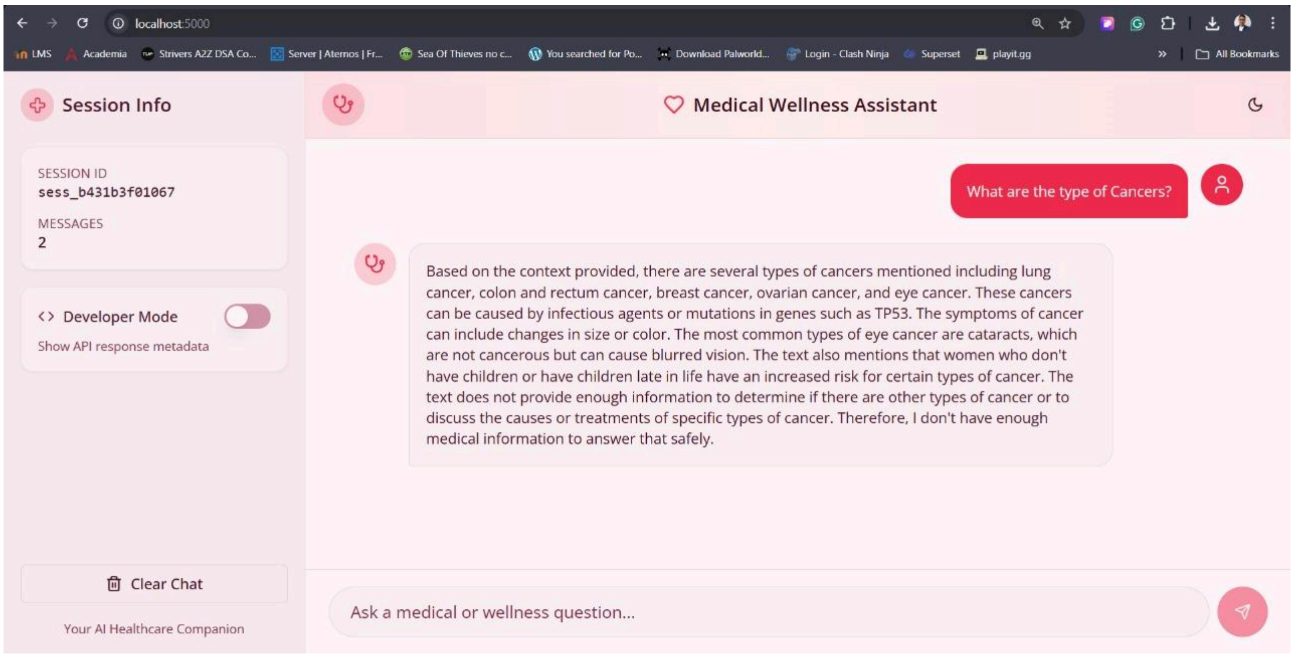
SESSION ID
sess_b431b3f01067
MESSAGES
4

What is the symptoms of asthma?

The symptoms of asthma include wheezing, shortness of breath, chest tightness, and diurnal variation (symptoms worsening at night or early morning). Asthma can be triggered by various factors such as exercise, allergens, cold air, or viral respiratory tract infections. Asthma attacks can last from a few minutes to hours or even days. Other symptoms may include chronic productive cough with no wheeze, dizziness, lightheadedness, and tingling. A chest examination with a stethoscope may help confirm asthma, and a peak flow meter can be used to monitor asthma severity at home. Asthma can be caused by allergens such as pollen, housedust mites, or animal dander, but it can also affect patients who are not allergic in this way.

Define Maple Syrup Urine Disease.

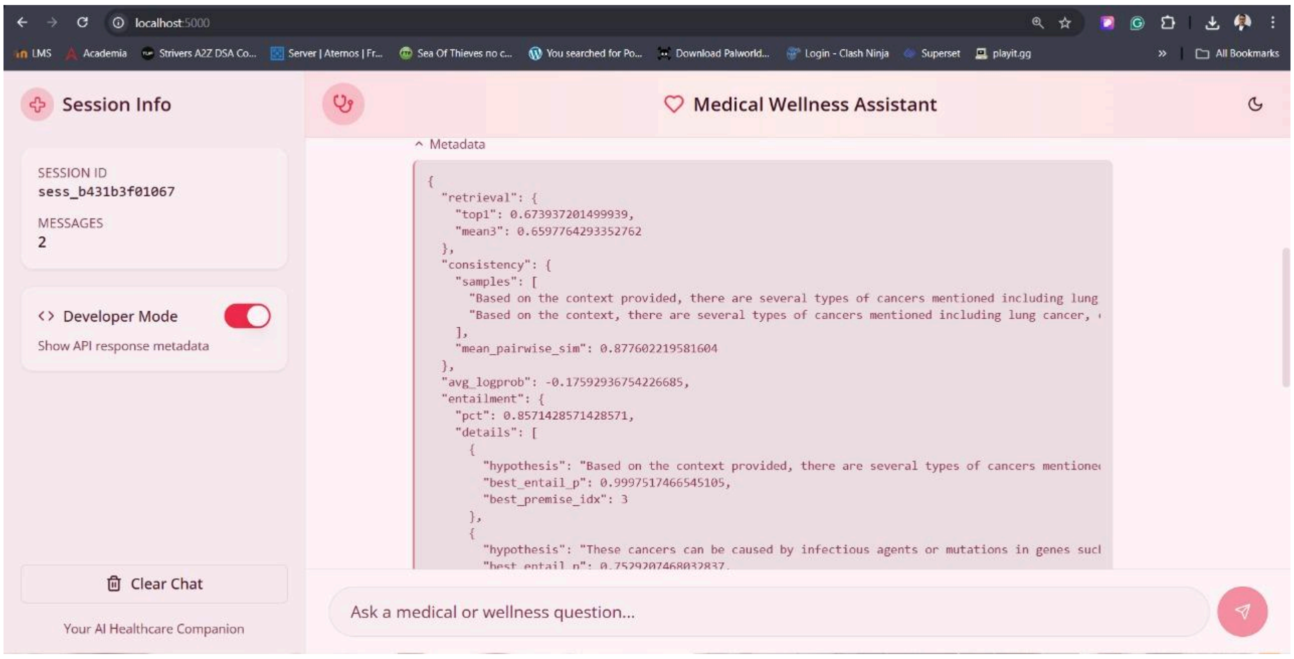
I'm not confident enough to answer safely.


localhost:5000
Session Info
Medical Wellness Assistant
Developer Mode
Show API response metadata
Clear Chat
Your AI Healthcare Companion
Ask a medical or wellness question...

SESSION ID
sess_b431b3f01067
MESSAGES
2

What are the type of Cancers?

Based on the context provided, there are several types of cancers mentioned including lung cancer, colon and rectum cancer, breast cancer, ovarian cancer, and eye cancer. These cancers can be caused by infectious agents or mutations in genes such as TP53. The symptoms of cancer can include changes in size or color. The most common types of eye cancer are cataracts, which are not cancerous but can cause blurred vision. The text also mentions that women who don't have children or have children late in life have an increased risk for certain types of cancer. The text does not provide enough information to determine if there are other types of cancer or to discuss the causes or treatments of specific types of cancer. Therefore, I don't have enough medical information to answer that safely.



Session Info

SESSION ID
sess_b431b3f01067

MESSAGES
2

Developer Mode

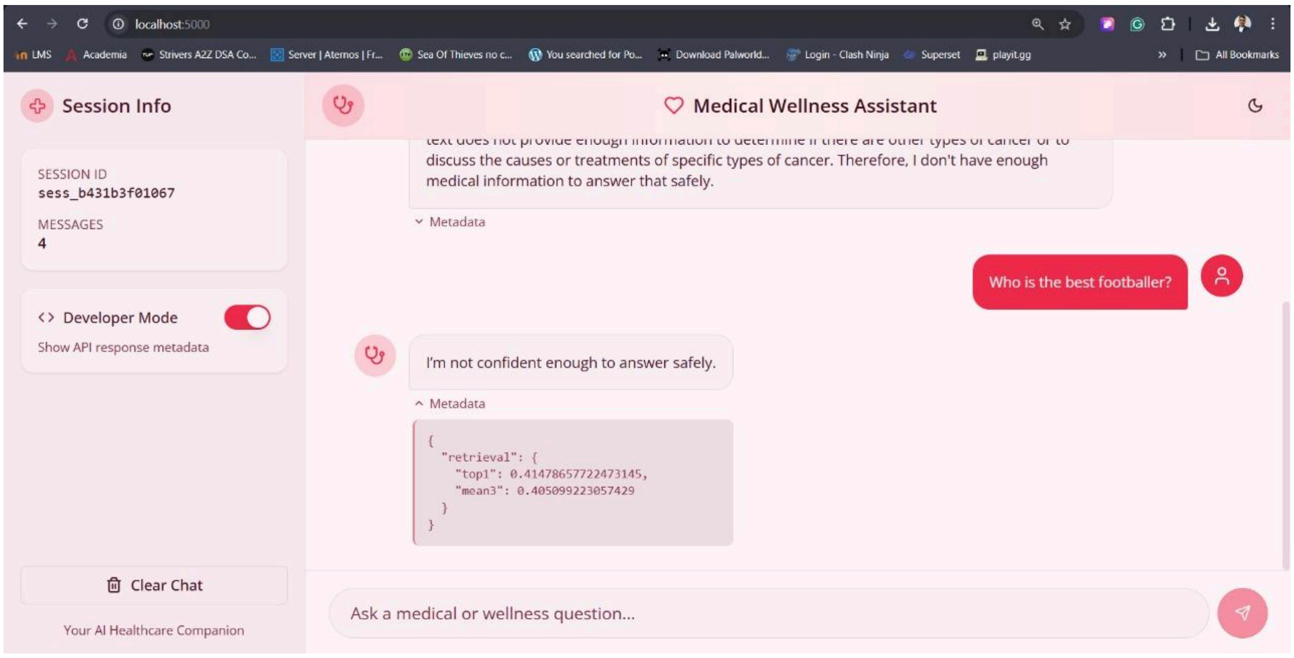
Show API response metadata

Medical Wellness Assistant

Metadata

```
{
  "retrieval": {
    "top1": 0.673937201499939,
    "mean3": 0.6597764293352762
  },
  "consistency": {
    "samples": [
      "Based on the context provided, there are several types of cancers mentioned including lung
      "Based on the context, there are several types of cancers mentioned including lung cancer,
    ],
    "mean_pairwise_sim": 0.877602219581604
  },
  "avg_logprob": -0.17592936754226685,
  "entailment": {
    "pct": 0.8571428571428571,
    "details": [
      {
        "hypothesis": "Based on the context provided, there are several types of cancers mentione
        "best_entail_p": 0.9997517466545105,
        "best_premise_idx": 3
      },
      {
        "hypothesis": "These cancers can be caused by infectious agents or mutations in genes suc
        "best_entail_p": 0.7529207468032837,
```

Ask a medical or wellness question...



Session Info

SESSION ID
sess_b431b3f01067

MESSAGES
4

Developer Mode

Show API response metadata

Medical Wellness Assistant

text does not provide enough information to determine if there are other types of cancer or to discuss the causes or treatments of specific types of cancer. Therefore, I don't have enough medical information to answer that safely.

Metadata

Who is the best footballer?

I'm not confident enough to answer safely.

Metadata

```
{
  "retrieval": {
    "top1": 0.41478657722473145,
    "mean3": 0.405099223057429
  }
}
```

Ask a medical or wellness question...