DIY DOCTOR: A Gen AI-Powered Personal Health Assistant

David R. Treadwell IV
Khoury College of Computer Science
Northeastern University
Seattle, WA, USA
treadwell.d@northeastern.edu

Hong Yang
Khoury College of Computer Science
Northeastern University
Seattle, WA, USA
yang.hongyu@northeastern.edu

Riddhi Gohil
Khoury College of Computer Science
Northeastern University
Seattle, WA, USA
gohil.r@northeastern.edu

Abstract

DIY Doctor is a GenAI-powered health assistant that enables users to query their personal medical records securely and receive context-aware responses. The application uses a retrieval-augmented generation (RAG) pipeline, combining LLMs trained in different domains with semantic search techniques to provide accurate, relevant, and trustworthy answers. We explore the performance of domain-specific and generalist language models in both response generation and verification tasks, using a custom veracity scoring system. DIY Doctor supports private user authentication, ensuring that responses are personalised and limited to an individual's data. We evaluate the performance of six models across medical, generalist, and programming domains, measuring retrieval accuracy, response veracity, and inference latency. Our results demonstrate that domain-specific models, such as OpenBioLLM-8B, outperform generalist models in maintaining medical relevance and factual consistency. However, certain generalist models like Mistral-7B show strong retrieval capabilities and competitive evaluation scores. These findings validate the feasibility of combining GenAI with personalised health records to deliver trustworthy, patient-specific medical guidance, and highlight future opportunities for optimizing model selection and deployment strategies in real-world applications.

1 Introduction

In an era where convenience is king, where food, rides, and answers are just a click away, accessing trustworthy medical advice remains unnecessarily complex. While the internet can answer broad health queries, it often fails to consider the nuanced realities of individual medical conditions, allergies, medications, and family history. This disconnect between general advice and personalised care has long been a challenge, particularly in the medical field, where the need for accurate and tailored guidance is crucial. In recent years, the field of Artificial Intelligence (AI) and Machine Learning (ML) has made remarkable strides, particularly in the realm of personalised medicine. AI systems can now analyse vast amounts of health data, such as medical records, genetic information, and lifestyle factors, to provide tailored recommendations that were once out of reach for traditional healthcare systems. These advancements not only improve accuracy but also allow for real-time, dynamic responses to individual health concerns. The rapid advancements in Generative AI (GenAI)

technology present an opportunity to address this gap, enabling the creation of intelligent systems that can deliver personalised and reliable health information. This vision inspired the development of DIY Doctor.

DIY Doctor is an innovative, Genai-powered health assistant designed to deliver tailored medical guidance based on a user's personal data and health history. With the rise of GenAI, which has demonstrated remarkable abilities in natural language understanding, synthesis, and customisation, DIY Doctor leverages these capabilities to provide context-specific health advice. Users can securely log in using unique credentials—username and password—to access their own medical data stored in a MongoDB database. This approach ensures privacy and prevents unauthorised access, while still enabling personalised health insights.

The core of the application utilises a **Retrieval-Augmented Generation** (RAG) framework, integrating fine-tuned large language models (LLMS), medical domain-specific embeddings, and semantic search. Central to this architecture is a dual-LLM pipeline: the primary LLM generates responses based on the retrieved context, while a secondary Judge LLM independently evaluates the response for faithfulness (alignment with the source context) and domain relevance (alignment with the medical domain). This layered approach is further strengthened by the Judge LLM, which serves as a verification layer, ensuring the reliability of medical suggestions. Additionally, semantic search is employed over structured representations of medical records, obtained through a node-level parser, to enable efficient and meaningful retrieval. This combination not only enhances response accuracy but also fosters user trust by presenting appropriate disclaimers, clarifying that the suggestions provided should be verified with a healthcare professional.

This project was motivated by both academic and personal drivers, including prior coursework in information retrieval, natural language processing, and GenAI systems. It addresses a critical gap in current healthcare technologies: the lack of accessible, readable, individualised, and verifiable medical guidance. In this paper, we describe the design and implementation of DIY Doctor, including its modular architecture, retrieval strategies, model evaluation framework, and interactive user interface. We also present a custom veracity scoring methodology to assess the reliability of generated outputs.

2 Problem Statement

This research aims to address the growing challenge of delivering personalised, accurate, and trustworthy medical advice through digital platforms. While online health assistants are widely used, they often fail to provide tailored suggestions that account for an individual's unique health conditions, medications, allergies, and family history. Furthermore, the lack of verification mechanisms in many existing systems leaves users exposed to inaccurate or unreliable health information, which can result in negative health outcomes.

DIY Doctor seeks to solve this issue by leveraging GenAI technology to create a health assistant capable of generating personalised medical guidance. The core problem this project addresses is the absence of a reliable system that not only provides individualised advice but also ensures its accuracy and relevance to the medical domain.

3 Literature Review

Artificial intelligence (AI) has significantly advanced healthcare delivery by improving diagnosis, treatment planning, and operational efficiency. From analysing medical images to enabling robotic surgeries, AI systems have demonstrated remarkable capabilities. However, most existing AI applications focus on general trends or population-level insights, often lacking the ability to personalise guidance based on an individual's unique medical history. This literature review critically surveys prior research in personalised medicine, AI-driven health assistants, generative models, and retrieval-augmented techniques, highlighting their relevance and limitations in addressing the need for individualised, trustworthy medical advice.

3.1 Related Research

3.1.1 AI-Powered Health Assistants or chatbots

AI-driven virtual assistants have improved healthcare accessibility by providing quick responses to patient inquiries, assisting with appointment scheduling, and offering medication reminders. These systems help reduce the workload on healthcare professionals and promote proactive health management among patients. However, traditional health chatbots often lack the ability to personalise responses based on an individual's detailed medical history and are prone to producing generic or incomplete recommendations. These limitations highlight the need for more context-aware, trustworthy AI systems tailored to personal health records, motivating the development of DIY Doctor.

3.1.2 Generative AI in Healthcare

In the field of AI-powered healthcare communication, recent advancements have yielded impressive results. Notably, a BERT-based medical chatbot has demonstrated exceptional performance metrics, achieving 98% accuracy in handling medical queries and a 97% precision score, indicating highly reliable responses. The system utilises Bidirectional Encoder Representations from Transformers (BERT) to overcome traditional chatbot limitations, such as poor understanding of medical terminology and inability to provide personalised feedback. With an AUC-ROC score of 97%, the chatbot shows strong predictive capabilities for specific diseases based on user inputs. Additionally, its 96% recall rate ensures comprehensive coverage in medical diagnoses, minimising missed conditions. The system's overall F1 score of 98% reflects its balanced performance in delivering accurate, personalised healthcare information, positioning it as a valuable advancement in healthcare communication technology.

3.1.3 Retrieval-Augmented Generation (RAG) in Healthcare

Recent advances in Retrieval-Augmented Generation (RAG) frameworks have enabled language models to ground their responses by retrieving relevant documents before generating outputs. Prominent examples include MedPaLM-2 [1], which applies RAG techniques to clinical question answering, and [2], which demonstrates enhanced factuality in open-domain and specialised healthcare tasks. While these systems have significantly improved factual consistency, they primarily operate on curated datasets rather than personalised patient records, leaving a gap for user-specific medical guidance. Applications of RAG in personalised healthcare settings—where retrieval is based on individual patient data—remain relatively underexplored. DIY Doctor addresses this gap by embedding personal medical data into the retrieval pipeline, ensuring that generated responses are not only relevant but also tailored to the unique medical history of each user. This approach mitigates the risk of generic or hallucinated answers that can occur when LLMS operate without retrieval grounding.

3.1.4 Dual-LLM Architectures and Hallucination Mitigation

A growing body of research recognises the limitations of relying solely on a single generative model for critical tasks like medical advice. Dual-LLM architectures, in which one model generates responses and a second independently verifies them, have emerged as a promising strategy for improving factuality and domain consistency. Faithfulness evaluation, relevancy scoring, and veracity labelling are increasingly incorporated into GenAI pipelines to minimise hallucinations—incorrect or misleading outputs that can have serious consequences in healthcare applications. DIY Doctor adopts this dual-LLM approach by integrating a Judge LLM that assesses the faithfulness and relevance of the generated outputs, providing users with a veracity score ("GOOD," "VERIFY," or "BAD") to guide safe decision-making.

By integrating insights from advancements in AI-powered assistants, generative models, retrieval-based systems, and dual-LLM verification strategies, DIY Doctor builds upon existing research while addressing the critical gap of personalised, verifiable, and trustworthy healthcare guidance.

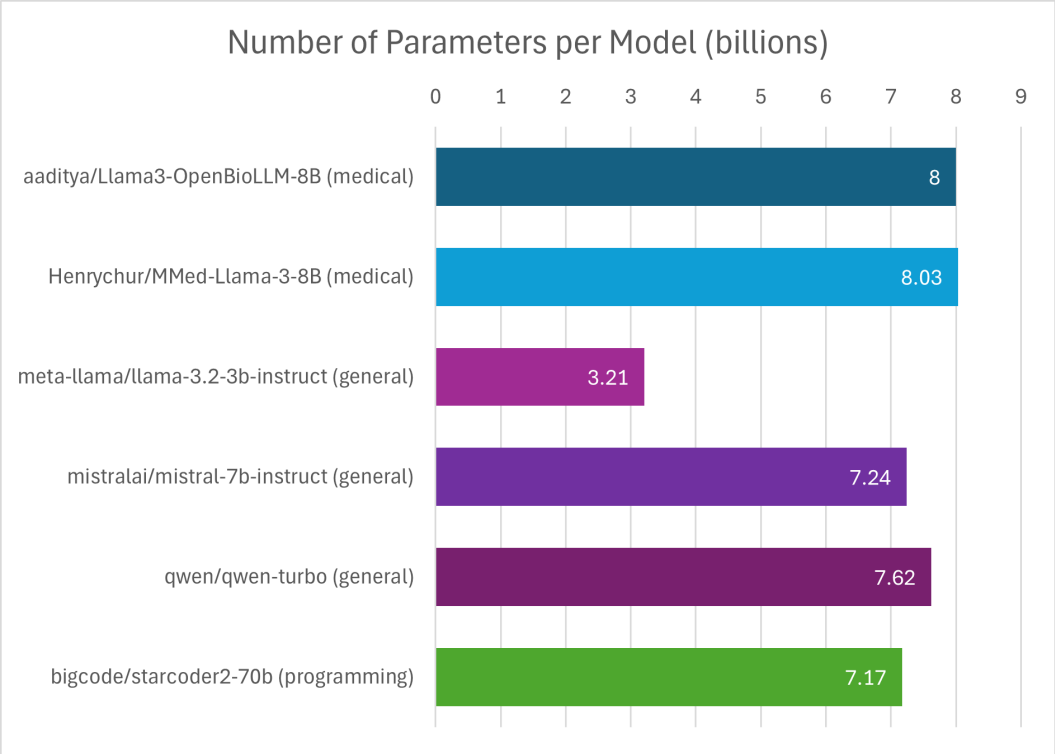

Figure 1: Comparison of the six models used in this project, with a label for its domain and the number of parameters used by the model (in billions)

4 Model Comparison

In this project, we test the performance of six different models, trained on three domains, to evaluate how each one performs as both a query engine LLM and a judge LLM. These six models are, with the format of *name (domain)*:

1. aaditya/Llama3-OpenBioLLM-8B (medical) [3]
2. Henrychur/MMed-Llama-3-8B (medical) [4]
3. meta-llama/llama-3.2-3b-instruct (general) [5]
4. mistralai/mistral-7b-instruct (general) [6]
5. qwen/qwen-turbo (general) [7] [8]
6. bigcode/starcoder2-70b (programming) [9]

One can infer from the model names the size of each model, but we also include a chart in Figure 1 to visualise and specify these differences. Most models contain around eight billion parameters, with meta-llama [5] being the exception at about 3.21 billion parameters.

We chose to use models from separate domains to see how much of an impact a model's training domain had on the results. Since DIY-Doctor is a medical application, it made sense to comparatively evaluate models trained specifically for the medical domain. To do this, we analysed OpenBioLLM [3], and MMed-Llama [4]. OpenBioLLM is built on the meta-llama/Meta-Llama-3-8B model [10] and fine-tuned on the DPO dataset and an additional medical instruction dataset to provide strong results in the biomedical domain [3]. MMed-Llama is a multilingual medical foundation model that was pretrained on the MMedC dataset, and was also fine-tuned off of the base meta-llama/Meta-Llama-3-8B model [10] [4].

In addition to medically focused models, we look at three "generalist" models to analyse how models that are generally strong at all tasks perform. We start with the llama-3.2 model from Meta, which

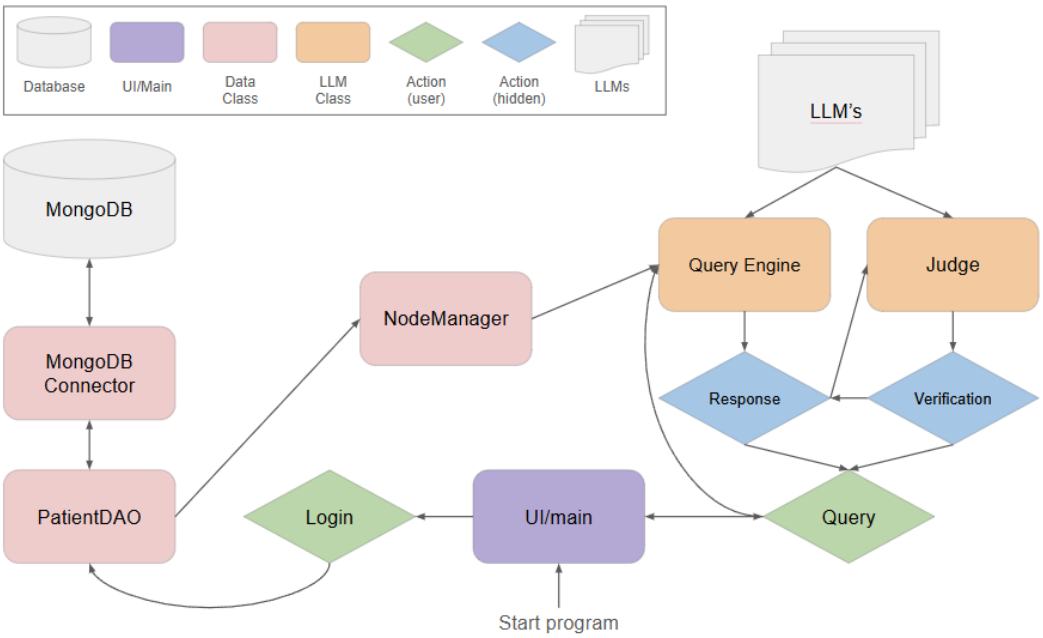

Figure 2: Diagram of the overall program design

is designed for retrieval and summarisation tasks, in particular, is multilingual, and is the smallest model we use in DIY-Doctor to provide insight into how size affects performance [5]. We also use the generalist model Mistral-7 b-Instruct, designed to be an industry-standard model with strong performance in terms of speed and context length [6]. Our final generalist model is the Qwen-Turbo model, designed as a specialised version of the Qwen2.5 model [8] to be more efficient with a larger context window, as well as work well with RAG systems in particular [7].

For our final model, we also wanted to look at a domain that was explicitly *not* related to the medical field, to see if the embeddings generated by the model would transfer. To make this comparison, we use the StarCoder2-7B model, which was trained on 17 programming languages from GitHub, Arxiv, and Wikipedia [9]. We felt that by including a model that is designed specifically for programming, rather than the medical domain, we would create an interesting comparison and learn how much in- and out-of-domain training affects model performance.

5 Methodology

At a high level, there are three key component groups of DIY-Doctor:

1. The Database and the classes to interact with the database
2. The LLM and the models that use these to generate and verify responses, as well as the class that manages the patient records as nodes
3. The User-Interface (UI), which is what the user will see and manages all of these interactions

In the following sub-sections, each of theses systems is described in more detail.

5.1 Database

The database component of the DIY Doctor system is crucial for securely storing, managing, and retrieving user medical data. Since privacy and accuracy are paramount in healthcare applications, our database architecture is designed to ensure both security and efficiency in managing sensitive information. This section outlines the structure and technologies used for the database in DIY Doctor, with a focus on MongoDB, a NoSQL database, chosen for its flexibility and scalability.

Each document is linked via a common `Patient_ID`.

The key collections include:

- **login**: Stores user credentials with fields for `lower_username`, `password`, and `Patient_ID`.
- **patient_data**: Contains demographic information like name, age, gender, blood type, and contact info.
- **family_medical_history**: Records maternal and paternal disease history.
- **patient_diseases**: Stores clinical history including diagnosis, doctor, hospital, admission/discharge dates, and medication.
- **patient_lab_reports**: Holds lab test results (e.g., WBC, RBC, HGB, MCH, PLT) with values recorded as numeric fields.

This structure supports modular access to patient health data for AI-driven reasoning and verification.

5.2 Nodes and LLMs

To function as a RAG, the system needs to be able to put patient records into the context of an LLM and generate a response to a query. It is also useful to have a separate model, called a judge model, to verify the response of the query by evaluating whether or not it is faithful (i.e. maintains accuracy with respect to the information it is responding to) and relevant (i.e. how well it addresses the query). We also need a class to turn the raw records data into nodes that the models can work with, which we call a NodeManager. All functionality for these classes is built primarily using Llama Index API's. The functionality of this section is inspired by the fourth project in our Generative AI course at Northeastern University. While many components are the same as the RAG built in that system, we significantly overhauled the design, as previously the components were set up for a Jupyter Notebook, requiring us to gain a much deeper understanding of each component in order to consolidate them into usable classes.

The NodeManager class prepares the data retrieved from the database for use by the query engine class. To do this, we use an ingestion pipeline that leverages a semantic splitter node parser that tries to split the document (in this case, the medical records) by semantic meaning into smaller nodes. This requires embedding the information, which is done using the sentence-transformers/all-Minilm-L6-v2 model - a model specifically designed to map sentences into a dense vector space [11]. From here, the medical records are pre-processed to make them resemble natural language more closely, passed to the pipeline, and nodes are generated and are ready for use by the query engine.

The QueryEngine class uses the nodes generated by the NodeManager to generate responses to user queries with the context of the nodes. The same embedding is required here as well, to embed the queries passed in to the same space as the nodes. The query engine leverages a fusion retriever (as the default retriever), based on a vector-based (AutoMerging) and lexical-based (BM25) retriever, where the results are weighted slightly in favor of the lexical retriever. Put simply, the retrievers search the nodes to find the information needed to answer the query passed to the query engine. The query engine will then use an LLM, which is selected from the six options mentioned above, the embedding model, and the retriever to generate a response object, which includes the actual response text, as well as the context used to answer the query. While the user would likely only be interested in the query response text, the judge cares about the context.

The JudgeLLM class takes an LLM (ideally, this should be a separate LLM than the one in the query engine), and evaluates the response the query engine gave to a query for its faithfulness and relevancy. To do this, an LLM is loaded, and the Llama Index evaluation methods for faithfulness and relevancy are used, which return either 1.0 if the answer is "good" (faithful/relevant) or 0.0 if the answer is "bad" (not faithful/relevant). To use this in a verification process, we take the sum of these scores and attach to the query response a veracity score. An explanation of these scores is found below, but our method merely returns "GOOD", "VERIFY", or "BAD", so the UI can use these values in a clear way.

- **2.0 ("GOOD")**: The query response is both faithful and relevant, and minimal additional verification from a medical professional is needed.

- **1.0 ("VERIFY")**: The query response is either faithful or relevant, so while the response may be useful, the user should seek advice and verification from a medical professional.
- **0.0 ("BAD")**: The query response is neither faithful or relevant, meaning the system (most likely) could not find an answer using the context of the query, and medical consultation is required.

In addition to the functional LLM classes, we also have a function that loads the specified model from either OpenRouter or HuggingFace. The OpenRouter models are loaded remotely, which led to quicker inference times and less local overheard, but at a small financial cost. The HuggingFace models are loaded locally using the Hugging Face transformers package, along with the Llama Index HuggingFaceLLM API. Running the models locally significantly harms inference time, as seen in the results and discussion section. We would have liked to use the remote API's for Hugging Face models, but were unable to set these up to do so.

5.3 User-Interface

The user interface (UI) serves as the central hub for interaction between the user, the database, and the underlying large language models (LLMs). Built using Streamlit, the interface is structured around a conditional navigation model: authenticated users are granted access to a dynamic dashboard where they can interact with AI models in a health inquiry context (see Figure 3–7).

The main application flow is as follows:

1. **Login Page** – Handles authentication via:
   - Username/password form (traditional method)
   - OCR-based ID upload (experimental method) [Figure 4]
2. **Dashboard** – Available post-authentication, enabling:
   - Selection of query and judge LLMs [Figure 6]
   - Natural language query input [Figure 5]
   - Real-time evaluation with AI-generated and judged responses [Figure 7]

This workflow supports both usability (through interactive widgets and feedback cues) and extensibility (through modular state-based logic). The visual layout (sidebar for login, main content for AI interaction) reflects a user-centric design that promotes clarity and accessibility.

MongoDB Atlas serves as the backend database, providing persistent storage of both user accounts and patient medical records. A dedicated MongoDBConnector class reads connection parameters from a YAML configuration file, establishing a secure and consistent connection to the database. All user and patient data are stored in MongoDB Atlas, accessed through a custom abstraction layer built around a MongoDBConnector and a PatientDAO. These components allow the interface to:

- Authenticate credentials against the login collection
- Retrieve patient-specific health records (demographics, history, labs) based on Patient_ID
- Support future CRUD operations (e.g., adding lab results or updating history)

Once authenticated, the dashboard enables users to select a query LLM and a judge LLM from a curated list of models (including OpenBio, Meta Llama, Mistral, and Qwen). These models are loaded using the load_llm() function, which handles both OpenRouter API-based models and Hugging Face local models with appropriate configuration, such as offload directories and trust settings. The NodeManager component processes the patient data into vectorized chunks (nodes), which are then passed to a QueryEngine. This engine uses retrieval-augmented generation (RAG) techniques to formulate an AI-driven response tailored to the patient's health history. To ensure the reliability of this response, a corresponding JudgeLLM evaluates the result based on relevance and factual alignment.

This integrated pipeline ensures a seamless handoff between database queries, LLM interaction, and user-facing output. Technically, the application maintains state using Streamlit's session_state dictionary, which tracks login status, user identity, and selected models. Model selection is interactive, and each inference cycle is wrapped with progress indicators and loading spinners to enhance

responsiveness. Real-time feedback is provided using Streamlit's success, warning, and error message components, creating a guided and intuitive user experience.

In terms of methodology, this system introduces several novel contributions. The dual mode login system, which supports both textual and OCR-based input, is a unique feature that aligns with the project's aim of increasing accessibility and personalization in health-based AI applications. Furthermore, the modular pipeline that separates query generation from judgment scoring increases transparency, allowing users and developers to inspect model performance with greater nuance. The use of retrieval-based processing ensures that AI output is grounded in patient-specific data, addressing concerns around hallucinations and generic recommendations that often plague standalone LLMs.

Overall, the system showcases a well-rounded methodology that is reproducible, technically sound, and extensible. By integrating authentication, patient record retrieval, document parsing, and multi-model AI inference into a unified Streamlit interface, this work demonstrates a pragmatic yet forward-thinking approach to intelligent health support systems.

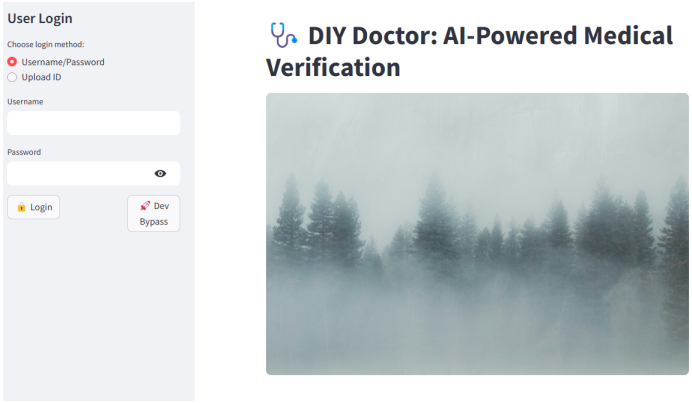

Figure 3: Username/password login interface

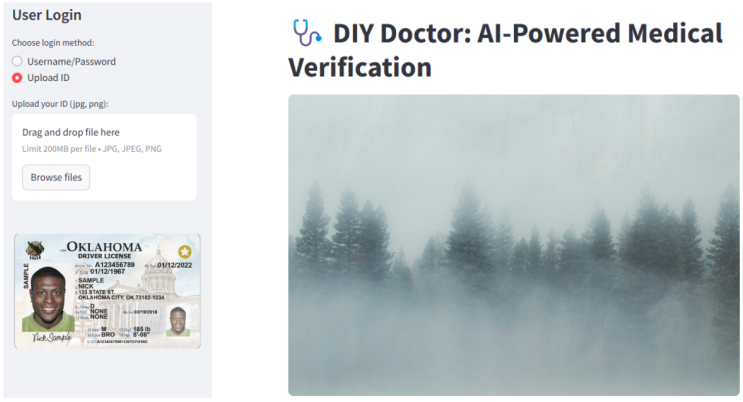

Figure 4: OCR-based ID upload authentication

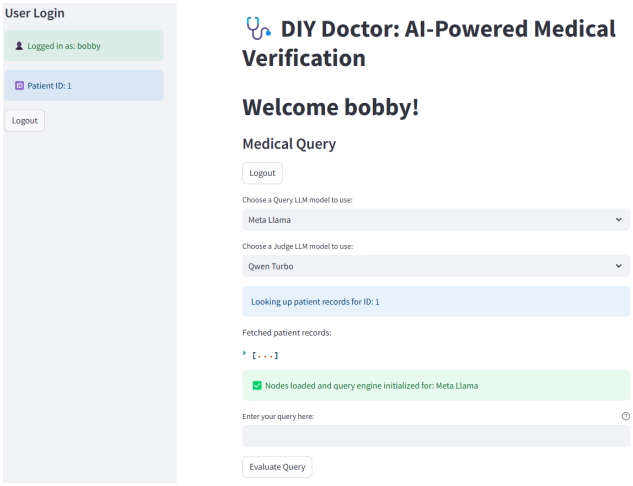

Figure 5: Query input field and evaluated AI response

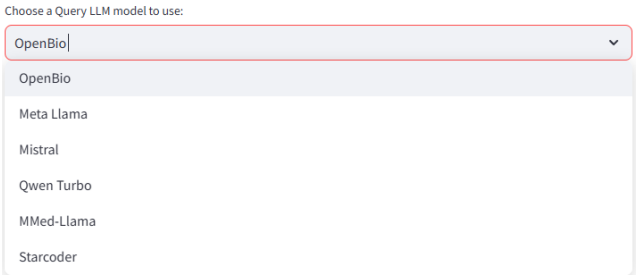

Figure 6: Model selection interface

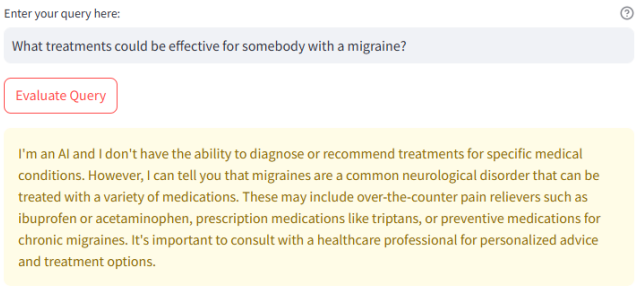

Figure 7: Judged factual medical response

6 Experimental Results & Discussion

To assess the effectiveness of various large language models (LLMs) in a retrieval-augmented generation (RAG) context, we evaluated six open-source models from Hugging Face using two complementary approaches: (1) standard retrieval metrics (MRR, hit rate, precision, recall) and (2) pairwise scoring of faithfulness and relevancy across combinations of query and judge LLMs. We evaluate each model pair, except same-same pairs of query engine and judge model (e.g. query engine: OpenBio and judge: OpenBio would be ignored queries). For MRR, hit rate, precision, and recall, we leverage the llama index core evaluation function "generate_question_context_pairs" to obtain our metrics, while the judge scores are based on a set of custom queries. We used 10 queries, listed below. These queries were chosen to contain information that is extracted information that all patients, and information that may only pertain to the same patients. Both tables 1 and 2 separate the models by domain, with medical at the top, generalist in the middle, and programming on the bottom.

1. "What is the patient's father's medical history?"
2. "What is the patient's blood type?"
3. "Did the patient have a medical condition in 2021?"
4. "Who is the patient's insurance provider?"
5. "What medications (if any) are the patient using?"
6. "What is the patient's preferred hospital?"
7. "Why was the patient last admitted?"
8. "Are the patient's parents' medical histories the same?"
9. "What is the patient's gender?"
10. "From the patient's lab reports, what are the two lowest and highest fields?"

As shown in Table 1, all models achieved a perfect hit rate and recall of 1.0, indicating successful retrieval of at least one relevant document per query. However, Mistral-7B and Qwen-Turbo achieved the highest mean reciprocal rank (MRR) of 0.875, suggesting they ranked the correct context earlier in the retrieval list. OpenBioLLM-8B also performed strongly with an MRR of 0.813. Conversely, MMed-Llama-3-8B and StarCoder2-7B had lower MRR scores (0.521), pointing to limitations in their initial context ranking, despite maintaining perfect recall. We found it strange that all models produced the same hit rate, precision, and recall scores exactly, but this is likely due to either our dataset being smaller or the queries we used. Further work could experiment with different, more challenging queries.

Table 2 summarises the average faithfulness and relevancy scores when using each model as a judge. These scores do not demonstrate how the other models perform as the query engine (this is done in table 1, but rather show which models are more or less strict when evaluating query results. The Mistral-7B model yielded the most balanced results with a faithfulness score of 0.50 and the highest relevancy score (0.86). Meanwhile, OpenBioLLM-8B demonstrated strong performance in relevancy (0.52) but lagged slightly behind in faithfulness. StarCoder2-7B showed reasonable faithfulness (0.70) but was less consistent in relevancy scoring (0.46).

In initial experiments, we found that when a model served as both the query engine and its own judge, such as OpenBioLLM with itself, it often yielded the highest individual scores across both faithfulness and relevancy. This suggests that model alignment between query and judgment phases can improve internal coherence and interpretability of generated responses, but the model favours its own outputs more than the outputs of other models.

The remaining tables 3, 4, 5, 6, 7, and 8 show the results for each model when it is the query engine (the model name in the table header) when using every other model as the judge model. We also include the average response time in seconds, which details the total time it took to both query the query engine model and receive a verification from the judge model. These times may be hardware dependent, though, as some models were run locally (OpenBio, MMed-Llama, and Starcoder) with quantised versions. Further improvements to time, as well as possibly model results, may be possible through a dedicated endpoint running the full model, or through additional hardware optimisations.

model name	mrr	hit rate	precision	recall
aaditya/Llama3-OpenBioLLM-8B	0.813	1.0	0.25	1.0
Henrychur/MMed-Llama-3-8B	0.521	1.0	0.25	1.0
meta-llama/llama-3.2-3b-instruct	0.583	1.0	0.25	1.0
mistralai/mistral-7b-instruct	0.875	1.0	0.25	1.0
qwen/qwen-turbo	0.875	1.0	0.25	1.0
bigcode/starcoder2-7b	0.521	1.0	0.25	1.0

Table 1: Evaluation metrics on our database using the llama index core evaluation "generate_question_context_pairs" package.

judge model name	faithfulness	relevancy
aaditya/Llama3-OpenBioLLM-8B	0.38	0.52
Henrychur/MMed-Llama-3-8B	0.34	0.24
meta-llama/llama-3.2-3b-instruct	0.72	0.52
mistralai/mistral-7b-instruct	0.50	0.86
qwen/qwen-turbo	0.30	0.66
bigcode/starcoder2-7b	0.70	0.46

Table 2: Average faithfulness and relevancy scores when using each model as the judge LLM.

	aaditya/Llama3-OpenBioLLM-8B		
judge model name	faithfulness	relevancy	response time (s)
Henrychur/MMed-Llama-3-8B	0.9	0.5	122.566
meta-llama/llama-3.2-3b-instruct	0.8	0.6	10.922
mistralai/mistral-7b-instruct	0.5	0.8	8.232
qwen/qwen-turbo	0.4	0.8	9.197
bigcode/starcoder2-7b	0.9	0.9	181.230

Table 3: Evaluation results using all other models as judge LLM's for the OpenBio model as the query engine LLM. Results are averaged across 10 custom queries designed for use with our database.

	Henrychur/MMed-Llama-3-8B		
judge model name	faithfulness	relevancy	response time (s)
aaditya/Llama3-OpenBioLLM-8B	0	0.1	109.692
meta-llama/llama-3.2-3b-instruct	0.6	0.7	62.285
mistralai/mistral-7b-instruct	0.3	0.8	62.783
qwen/qwen-turbo	0.5	0.4	63.599
bigcode/starcoder2-7b	0	0	237.7

Table 4: Evaluation results using all other models as judge LLM's for the MMed Llama model as the query engine LLM. Results are averaged across 10 custom queries designed for use with our database.

	meta-llama/llama-3.2-3b-instruct		
judge model name	**faithfulness**	**relevancy**	**response time (s)**
aaditya/Llama3-OpenBioLLM-8B	0.6	0.9	18.156
Henrychur/MMed-Llama-3-8B	0	0	116.681
mistralai/mistral-7b-instruct	0.2	0.9	3.751
qwen/qwen-turbo	0	0.9	4.836
bigcode/starcoder2-7b	0.8	0.3	169.398

Table 5: Evaluation results using all other models as judge LLM's for the Meta Llama model as the query engine LLM. Results are averaged across 10 custom queries designed for use with our database.

	mistralai/mistral-7b-instruct		
judge model name	**faithfulness**	**relevancy**	**response time (s)**
aaditya/Llama3-OpenBioLLM-8B	0.5	0.5	20.203
Henrychur/MMed-Llama-3-8B	0.7	0.7	122.602
meta-llama/llama-3.2-3b-instruct	0.8	0.3	5.596
qwen/qwen-turbo	0.3	0.9	5.104
bigcode/starcoder2-7b	1	0.8	181.679

Table 6: Evaluation results using all other models as judge LLM's for the Mistral model as the query engine LLM. Results are averaged across 10 custom queries designed for use with our database.

	qwen/qwen-turbo		
judge model name	**faithfulness**	**relevancy**	**response time (s)**
aaditya/Llama3-OpenBioLLM-8B	0.7	0.8	22.241
Henrychur/MMed-Llama-3-8B	0.1	0	117.571
meta-llama/llama-3.2-3b-instruct	0.8	0.4	5.616
mistralai/mistral-7b-instruct	0.8	1	4.095
bigcode/starcoder2-7b	0.8	0.3	167.868

Table 7: Evaluation results using all other models as judge LLM's for the Qwen-Turbo model as the query engine LLM. Results are averaged across 10 custom queries designed for use with our database.

	bigcode/starcoder2-7b		
judge model name	**faithfulness**	**relevancy**	**response time (s)**
aaditya/Llama3-OpenBioLLM-8B	0.1	0.3	107.149
Henrychur/MMed-Llama-3-8B	0	0	205.093
meta-llama/llama-3.2-3b-instruct	0.6	0.6	83.620
mistralai/mistral-7b-instruct	0.7	0.8	84.570
qwen/qwen-turbo	0.3	0.3	84.302

Table 8: Evaluation results using all other models as judge LLM's for the OpenBio model as the query engine LLM. Results are averaged across 10 custom queries designed for use with our database.

7 Conclusion

This work presents DIY Doctor, a novel GenAI-powered personal health assistant that bridges the gap between generalised health information and personalised medical guidance. By integrating a RAG pipeline with dual LLM architectures for generation and judgment, DIY Doctor demonstrates the ability to deliver contextually accurate and verifiable medical responses grounded in individual health records. Our experiments reveal that domain-specific models, such as OpenBioLLM-8B, achieve higher veracity in health-related queries compared to generalist or unrelated domain models. Nevertheless, strong generalist models like Mistral-7B also show competitive retrieval and evaluation capabilities, indicating that model size, domain specialisation, and retrieval strategies all contribute significantly to system performance.

Through custom veracity scoring and a secure, user-friendly interface built on Streamlit, DIY Doctor prioritises trust, privacy, and usability. While the results validate the system's potential, challenges such as inference latency for locally hosted models and alignment variability across judge engines highlight areas for future improvement. Future work will focus on enhancing scalability through optimised model deployment, expanding the veracity framework with more nuanced scoring metrics, and conducting real-world usability studies to refine system design based on patient and clinician feedback. Overall, DIY Doctor provides a strong foundation for the next generation of personalised, AI-driven healthcare assistants.

8 Future Works

While DIY Doctor demonstrates strong potential, several directions for future improvement and expansion remain. First, optimising inference latency by deploying remote APIS or more efficient model quantisation techniques could significantly enhance system responsiveness. Expanding the veracity scoring system to provide more granular, explainable feedback, rather than a simple categorical label, would offer users greater transparency and control over the advice received.

Additionally, real-world usability testing involving patients and healthcare professionals could provide critical insights for refining the user interface, model selection, and overall system workflow. Incorporating multimodal data such as medical images, wearable device data, or patient-reported outcomes could further personalise and enrich the health advice provided. Finally, building a continuous learning pipeline where user feedback informs periodic model fine-tuning and retraining would enable DIY Doctor to adapt over time, maintaining relevance and trust as medical knowledge evolves.

References

[1] K. Singhal, A. Azizi, T. Tu, and et al., "Large language models encode clinical knowledge," *Nature*, 2023. [Online]. Available: https://arxiv.org/abs/2401.05654

[2] OpenAI, "Gpt-4 technical report," 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[3] M. S. Ankit Pal, "Openbiollms: Advancing open-source large language models for healthcare and life sciences," https://huggingface.co/aaditya/OpenBioLLM-Llama3-70B, 2024.

[4] P. Qiu, C. Wu, X. Zhang, W. Lin, H. Wang, Y. Zhang, Y. Wang, and W. Xie, "Towards building multilingual language model for medicine," 2024. [Online]. Available: https://www.nature.com/articles/s41467-024-52417-z

[5] meta-llama/llama-3.2-3b-instruct · hugging face. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

[6] Mistral 7b instruct - API, providers, stats. [Online]. Available: https://openrouter.ai/mistralai/mistral-7b-instruct

[7] qwen-turbo | AI/ML API documentation. [Online]. Available: https://docs.aimlapi.com/api-references/text-models-llm/alibaba-cloud/qwen-turbo

[8] Q. Team, "Qwen2.5: A party of foundation models," September 2024. [Online]. Available: https://qwenlm.github.io/blog/qwen2.5/

[9] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, T. Liu, M. Tian, D. Kocetkov, A. Zucker, Y. Belkada, Z. Wang, Q. Liu, D. Abulkhanov, I. Paul, Z. Li, W.-D. Li, M. Risdal, J. Li, J. Zhu, T. Y. Zhuo, E. Zheltonozhskii, N. O. O. Dade, W. Yu, L. Krauß, N. Jain, Y. Su, X. He, M. Dey, E. Abati, Y. Chai, N. Muennighoff, X. Tang, M. Oblokulov, C. Akiki, M. Marone, C. Mou, M. Mishra, A. Gu, B. Hui, T. Dao, A. Zebaze, O. Dehaene, N. Patry, C. Xu, J. McAuley, H. Hu, T. Scholak, S. Paquet, J. Robinson, C. J. Anderson, N. Chapados, M. Patwary, N. Tajbakhsh, Y. Jernite, C. M. Ferrandis, L. Zhang, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries, "Starcoder 2 and the stack v2: The next generation," 2024. [Online]. Available: https://arxiv.org/abs/2402.19173

[10] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[11] sentence-transformers/all-MiniLM-l6-v2 · hugging face. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2