MINISTRY OF EDUCATION AND TRAINING
CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY

GRADUATION THESIS
BACHELOR OF ENGINEERING IN INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

APPLYING HYBRID RAG TECHNOLOGY IN A CHATBOT FOR CONSULTING ON SEXUALLY TRANSMITTED DISEASES (STDs)

Student: Le Huynh Dang
Student ID: B2111977
Class: 2021-2025 (K47)
Advisor: Dr. Lam Nhut Khang

Can Tho, 12/2025

MINISTRY OF EDUCATION AND TRAINING
CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

GRADUATION THESIS
BACHELOR OF ENGINEERING IN INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

APPLYING HYBRID RAG TECHNOLOGY IN A CHATBOT FOR CONSULTING ON SEXUALLY TRANSMITTED DISEASES (STDs)

Student: Le Huynh Dang
Student ID: B2111977
Class: 2021-2025 (Cohort K47)
Advisor: Dr. Lam Nhut Khang

Can Tho, 12/2025

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my heartfelt gratitude to the College of Information and Communication Technology and its dedicated instructors for their invaluable support and guidance throughout my academic journey. I am especially grateful to Dr. Lam Nhut Khang for providing me with the opportunity to conduct this thesis and for her expert advice and assistance during the development of this thesis.

I would also like to express my deepest appreciation to the professors, colleagues, and friends who have contributed to the success of this thesis. Special thanks to my parents for their unwavering support, encouragement, and guidance, which have helped me appreciate the importance of education and the value of perseverance in life.

To everyone who has supported me, shared their knowledge, or positively influenced my academic and personal growth, I am truly thankful. Your contributions, whether big or small, have made a significant impact on this journey.

Lastly, I would like to extend my best wishes to all those reading this acknowledgment. May you have good health, happiness, and prosperity in all your future endeavors.

Thank you!

Can Tho, ……., 2025

Written by

Le Huynh Dang

ABSTRACT

This thesis develops a text-based medical chatbot that provides accurate, accessible, and confidential advice on sexually transmitted diseases (STDs). The system implements Retrieval-Augmented Generation (RAG) with a hybrid retriever that combines semantic search (FAISS, L2-normalized embeddings, IndexFlatIP) and lexical search (BM25). Candidate passages from both channels are merged with Reciprocal Rank Fusion (RRF), then used to ground the language model's response with clear citations. The design emphasizes safety (non-diagnostic wording, escalation for red-flag symptoms), privacy (minimal data retention, no PHI in logs), and usability (bilingual VI/EN UI, WCAG-aligned basics). Experiments indicate the hybrid RAG pipeline improves robustness on short, layperson queries compared with single-channel retrieval. Note on scope: Graph-structured retrieval (GraphRAG) is not implemented in this build; it is discussed as related work and future extension. The presented system demonstrates that a pragmatic hybrid RAG approach can deliver reliable, transparent guidance for STD information-seeking at low latency and cost.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
STD	Sexually Transmitted Disease
STDs	Sexually Transmitted Diseases
AI	Artificial Intelligence
NLP	Natural Language Processing
LLM	Large Language Model
LLMs	Large Language Models
RNN	Recurrent Neural Network
RAG	Retrieval-Augmented Generation
API	Application Programming Interface
HIPAA	Health Insurance Portability and Accountability Act
GDPR	General Data Protection Regulation
FM	Foundation Model

CHAPTER 1: INTRODUCTION

1. The purpose of the study

The aim of this study is to develop a chatbot application designed to provide accurate, accessible, and confidential STI/STD information. The chatbot seeks to bridge the gap between users and expert health information by delivering reliable, real-time advice through a user-friendly text-based platform, offering a practical solution for sexual health guidance.

2. The problem of the study

Sexual health is often a sensitive and stigmatized topic, especially regarding STDs. Many individuals seek advice but hesitate to consult healthcare professionals due to fears of judgment, concerns over confidentiality, or the lack of easily accessible resources. The central problem is the lack of a confidential, high-quality digital assistant that can answer STD questions in everyday language.

3. Related work

Previous research in the field of digital health has explored various methods for delivering sexual health advice, particularly through mobile apps, websites, and chatbots. For example, platforms like Planned Parenthood's Chatbot and Ada Health have made significant strides in improving access to information related to sexually transmitted diseases (STDs). Despite their usefulness, many of these platforms still face challenges, such as [1]. These systems often provide generic responses that fail to address individual user concerns, which can result in users receiving advice that may not be relevant to their specific situations.

AI-powered solutions that Tom Nadarzynski and co-workers [2] have shown promise in enhancing the accuracy and relevance of health advice. However, these AI-driven solutions often struggle with providing individualized, contextually aware responses, particularly in sensitive areas like sexual health. While some systems rely on text or voice inputs, they often overlook the importance of integrating both textual and visual data [3] to provide a more comprehensive user experience.

Advancements in artificial intelligence, especially in natural language processing (NLP) and reasoning over structured medical data, have demonstrated significant potential in improving the accuracy and personalization of health advice. [4] . However, despite these advancements, few studies have explored the use of these technologies specifically in the context of STD-related queries, and even fewer have combined both textual and visual data effectively [5].

Several studies have explored the integration of real-time information retrieval to enhance chatbot performance. Researchers highlight the potential of AI-powered chatbots in delivering real-time, personalized sexual health advice, yet they also note challenges related to ensuring privacy and confidentiality [6] . Other works further discuss the integration of NLP and AI in healthcare, emphasizing their value in delivering personalized care in sensitive areas such as sexual health [7]. Similarly, some focus on the challenges and applications of NLP in healthcare AI, specifically in personalized health consultations [8].

Incorporating external knowledge into generative AI systems has been a significant area of focus in recent studies. Researchers review how external knowledge integration, including structured medical data, can improve chatbot accuracy in healthcare, which is especially important in contexts like STD advice [9] . Others discuss methods to build trust in AI-powered healthcare systems, a critical factor in user adoption of digital health platforms [10]. Additionally, privacy and user engagement concerns have been addressed, which are essential in improving the effectiveness of AI systems in delivering sexual health advice [11].

A novel approach using Retrieval-Augmented Generation (RAG) has also been presented to improve health chatbot responses. This technique, which combines real-time data retrieval and generative models, has shown promise in offering more accurate, context-aware advice [12] . Further research explores how RAG can be applied specifically in healthcare chatbots, enabling the integration of up-to-date, context-specific information to enhance the quality of STD-related advice [13] . Other works focus on the broader integration of RAG technology in medical AI systems, particularly in improving user interaction and increasing the accuracy of responses in complex healthcare queries [14] , [15] . Both studies highlight how GraphRAG can be used to link knowledge from external databases, improving the AI's ability to respond to intricate medical questions, such as those related to sexually transmitted diseases.

4. Thesis and general approach

I build a text-based chatbot that delivers accurate and confidential STD advice using hybrid RAG. User questions are answered by a language model that is constrained to retrieved evidence: I run BM25 and FAISS in parallel, fuse their rankings with RRF, and pass only the top-K grounded snippets into the prompt. The system focuses on safety (non-diagnostic language, escalation for red-flags), privacy (data minimization), and practical operations (fast startup, low latency). Graph-based retrieval is out of scope for the implemented system and appears only in related work/future work.

5. The criteria for study's success

The success of this study will be measured against several key criteria that focus on the chatbot's performance, its ability to meet user needs, and its overall impact on providing reliable sexual health advice for STDs. These criteria include:

1. **Accuracy of responses**: The chatbot's ability to provide medically accurate and up-to-date information related to STDs will be a primary measure of success. The responses will be evaluated based on their alignment with current medical guidelines and best practices for sexual health.
2. **Personalization of advice**: A successful chatbot should offer personalized responses tailored to the user's input and context. The ability to process individual queries and provide advice specific to the user's situation (such as symptoms, concerns, or preferences) will be a key indicator of the study's success.
3. **User accessibility and engagement**: The platform must be easy to use and accessible, allowing users to interact with the chatbot in a straightforward

manner. A successful study will show high user engagement, measured by the frequency of use and the overall satisfaction of the users with the interaction.

4. **Confidentiality and privacy protection**: Ensuring user confidentiality is critical, especially when dealing with sensitive topics like sexual health. The chatbot's ability to protect user data and maintain privacy standards in compliance with legal and ethical guidelines will be an essential success criterion.
5. **Effectiveness in providing STD advice**: The chatbot's effectiveness will be measured by its ability to provide helpful, clear, and actionable advice regarding STD prevention, testing, and treatment options. Success will be based on whether users find the advice useful and feel more informed after interacting with the system.
6. **Scalability and adaptability**: The chatbot should be scalable, allowing it to serve a large number of users without compromising performance. Additionally, the ability to adapt to evolving medical knowledge, user feedback, and new developments in AI technology will be important for the long-term success of the application.
7. **User satisfaction and feedback**: Finally, user feedback will serve as an essential criterion for success. Surveys, interviews, or feedback forms will assess users' satisfaction with the chatbot's performance, as well as its impact on their understanding and decision-making regarding sexual health.

These criteria together will ensure that the study not only provides a functional chatbot but also one that offers significant value in improving access to accurate, personalized, and confidential STD-related advice.

6. Outline of the thesis

This thesis focuses on the development of a medical chatbot leveraging large language models, with particular emphasis on the Retrieval-Augmented Generation (RAG) system. The structure of the thesis is as follows:

Chapter 1. Introduces the research background, objectives, and significance of the study

Chapter 2. Presents the theoretical foundations, key concepts, and a review of related literature

Chapter 3. Outlines the methodology and the detailed process of building the chatbot model

Chapter 4. Discusses the implementation, including technical details, and presents the results of the system evaluation

Chapter 5. Concludes the thesis by summarizing the findings and suggesting directions for future research and improvements

CHAPTER 2: THEORETICAL BASIS

1. Introduction to digital health and chatbots

Digital health is the integration of technology into healthcare to improve access to services, enhance the quality of care, and promote health education. The use of digital platforms, including mobile applications, websites, and chatbots, has grown significantly in recent years, providing a new avenue for users to access health information conveniently and confidentially. Chatbots, specifically, are artificial intelligence (AI)-powered tools designed to simulate conversation with users and offer immediate, real-time responses. In the context of sexual health, chatbots have the potential to break down barriers like stigma, embarrassment, and confidentiality concerns, making them valuable tools for providing sexual health advice, including guidance on sexually transmitted diseases (STDs).

Figure 1. Digital Health

Source: https://personalizemymedicine.com/medical-innovation/directories-medical-innovation/digital-health/

2. Chatbots in healthcare

The use of chatbots in healthcare has expanded across various fields, including mental health, general wellness, and sexual health. Chatbots in healthcare utilize natural language processing (NLP) and machine learning algorithms to process user inputs and generate appropriate responses. For instance, platforms like Ada Health and Planned Parenthood's Chatbot have aimed to increase the accessibility of health-related information, providing users with timely advice. In particular, AI chatbots that focus on sexual health can help users learn about STDs, preventive measures, testing options, and treatment plans, offering guidance that may be difficult or uncomfortable for individuals to discuss with healthcare professionals in person.

However, many existing chatbots face challenges regarding user engagement and providing accurate, personalized advice. Generic or one-size-fits-all responses often fail to address the specific needs of individuals, leading to a lack of trust in the system. Additionally, ensuring confidentiality is crucial when dealing with sensitive topics like sexual health, and current systems do not always meet the privacy standards necessary for such advice.

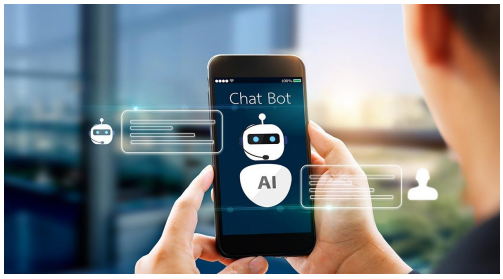

Figure 2. Chatbot

Source: https://pg-p.ctme.caltech.edu/blog/ai-ml/what-is-ai-chatbot-how-do-they-work

3. LLMs

3.1. What are LLMs?

Large Language Models (LLMs) are AI models trained on massive datasets of text and code to understand, generate, and manipulate human language. They power various applications like chatbots, content creation tools, and code generation systems. LLMs are a subset of deep learning models that utilize neural networks with multiple layers to analyze data and learn complex patterns.

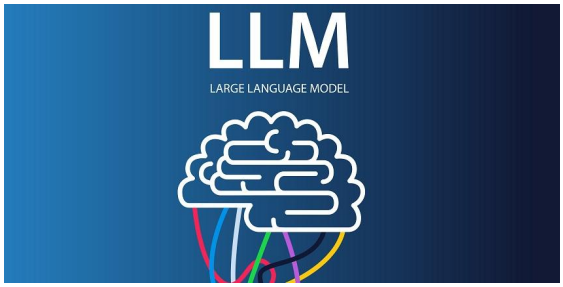

Figure 3. What is LLM?

Source: https://www.uctoday.com/unified-communications/what-is-a-large-language-model-defining-llms/

3.2. The importance of LLMs

LLMs support many downstream tasks, including:

- **Text generation:** LLMs can produce diverse forms of text, from articles and stories to poems and essays, making them versatile tools for content creation.
- **Code generation:** They are capable of generating code in multiple programming languages, assisting developers in creating software solutions efficiently.
- **Translation:** LLMs can seamlessly translate text between different languages, bridging communication gaps and enabling global interactions.
- **Summarization:** These models can condense lengthy documents into concise summaries, making it easier to digest complex information.
- **Question answering:** LLMs can respond to questions by leveraging the knowledge they've accumulated during training, offering insights and solutions based on a vast database of information.
- **Creative content generation:** Whether it's scripts, marketing copy, or other creative content, LLMs excel at generating original and engaging material.

- **Chatbots and conversational AI:** As the backbone of many chatbot systems, LLMs enable natural, fluid conversations, transforming how businesses and users interact with AI.

3.3. How it works

Large Language Models (LLMs) are advanced AI systems that learn patterns in language through extensive training, enabling them to generate coherent, contextually relevant responses to user inputs across a wide range of tasks.

Training:

LLMs are trained on massive datasets using self-supervised learning. During training, the model identifies patterns, relationships, and structures in the language by processing large amounts of text data. This allows the model to learn grammar, context, sentence structure, and even reasoning, enabling it to generate text that is contextually appropriate and coherent.

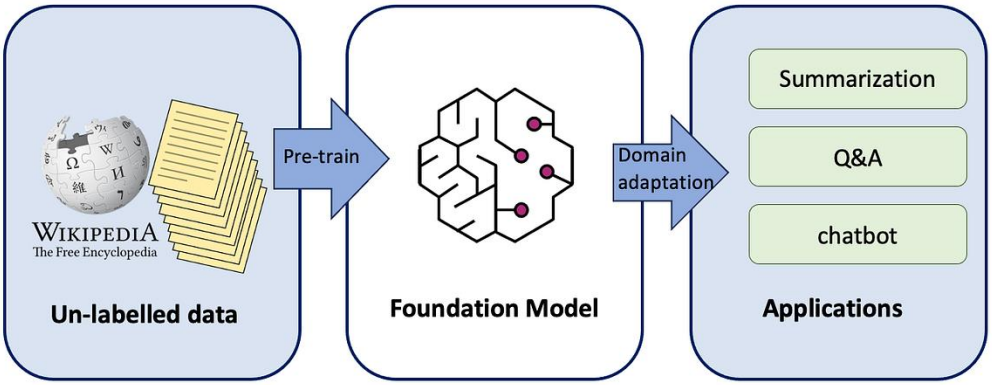

Figure 4. Training LLMs

Source: https://ai.gopubby.com/the-training-pipeline-of-large-language-models-afd5fa57df46

Input and Output:

Users interact with LLMs through natural language prompts, such as questions or instructions. The model processes these inputs and generates corresponding outputs, which could be text, code, or other responses. These outputs are based on the relationships and patterns the model has learned during its training.

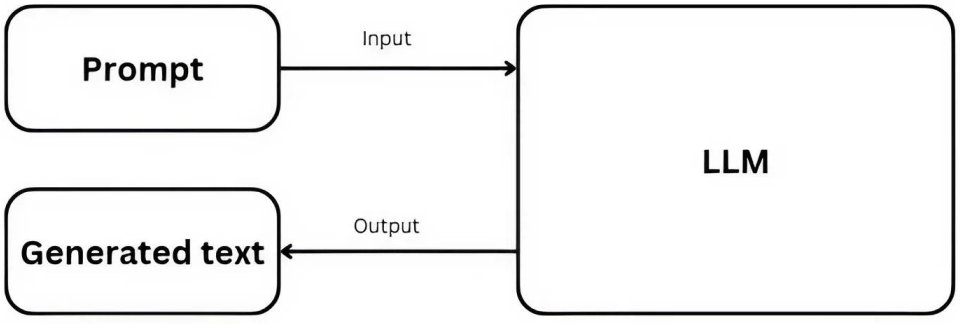

Figure 5. Input and output of LLMs model

Source: https://ubiai.tools/prompt-engineering-how-to-talk-to-llms-like-a-pro/

Contextual understanding:

A key strength of LLMs is their ability to understand context, grammar, and language nuances. This allows them to generate responses that are not only grammatically correct but also contextually relevant. LLMs can maintain coherence across longer conversations or text sequences, adjusting to the context for more accurate and insightful responses.

Parallel processing:

Unlike older models, such as recurrent neural networks (RNNs), which process data sequentially, LLMs can process entire sequences of text in parallel. This parallel processing significantly speeds up training, allowing for the efficient handling of large datasets. According to Amazon Web Services, this capability makes LLMs faster to train and more effective in generating real-time responses.

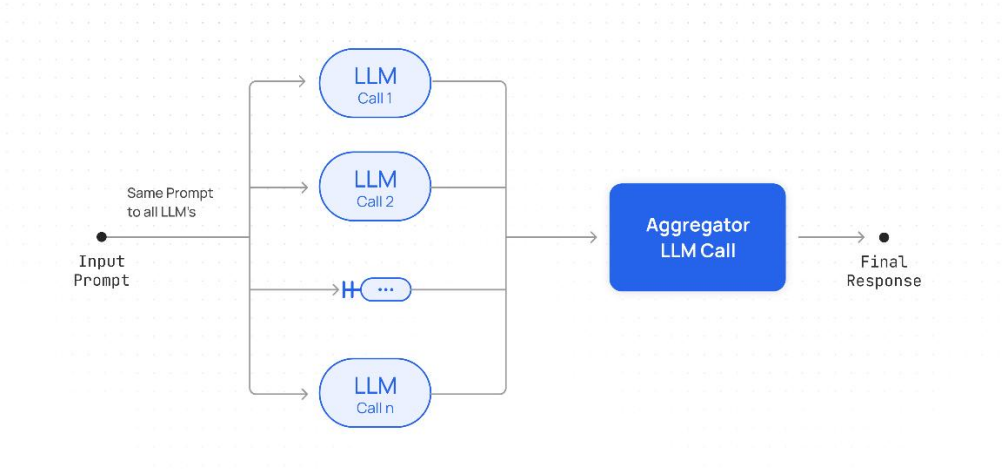

Figure 6. Parallel processing

Source: https://docs.together.ai/docs/parallel-workflows?utm_source=chatgpt.com

3.4. Some popular LLMs

There are some popular LLMs that have made significant advancements in natural language processing and AI applications:

- **GPT-3 (Generative Pretrained Transformer 3)**
  Developed by OpenAI, GPT-3 is one of the most widely known and used LLMs. It has 175 billion parameters and is capable of performing a wide range of language tasks, including text generation, translation, summarization, question-answering, and more.
- **GPT-4**
  Also developed by OpenAI, GPT-4 is an advanced version of GPT-3. It offers improved reasoning, better contextual understanding, and more accurate responses. It can handle more complex tasks and has a better understanding of nuanced instructions.
- **BERT (Bidirectional Encoder Representations from Transformers)**
  Developed by Google, BERT focuses on understanding the context of words

in a sentence by processing text bidirectionally (considering both left and right context). It's widely used for tasks like question-answering and language understanding.

- **T5 (Text-to-Text Transfer Transformer)**
  T5, also by Google, frames all NLP tasks as text-to-text problems, which makes it highly versatile. Tasks such as summarization, translation, and question-answering are all treated as generating text outputs based on text inputs.
- **RoBERTa (Robustly optimized BERT approach)**
  A variant of BERT, RoBERTa is optimized for better performance by using larger training data and more training steps. It achieves state-of-the-art performance in several NLP tasks, including sentiment analysis and text classification.
- **XLNet**
  XLNet is another model developed by Google/CMU that extends BERT by incorporating autoregressive capabilities, allowing it to capture better dependencies between words. It outperforms BERT on a number of NLP benchmarks.
- **LaMDA (Language Model for Dialogue Applications)**
  Another model from Google, LaMDA is designed specifically for conversational AI. It focuses on making conversations more natural and contextually aware, with a goal to handle open-ended discussions effectively.
- **BLOOM**
  BLOOM (BigScience Large Open-science Open-access Multilingual Language Model) is a large-scale multilingual language model developed as part of a collaborative, open-access project. It supports 46 languages and aims to promote transparency and inclusivity in AI research.
- **Megatron**
  Developed by NVIDIA, Megatron is a highly scalable transformer model that has been used for tasks like text generation and language modeling. It can be trained on massive datasets and utilized for various language-related applications.
- **ChatGPT**
  Built on the GPT architecture, ChatGPT is optimized specifically for conversational applications. It is fine-tuned to generate human-like, context-aware dialogue, making it ideal for chatbots and interactive AI systems.

4. Retrieval-Augmented Generation (RAG).

4.1. What is RAG?

Retrieval-Augmented Generation (RAG) is the process of optimizing the output of a large language model, so it references an authoritative knowledge base outside of its training data sources before generating a response. Large Language Models (LLMs) are trained on vast volumes of data and use billions of parameters to generate original output for tasks like answering questions, translating languages, and completing sentences. RAG extends the already powerful capabilities of LLMs

to specific domains or an organization's internal knowledge base, all without the need to retrain the model. It is a cost-effective approach to improving LLM output so it remains relevant, accurate, and useful in various contexts.

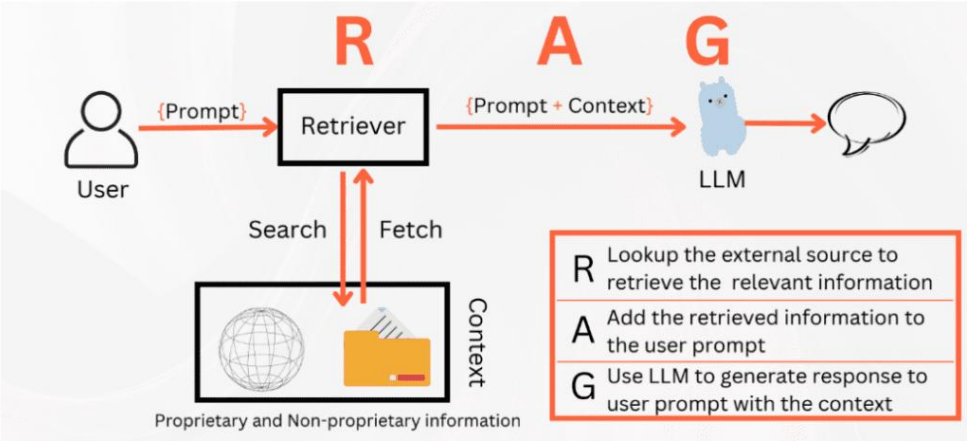

Figure 7. RAG

Source: https://nayakpplaban.medium.com/fine-tune-rag-with-llama2-using-gradient-ai-on-medical-reasoning-task-395bca26a551

4.2. The importance of RAG

Benefits of RAG for Organizational Generative AI Efforts

Cost-Effective Implementation

Chatbot development typically begins with the use of baseline models (FMs), which are large language models (LLMs) that are accessible via APIs and trained on large amounts of general data. However, retraining these models with organization- or industry-specific data can be computationally and financially expensive. RAGs are a more cost-effective way to update an LLM with new data without having to retrain the entire model. This makes generative AI more accessible and feasible for many organizations.

Update Current Information

While the initial training data for an LLM may be adequate, maintaining the information is a challenge. RAG technology allows developers to connect LLM to online data sources such as social media, news sites, or other frequently updated information sources. This allows LLM to provide users with the most up-to-date information.

Increasing user trust

RAG helps LLM provide accurate information with full references. The returned results can include citations or references to source documents, allowing users to double-check the information if additional details or clarification are needed. This increases user trust and confidence in the generative AI solution.

Increasing developer control

With RAG, developers can control and tailor the information sources used by LLM, making it easier to adapt to new requirements and scenarios. They can also restrict access to sensitive information according to different access levels, ensuring that LLM only generates appropriate responses. Additionally, developers can easily fix issues if LLM references incorrect information sources, which helps

organizations deploy generative AI technology more confidently across a variety of applications.

4.3. How it works

Without RAG, LLM generates responses based on existing knowledge from the training data. In contrast, with RAG, an information retrieval system is integrated into the process. This system uses user input to retrieve relevant data from external sources. Both the user query and the retrieved data are provided to the LLM, which then combines this new information with its previously trained knowledge to generate more accurate and informative responses. The following steps outline how the process works:

Generate external data

External data is information that is outside of the LLM's original training data. It can be retrieved from a variety of sources, including APIs, databases, or document repositories. This data can be in a variety of formats, such as files, records, or long text. Another technique, called embedding, converts data into numeric form and stores it in a vector database. This creates a knowledge base that generative AI models can interpret and use.

Retrieving relevant information

The next step involves performing relevance search. The user's query is converted into a vector and compared to the data in the vector database. For example, in an intelligent chatbot designed for human resources, if an employee asks "How many days of annual leave do I have?", the system retrieves relevant documents, such as the company's annual leave policy and the employee's leave history. These specific documents are returned because they are deemed to be the most relevant to the query, based on vector-based similarity calculations.

Improving LLM prompts

After retrieving relevant data, the RAG model improves the user's input by adding relevant information to the prompt. This process involves designing prompts to ensure that data is communicated effectively to the LLM. Enhanced prompts allow the LLM to generate more accurate and contextual responses.

Updating external data

A common concern in maintaining the performance of RAG-based systems. To ensure that retrieved information remains current, documents and their embedded representations should be updated regularly. This can be achieved through either real-time automated updates or periodic batch updates. This issue is similar to those in data analytics, where various data management strategies are employed to handle updates and changes efficiently.

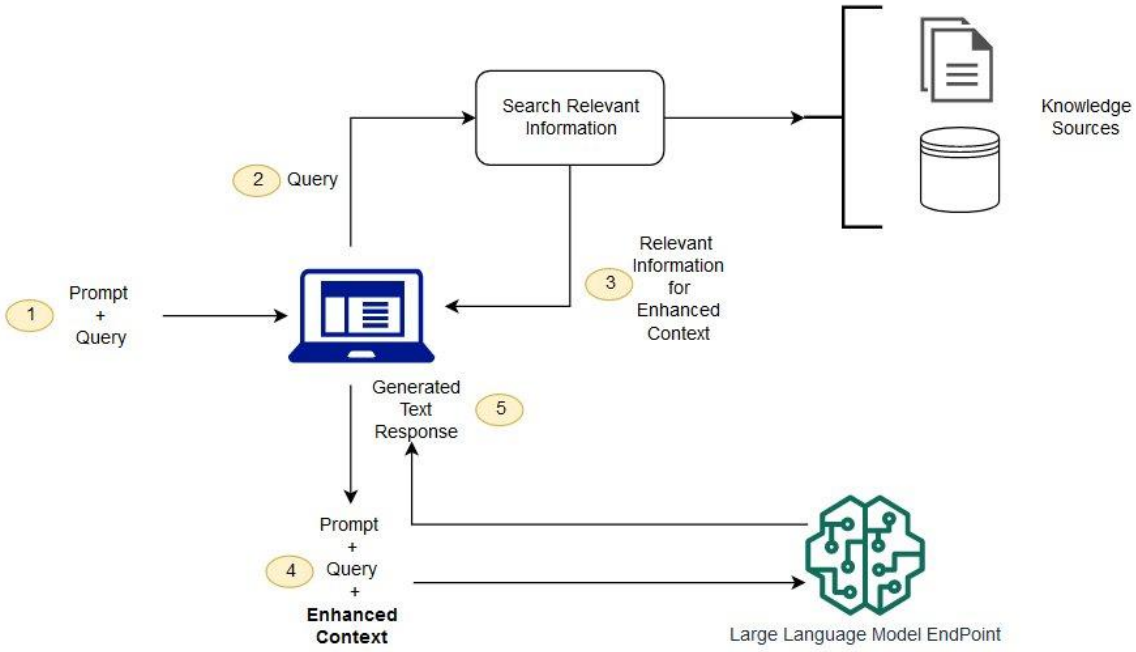

Figure 8. The conceptual flow of using RAG with LLMs.
Source: https://aws.amazon.com/what-is/retrieval-augmented-generation/

Reasoning over medical data refers to using structured medical knowledge to make informed decisions or provide advice. In a medical chatbot, this involves analyzing and interpreting user input (such as symptoms or personal history) and mapping it to appropriate medical information or guidelines. For instance, when a user describes certain symptoms, the chatbot may apply rules from medical databases, diagnostic criteria, or clinical knowledge to deliver personalized advice.

Retrieval-Augmented Generation (RAG) enhances this reasoning process by combining two distinct approaches: information retrieval and text generation. Within a chatbot, RAG first retrieves relevant information from a knowledge base or database, and then employs text generation techniques to produce responses that are coherent, accurate, and contextually appropriate. This integration significantly improves the chatbot's ability to provide personalized and reliable responses, especially in handling complex medical inquiries such as those related to STDs.

4.4. Graph-based RAG and medical knowledge graphs

Recent work extends basic RAG by exploiting the structure of domain knowledge. Instead of indexing only independent passages, GraphRAG represents entities (such as diseases, symptoms, drugs and laboratory tests) and their relations as a graph. At query time, the system can retrieve not only single documents but also multi-hop paths that connect relevant concepts, for example linking an STI to its typical symptoms and recommended tests. In the medical domain, similar ideas appear in clinical knowledge graphs that integrate guidelines, drug–drug interactions and epidemiological data. Combining such graphs with a language model can, in principle, reduce hallucinations and make explanations more transparent, because the model can cite the nodes and edges that support its answer.

However, graph-based RAG requires more complex data engineering than the document-level RAG used in this thesis, and therefore remains outside the implemented scope.

4.5. Reinforcement learning from Human Feedback (RLHF)

Another important line of work in modern language models is Reinforcement Learning from Human Feedback (RLHF). In this paradigm, human annotators compare alternative model outputs and provide preference labels. A separate "reward model" is trained on these preferences, and the base language model is then fine-tuned with reinforcement learning so that its answers align better with human expectations. RLHF has been widely used to make general-purpose chatbots more helpful, honest and harmless. In the context of digital health, RLHF can be used to penalise dangerous suggestions, over-confident diagnoses or breaches of privacy, and to reward answers that encourage appropriate testing and consultation with clinicians. Although this thesis does not train any RLHF components, the deployed GPT-4o-mini model already benefits from such alignment at the platform level, which contributes to its safe behaviour when combined with RAG.

5. Hybrid search

5.1. What is hybrid search?

Hybrid search integrates both dense and sparse vector approaches to capitalize on the strengths of each method. Dense vectors are particularly effective at capturing the semantic context of a query, while sparse vectors are more precise in identifying keyword matches. For example, in the query "How to catch an Alaskan Pollock," the dense vector representation can interpret "catch" in the sense of fishing, rather than baseball or illness, whereas the sparse vector approach focuses only on locating the phrase "Alaskan Pollock." This illustrates how hybrid search leverages the complementary benefits of both techniques.

5.2. How hybrid search works?

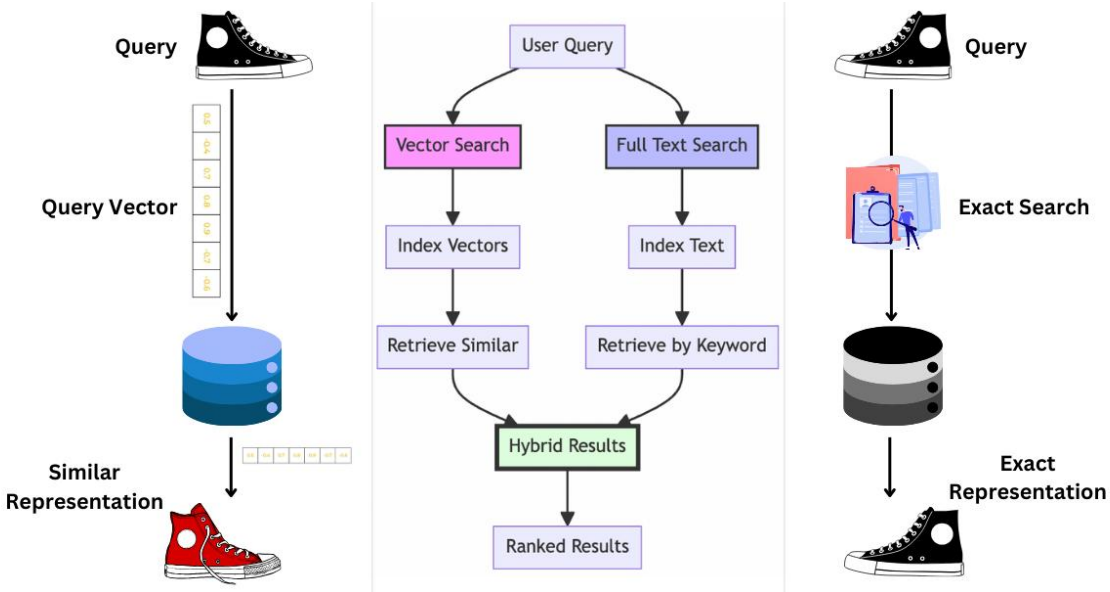

Figure 9. The conceptual flow of using hybrid search.

Source: https://www.kaggle.com/code/mertsengil/baai-bge-m3-hybrid-search-with-langchain

Hybrid search works by simultaneously running traditional keyword-based search (e.g., BM25) and modern vector search (which uses AI to understand semantic meaning) in parallel, then merging the results using an algorithm like Reciprocal Rank Fusion (RRF). This combination provides a more comprehensive and accurate search experience by balancing the precision of exact word matches with the contextual understanding of semantic similarity, leading to better results than either method could achieve alone.

Here's a detailed breakdown of how it works:

Keyword (Lexical) Search
The first component of hybrid search is keyword, or lexical, search. This traditional approach relies on exact term matching between the query and the documents in the collection. Search engines using lexical methods typically employ ranking algorithms such as **Okapi BM25** to determine relevance. BM25 calculates scores based on factors such as the frequency of the query term in each document, the overall document length, and the distribution of terms across the dataset. Although lexical search is efficient and effective for locating documents that contain the precise query words, it often struggles when users employ synonyms, paraphrases, or when the meaning of the query is not explicitly expressed in the text.

Semantic (Vector) Search
The second component is semantic, or vector-based, search. In this method, both queries and documents are transformed into dense vector embeddings using machine learning models such as BERT or other large language models. These embeddings represent not only the literal text but also its semantic meaning, enabling the system to capture context, synonyms, and conceptual relationships. For example, a vector search can identify that the words *"catch"* and *"fish"* are related in the context of fishing, even if the exact word does not appear in the document. This approach overcomes many of the limitations of keyword search by emphasizing meaning rather than word-for-word correspondence.

Parallel Execution
Hybrid search combines these two approaches by running them in **parallel**. When a query is entered, the system simultaneously performs lexical search to identify exact keyword matches and vector search to retrieve semantically similar content. This dual execution produces two separate ranked lists of candidate documents, each reflecting the strengths of its respective method. Running both processes in parallel ensures that the system captures both the precision of exact keyword matching and the flexibility of semantic understanding.

Reciprocal Rank Fusion (RRF)
The challenge lies in effectively merging the results from these two independent searches. Hybrid search typically uses the **Reciprocal Rank Fusion (RRF)** algorithm for this purpose. RRF assigns scores to documents based on their

positions in both ranked lists. Documents that appear near the top in both the keyword and semantic rankings receive higher scores, while those that are strong in only one list are still considered but weighted less. This fusion mechanism ensures that documents highly relevant in terms of both exact wording and contextual meaning rise to the top of the combined ranking.

Final Re-ranking

After the fusion process, the merged list undergoes a **final re-ranking step**. In this stage, the system recalculates scores using a hybrid scoring strategy that balances the contributions of lexical and semantic relevance. The goal is to produce a unified ranking where the most contextually meaningful and lexically accurate results appear first. This re-ranking ensures that end-users receive results that not only match their query keywords but also align closely with the intended meaning, providing a more reliable and user-friendly search experience.

6. Privacy and confidentiality in Digital Health

Privacy and confidentiality are particularly critical when it comes to digital health, as users often share sensitive, personal information. In the case of a chatbot providing STD-related advice, ensuring that users' data is secure and confidential is paramount. The chatbot must be designed in a way that complies with privacy laws such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States or the General Data Protection Regulation (GDPR) in the European Union. This includes ensuring that any personal information, such as health history, symptoms, and test results, is stored securely and not shared without the user's consent.

Effective encryption, anonymization of data, and clear privacy policies are essential for maintaining trust with users. Furthermore, the chatbot must provide transparent communication about how data is used and ensure that users have control over their information. This is particularly important in the context of sexual health, where users may already feel vulnerable or concerned about disclosing personal details.

7. Ethical and explainable AI in healthcare

Beyond privacy, medical chatbots raise broader ethical questions. First, fairness and equity are important: systems trained mainly on English and high-income country data may under-represent local practice in Vietnam or minority user groups. Second, explainability matters because users and clinicians need to understand why a recommendation was made, especially when it concerns testing, medication or vaccination. RAG partially improves transparency by forcing the model to ground its answers in cited passages, but the reasoning steps of the language model are still largely opaque. Finally, responsibility and accountability must be considered: a chatbot should not replace professional medical advice, and its limitations must be communicated clearly. These concerns motivate design choices in this thesis, such as non-diagnostic language, conservative safety prompts and explicit reminders to seek in-person care for high-risk situations.

8. ChatGPT – The Generation Component

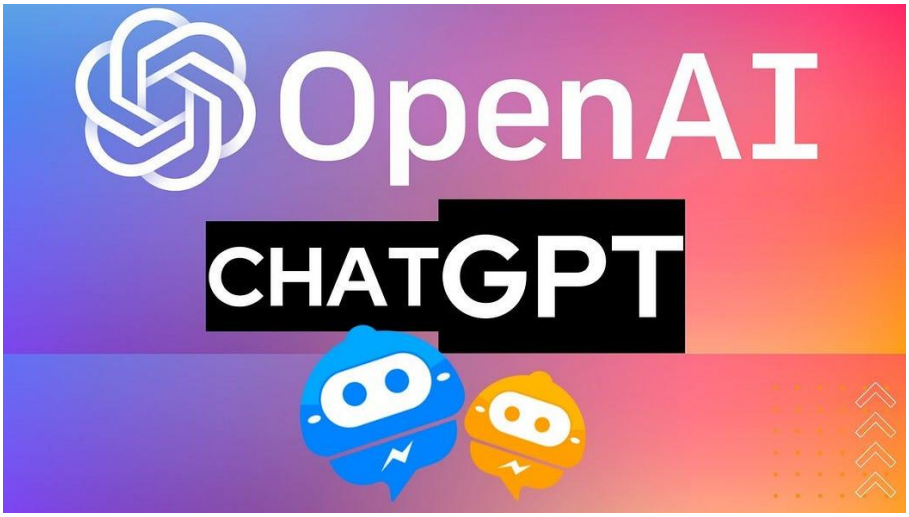

Figure 10. ChatGPT

Source: https://abhiappmobiledeveloper.medium.com/chatgpt-openais-new-artificial-intelligent-chatbot-93af03ea3f2b

In this research, ChatGPT serves as the primary language model responsible for generating answers to user queries. Once relevant documents are retrieved through the hybrid search process, they are supplied to ChatGPT together with the user's original question. The model processes this combined input and produces a coherent, natural-sounding response that is easy to comprehend.

ChatGPT belongs to the GPT (Generative Pre-trained Transformer) family created by OpenAI. Trained on vast collections of textual data, it is capable of generating language that closely resembles human communication. For this project, the system accesses ChatGPT via the OpenAI API, enabling real-time interaction with the model and instant generation of responses.

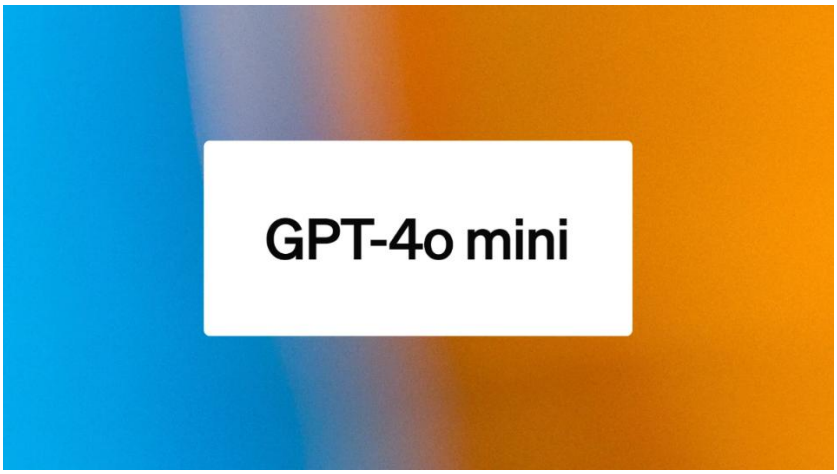

Figure 11. GPT-4o mini model

Source: https://textcortex.com/post/openais-gpt-4o-mini-review

The choice of ChatGPT is motivated by its ability to understand context, integrate information from various sources, and produce responses that are both

fluent and user-friendly. These characteristics make it particularly well-suited for a medical chatbot, where users expect answers that are accurate as well as clear and supportive. In this study, the ChatGPT-4o mini model is employed to process the results retrieved through hybrid search, select the most appropriate answer from the top-k candidates, and refine it into a more natural and conversational response.

Why GPT-4o-mini instead of GPT-5?

Criteria	GPT-4o-mini	GPT-5
Accuracy	Good for general questions but limited in complex reasoning. May hallucinate without clear context.	More accurate and reliable. Stronger multi-step reasoning, fewer hallucinations.
Response Speed	Very fast, near real-time. Best for lightweight interactive chatbot applications.	Faster than GPT-4, but slower than the mini version due to larger size.
Cost (OpenAI API)	$0.15 / 1M input tokens $0.60 / 1M output tokens	$1.25 / 1M input tokens $10.00 / 1M output tokens (Mini & Nano variants also exist with cheaper pricing)
RAG Compatibility	Works well with proper retrieval. Best for cost-sensitive apps.	Excellent for deep reasoning. More consistent with long or complex inputs.
Context Window	Up to 128K tokens	Up to 200K tokens (better for long-term context and larger documents)

Table 1. Compare GPT-4o-mini and GPT-5

9. Conclusion

This chapter has outlined the key concepts and theoretical foundations related to the development of a medical chatbot for STD advice. It explored the role of AI, NLP, and medical data reasoning in improving the accuracy and personalization of chatbot responses. Additionally, the importance of privacy and confidentiality in digital health solutions was highlighted. The integration of advanced AI techniques, such as RAG, shows significant promise in addressing the gaps found in existing STD-related chatbot solutions, offering the potential for a more personalized, accurate, and secure user experience.

The next chapter will present the methodology and design of the chatbot, detailing the process of building the system and the specific AI techniques utilized.

CHAPTER 3: METHODOLOGY

1. Research design

I adopt a system-building methodology to design, implement, and evaluate an STD-focused medical chatbot using hybrid Retrieval-Augmented Generation (RAG):

- **Data acquisition:** curate STD guidance from reputable sources with permissive educational use.
- **Preprocessing:** normalization, chunking (200–600 tokens with overlap).Retrieval: dual-channel BM25 (lexical) and FAISS (semantic) on L2-normalized embeddings; RRF fusion and de-dup.
- **Generation:** safety-first prompting grounded strictly in retrieved context with citations.
- **System integration:** Web UI, FastAPI backend, MySQL schema for users/sessions/conversations/messages.
- **Evaluation:** accuracy/grounding, latency targets, safety/advice quality, and usability.

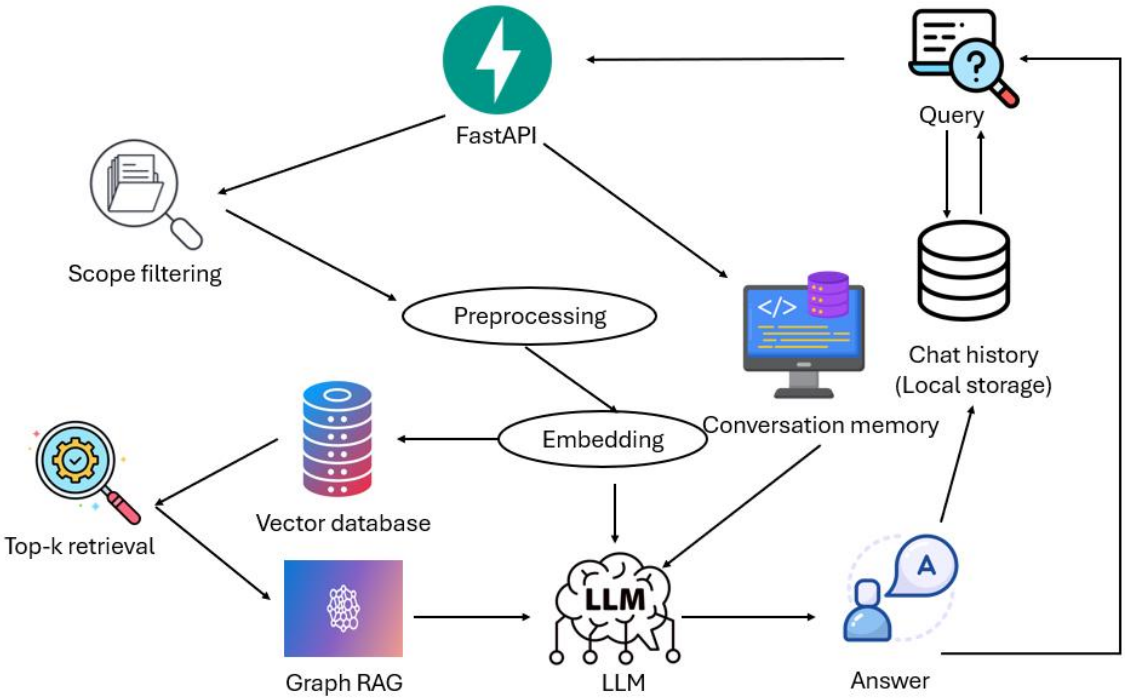

Figure 12. System architecture of the chatbot

2. Requirements

This section specifies what the system **must** and **should** do before any design or implementation decisions are made. Requirements are grouped into functional and non-functional categories and written in a thesis-friendly style: each item states intent, scope, and—where appropriate—**measurable acceptance criteria** so that later chapters can evaluate whether the system meets its objectives.

2.1. Functional requirements

The system shall provide the following **must-have** functions.

ID	Requirement	Rationale
1	User registration & login (JWT)	Secure access to personal threads.
2	Create/list/rename/delete conversations	Organize multi-topic sessions.
3	Ask text questions	Core interaction of the system.
4	Ask image + text questions	Support skin/lesion-style queries.
5	Persist Q&A per conversation	Auditability; system memory.

Table 2. Functional requirements.

Acceptance criteria:

1. Successful registration returns an access token (short-lived) and a refresh token (long-lived); subsequent authenticated requests include Authorization: Bearer <access>. Incorrect credentials must be rejected with HTTP 401 and no information leakage about which field failed.

2. The Conversations view lists threads sorted by updated_at; users can create, rename (title limited to ≤200 chars), and delete their own threads. Deleting a thread removes all associated Q&A.

3. Posting a text question returns an assistant answer within the performance envelope of N1 (see §2.3) and logs a Q&A pair to storage.

4. Uploading an image with a text question stores the file path, includes the image in the model request when available, and returns an answer; unsupported file types and >configured size must be rejected gracefully.

5. Each Q&A pair is stored with timestamps and convo_id; fetching a thread reconstructs user/assistant “bubbles” in order.

2.2. Nonfunctional requirements

These requirements constrain **how** the system performs its functions:

ID	Attribute	Requirement (testable target)
N1 – Performance	Latency	end-to-end p50 ≤ 3 s, p95 ≤ 7 s on reference host (≥4 vCPU, 8 GB RAM, stable network) with top-k ≤ 5; /chat-image adds ≤ 1.5 s.
N2 – Availability	Robustness	No unhandled exceptions crash the API; UI offers “Retry” for transient failures.
N3 – Usability	Learnability	First-time user can ask a question and read the answer without a tutorial; critical actions have clear labels/confirmations.
N4 –	A11y basics	Full keyboard navigation; color

Accessibility		contrast meets WCAG AA; images have alt text.
N5 – Privacy	Data minimization	Store only username, conversation Q&A, and bcrypt-hashed refresh tokens. No plaintext tokens or medical images in logs.
N6 – Security	Auth & transport	HTTPS enforced; JWT HS256 access tokens (short-lived); refresh tokens stored as bcrypt hashes; CORS restricted to prod origin.
N7 – Safety	Medical disclaimers	Prompts ground to retrieved context; assistant uses non-diagnostic language; advises care for red-flag symptoms.
N8 – Maintainability	Modularity	Separate UI, API, retrieval, storage; config via .env; no hard-coded secrets.
N9 – Portability	Environment	Runs on Windows & Linux; no GPU required; optional local FAISS on disk.
N10 – Observability	Logs/metrics	Access logs: route, latency, status; errors include trace IDs; counters for /chat and model timeouts.

Table 3. Non-functional requirements (N-series).

3. Knowledge curation and pre-processing

Sources: Public health guidance and reputable medical portals (VI/EN). Only pages with permissive terms for educational use are included.

data > sources_urls.txt

```
1  https://www.who.int/news-room/fact-sheets/detail/sexually-transmitted-infections-(stis)
2  https://www.cdc.gov/chlamydia/about/index.html
3  https://www.cdc.gov/gonorrhea/about/index.html
4  https://www.cdc.gov/syphilis/about/index.html
5  https://www.cdc.gov/trichomoniasis/about/index.html
6  https://www.cdc.gov/sti/about/about-genital-hpv-infection.html
7  https://www.cdc.gov/hiv/causes/index.html
8  https://www.cdc.gov/hepatitis-b/media/HepBSexualHealth.pdf
9  https://www.nhs.uk/conditions/chlamydia/
10 https://www.nhs.uk/conditions/gonorrhoea/
11 https://hivinfo.nih.gov/understanding-hiv/fact-sheets/hiv-and-aids-basics
12 https://womenshealth.gov/patient-materials/health-topic/sexually-transmitted-infections
```

Figure 13. Curated medical source URLs for knowledge-base construction

Normalization:

- Unicode and Vietnamese diacritics normalization; remove boilerplate and navigation text.
- Sentence and paragraph fixes; strip tables that cannot be parsed reliably.

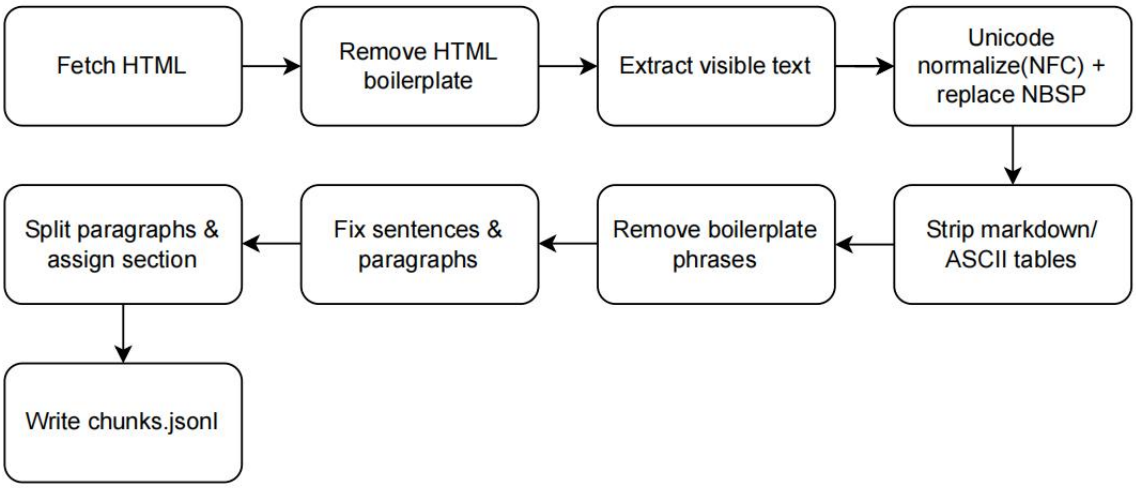

Figure 14. Normalization pipeline used during corpus construction

Chunking: Documents are split into **200–600-token chunks** with a **20–40-token overlap** to prevent information loss at boundaries.

4. Embedding and retrieval pipeline

4.1. Embedding model selection

I evaluate candidate sentence encoders on four criteria:

1. **Multilingual coverage (vi/en)** for mixed terminology.
2. **Retrieval quality** for short questions vs. short paragraphs.
3. **Latency/cost** on CPU inference.
4. **Compatibility** with FAISS and simple hybridization with BM25.

Given these constraints, I adopt **OpenAI text-embedding-3-small** as the default: it offers strong semantic retrieval quality, multilingual robustness, and low cost/latency. For an open-source alternative (offline builds), **multilingual-e5-base** is a good drop-in; it performs well on cross-lingual queries.

Model (family)	Lang	Dim	Relative speed*	Memory (GB)†	Notes / Typical use
OpenAI text-embedding-3-small	vi/en	1536	Fast (API)	0 (hosted)	Strong semantic search; low operational overhead
multilingual-E5-base	vi/en	768	Medium	~1.1	Good cross-lingual alignment; OSS; reproducible
multi-qa-	en	768	Medium	~0.9	QA-

mpnet-base-dot-v1	(ok vi)				oriented; widely used in RAG baselines
BGE-m3	vi/en	1024	Medium	~1.3	Strong multilingual retrieval; OSS

Table 4. Hosted encoder candidates

4.2. Sentence embeddings

I use **text-embedding-3-small (1536-D)** for robust vi/en coverage at low latency/cost. All query and chunk vectors are **L2-normalized**, making the **dot product equivalent to cosine similarity**. Embeddings are stored in a **FAISS IndexFlatIP** with a **row→chunk-ID** map and a **JSONL metadata** file (title, section, URL) to enable citations and fast reloads. A small build manifest records the model ID, dimension, build time, and chunker version for reproducibility. This configuration offered the best balance of retrieval quality and speed for our corpus; **multilingual-e5-base** remains a viable OSS fallback for offline builds.

4.3. Building vector database

4.3.1. Text normalization and tokenization

To minimize spurious lexical differences before encoding, raw text is standardized as follows: Unicode NFC normalization, Vietnamese diacritic normalization, lowercasing for English segments, de-hyphenation and line-break repair, and sentence/heading segmentation. Tokenization follows the encoder's WordPiece/BPE rules; long sections are split with a sliding window that respects sentence boundaries.

4.3.2. Model candidates and selection criteria

Candidate encoders are shortlisted by four criteria: language coverage (vi/en), retrieval quality reported in open benchmarks, embedding dimensionality (memory/speed), and availability in common libraries.

Model (HF id)	Dim	Lang	Notes / Suitability
sentence-transformers/multi-qa-mpnet-base-dot-v1	768	multi	Strong general QA retrieval; mature ecosystem; good balance for CPU.
intfloat/multilingual-e5-base	768	multi	Good cross-lingual alignment; consistent with E5 family prompts.
BAAI/bge-small-en-v1.5	384	en	Very fast/small; weaker on vi, usable if corpus mostly EN.

BAAI/bge-m3	1024	multi	Unified dense+lexical+multi-vector; higher memory footprint.
sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2	384	multi	Lightweight multilingual baseline; competitive speed.

Table 5. Open-source encoder candidates

4.3.3. Hyperparameters and operational settings

Operational parameters govern throughput and stability on CPU.

Setting	Value / Rationale
Encoder	text-embedding-3-small
Dimension	1536 (base-size encoders)
Max sequence length	512 tokens; sliding window for longer spans
Batch size (CPU)	64 (tuned to available RAM)
Dtype	float32 (quantization optional; see §3.2.5)
Output	L2-normalized vectors
Preprocessing	Unicode NFC, VI diacritic normalization, sentence/heading segmentation

Table 6. Encoder settings

4.3.4. Robustness to layperson input

To mitigate vocabulary mismatch and minor spelling errors, embedding retrieval is combined later with BM25 and fuzzy matching. Within the encoder, subword tokenization already reduces OOV sensitivity; remaining noise is handled by the hybrid fusion strategy.

4.3.5. Reproducibility

This work collects content from HTML/PDF sources, cleans it, and normalizes to Unicode. The text is split by headings into ~200–400-word chunks with ~60 words of overlap. Each chunk is tagged with metadata (id, url, title, language, section, date_accessed) to preserve provenance and enable tracing. We also log per-URL chunk counts to monitor quality and keep the corpus balanced. The resulting dataset is saved to data/chunks.jsonl and serves as the input to the indexing stages.

```
[Running] python -u "c:\Users\Lehuy\Downloads\LVTN\Thesis\scripts\build_chunks.py"
[OK] https://www.cdc.gov/sti/about/index.html -> 1 chunks
[OK] https://www.cdc.gov/std/treatment-guidelines/default.htm -> 1 chunks
[OK] https://www.cdc.gov/herpes/about/index.html -> 1 chunks
[OK] https://www.cdc.gov/hpv/index.html -> 1 chunks
[OK] https://www.cdc.gov/hepatitis/hbv/index.htm -> 1 chunks
[OK] https://www.cdc.gov/hepatitis/hcv/index.htm -> 1 chunks
[OK] https://www.who.int/news-room/fact-sheets/detail/gonorrhoea -> 0 chunks
[OK] https://www.who.int/news-room/fact-sheets/detail/syphilis -> 1 chunks
[OK] https://www.who.int/news-room/fact-sheets/detail/trichomoniasis -> 1 chunks
[OK] https://www.who.int/news-room/fact-sheets/detail/herpes-simplex-virus -> 3 chunks
[OK] https://www.who.int/news-room/fact-sheets/detail/human-papillomavirus-(hpv)-and-cervical-cancer -> 5 chunks
[OK] https://www.who.int/news-room/fact-sheets/detail/hepatitis-b -> 3 chunks
[OK] https://www.who.int/news-room/fact-sheets/detail/hepatitis-c -> 2 chunks
[OK] https://www.nhs.uk/conditions/syphilis/ -> 1 chunks
[OK] https://www.nhs.uk/conditions/trichomoniasis/ -> 1 chunks
[OK] https://www.nhs.uk/conditions/genital-herpes/ -> 1 chunks
[OK] https://www.nhs.uk/conditions/genital-warts/ -> 1 chunks
[OK] https://www.nhs.uk/conditions/hiv-and-aids/ -> 1 chunks
[OK] https://www.nhs.uk/conditions/hepatitis-b/ -> 2 chunks
[OK] https://www.nhs.uk/conditions/hepatitis-c/ -> 1 chunks
[OK] https://www.nhs.uk/conditions/pelvic-inflammatory-disease-pid/ -> 1 chunks
[OK] https://medlineplus.gov/sexuallytransmitteddiseases.html -> 4 chunks
[OK] https://medlineplus.gov/chlamydiainfections.html -> 3 chunks
[OK] https://medlineplus.gov/gonorrhea.html -> 2 chunks
[OK] https://medlineplus.gov/syphilis.html -> 3 chunks
[OK] https://medlineplus.gov/trichomoniasis.html -> 2 chunks
[OK] https://medlineplus.gov/genitalherpes.html -> 2 chunks
[OK] https://medlineplus.gov/hiv.html -> 3 chunks
[OK] https://medlineplus.gov/hepatitisb.html -> 3 chunks
[OK] https://medlineplus.gov/hpv.html -> 3 chunks
```

Figure 15. Vectorization and indexing summary

In the **vectorization and indexing** stage, all chunks from data/chunks.jsonl are embedded into **1536-dimensional**, L2-normalized vectors and indexed with **FAISS / IndexFlatIP** (cosine via inner product). The build produces two artifacts-data/faiss.index and data/faiss.ids.npy-with **270 vectors** added (nvecs = 270). A manifest is then updated (model ID, index type, SHA-256 of each file) to ensure verifiability and exact reproducibility.

```
[Running] python -u "c:\Users\Lehuy\Downloads\LVTN\Thesis\scripts\build_faiss.py"
Saved: data/faiss.index & data/faiss.ids.npy | dim: 1536 | nvecs: 270
```

Figure 16. FAISS build log

Figure 17. FAISS index and metadata files for reproducibility

The build_graph_from_chunks.py script reads the same data/chunks.jsonl corpus and writes the GraphRAG structure to data/graph.json. The log confirms that the build completed successfully and reports the resulting graph size (14 nodes from 8 source documents), which supports later debugging and reproducibility.

Figure 18. Graph build log

4.4 Lexical Retrieval (BM25)

In parallel with the vector index, the system maintains a **BM25** channel over the same chunk set. BM25 ranks passages by term frequency and inverse document frequency with length normalization, which is particularly effective for **clinical abbreviations and exact terms** (e.g., NAAT, PCR, swab, HBV) that semantic retrieval may underweight. Vietnamese and English text are tokenized into alphanumeric terms (Vietnamese diacritics preserved); English is lower-cased.

Parameters: $k_1 = 1.5, b = 0.75$.

Retrieval: return **top-8** passages by lexical score. This list is then passed to **RRF** to be fused with the FAISS results, followed by **de-duplication** by *(source, section)* and **intent-aware prioritization** (Symptoms / Diagnosis / Treatment / Prevention).

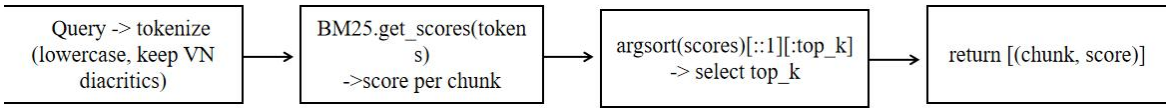

Figure 19. BM25 search pipeline

4.5. Hybrid Retrieval

I fuse the **BM25 (lexical)** and **FAISS (semantic)** ranking lists using **Reciprocal Rank Fusion (RRF)**:

$$\mathrm{RRF}(d) = \sum_{l \in \{\mathrm{BM25,FAISS}\}} \frac{1}{k + \mathrm{rank}_l(d)}, k = 60 \quad (1)$$

where $\mathrm{rank}_l(d)$is the 1-based rank of document d in list l. A larger k dampens the influence of deep ranks and increases stability. After fusion, I **de-duplicate** by *(source, section)*, **prioritize** passages aligned with the detected **intent**, and select the **top-6** passages to form the **prompt context** for the generation step.

4.6. Safe prompting and answer generation

This work adopts a safety-first prompting scheme to control the behavior of the generator. A fixed system prompt encodes clinical and ethical constraints: the assistant must avoid definitive diagnoses, state uncertainty explicitly, and direct users to professional care when red-flag symptoms are mentioned (e.g., severe pain, high fever, genital bleeding, symptoms of sepsis). The prompt also enforces privacy protections—the model should not solicit or retain personally identifiable information and must not store PHI in logs or retrieval indices—and requires a non-judgmental, stigma-free tone appropriate to sexual-health contexts.

Answers are grounded in retrieved passages. The message structure is:

1. *system*: safety rules (plus an optional context block containing the top-K cited passages),
2. *user*: the current question.

The assistant is instructed to use only the provided context, and to append a Sources section that lists the title · section · URL of each passage supporting the answer. If retrieved evidence is weak or conflicting, the model must communicate limitations, refrain from speculation, and provide next-step guidance (self-care, testing options, or urgent care escalation). The final output is concise, plain-language, and action-oriented while remaining transparent about uncertainty and provenance.

4.7. System integration

This section describes how the web UI, FastAPI back end, and relational database work together to deliver the STD chatbot after adding BM25 + FAISS hybrid retrieval. The UI handles interaction and citations; the API orchestrates intent detection → retrieval → safe LLM generation; the database provides identity, sessions, conversation state, and auditable Q&A logs.

5. Web Front-End

5.1. User flows

To support reliable and privacy-preserving interaction with the chatbot, the web client consolidates authentication and a chat workspace within a single page application. After users register and sign in, a short-lived JWT access token authorizes API calls while a refresh token enables seamless renewal without repeated logins. The chat surface is designed for clinical information seeking: it accepts free-text questions and optional image uploads (e.g., lesion/skin photos), organizes dialogue into named conversations that can be created, renamed, or deleted, and replays history as user/assistant message bubbles for traceability. Each answer is accompanied by clickable citations (title · section · URL) to make evidence explicit.

- **Authentication:** Users register and log in. A short-lived access token (JWT) is stored client-side; a **refresh token** is used to obtain new access tokens.
- Chat workspace:
    - **Text input** and **image upload** for lesion/skin questions.
    - **Conversation manager** (create/rename/delete), and a **scrollable history** that replays bubbles.
    - **Citations** rendered under each answer (title · section · URL), opening in a new tab.

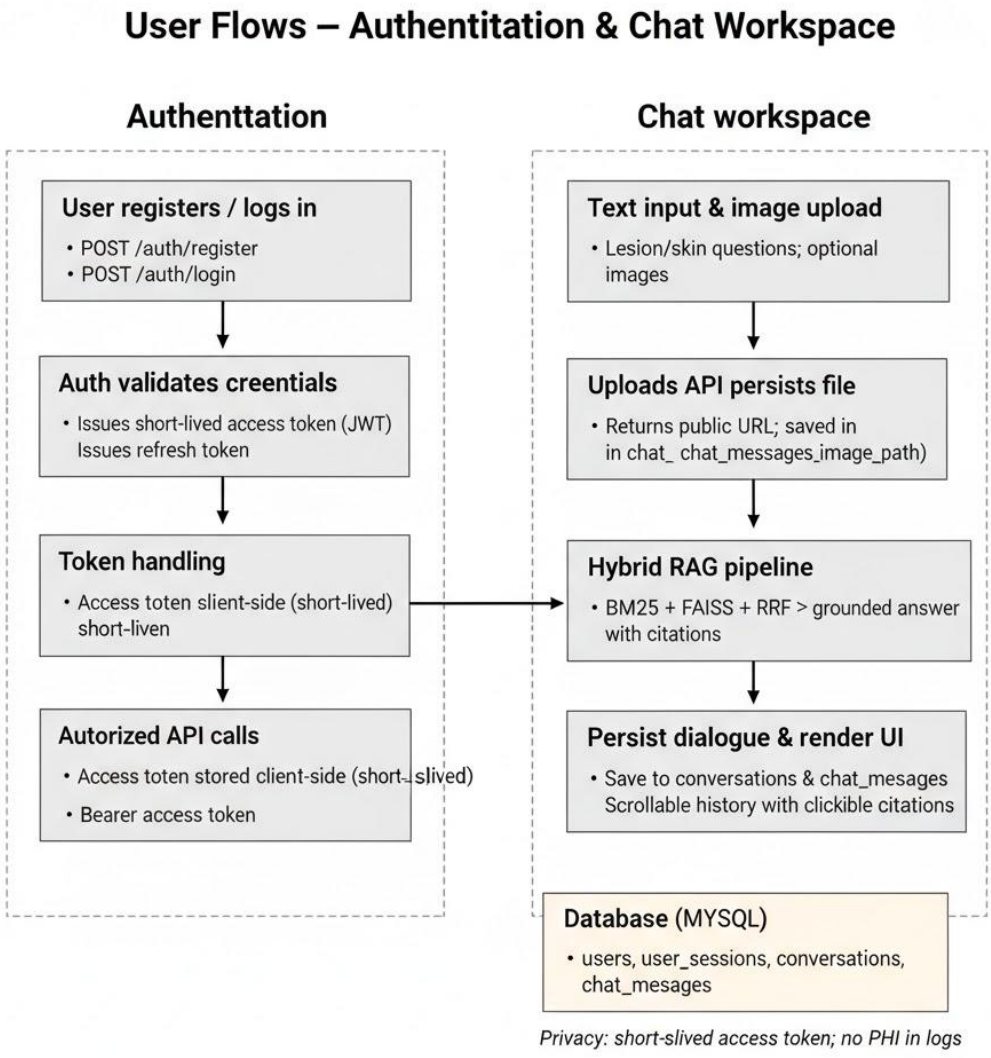

Figure 20. User flows – authentication and chat workplace

5.2 UX & accessibility

To accommodate a broad user base and reduce friction during health-information seeking, the interface follows simple, practical accessibility choices. All labels and key actions are bilingual (VI/EN), typography and layout maintain clear visual contrast, and the entire chat surface is keyboard-navigable with predictable tab order and visible focus states. The UI reinforces responsible use through safety hints (e.g., "This is not a diagnosis; seek care for red-flag symptoms.") shown near the input area and in escalation banners when appropriate. When problems occur—loss of network, expired tokens, or API failures—the client surfaces graceful error toasts that explain the issue and suggest the next action, while preserving the user's current text and conversation state.

5.3 Static assets and uploads

The API exposes a public mount for user uploads at /uploads/<uuid>.<ext>. When a user submits an image via POST /chat-image (multipart/form-data), the

server persists the file under an application-scoped uploads directory and returns the public URL in the response. The URL is also stored in chat_messages.image_path so that the front-end can render the image inline in the conversation history.

```
app.mount("/uploads", StaticFiles(directory=str(UPLOAD_DIR)), name="uploads")
```

Figure 21. FastAPI static uploads mount and authentication endpoints

6. Front-End Implementation

The user interface is designed as a single-page application (SPA) that seamlessly toggles between the Authentication and main App screens. The sidebar displays a list of conversations, while the main pane shows message threads along with the message composer, which includes a textarea, image picker, and send button.

Styling is managed via style.css, where dark theme variables and chat bubble styles are defined to ensure a consistent and visually appealing experience.

Key toolbar actions include:

- Starting a new chat
- Exporting conversation snapshots in JSON format
- Clearing local selections
- Logging out securely

7. Back-End API (FastAPI)

7.1. Endpoints

Auth: POST /auth/register, POST /auth/login, POST /auth/refresh, POST /auth/logout, GET /me

Figure 22. FastAPI authentication endpoints

Conversations:
GET /conversations, POST /conversations, PATCH /conversations/{id}, DELETE /conversations/{id}, GET /conversations/{id}/messages

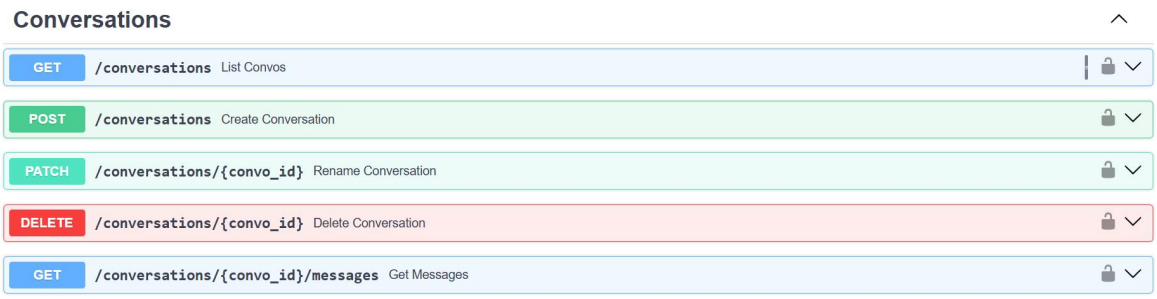

Figure 23. FastAPI conversation endpoints

Chat: POST /chat (text), POST /chat-image (image + optional text)

Figure 24. FastAPI chat endpoints

Voice: POST /voice/tts, POST /voice/transcribe, GET /voice/voices

All responses are JSON; errors use standard HTTP codes with user-readable messages.

Auth:

POST /auth/register, POST /auth/login → issue **access JWT (short-lived)** + **refresh token**.

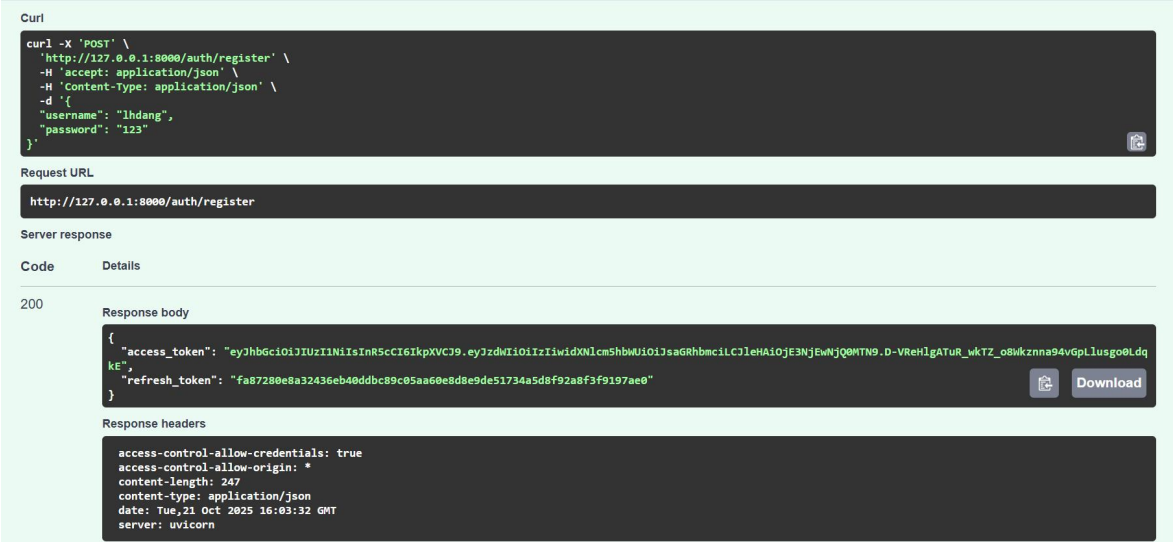

Figure 25. Swagger UI for register

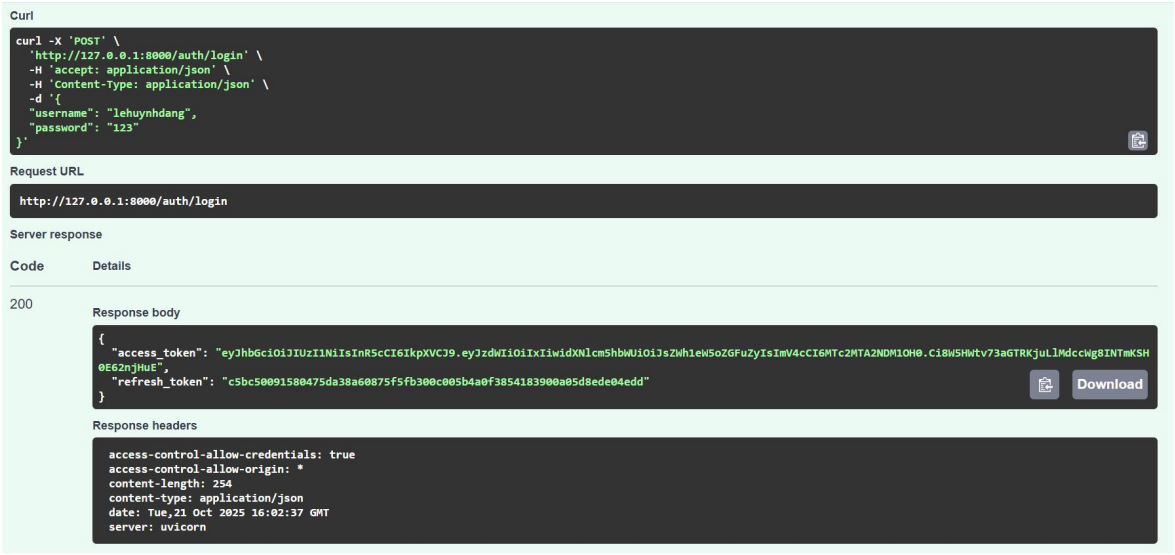

Figure 26. Swagger UI for login

POST /auth/logout → server-side refresh **revocation** (revoked_at).

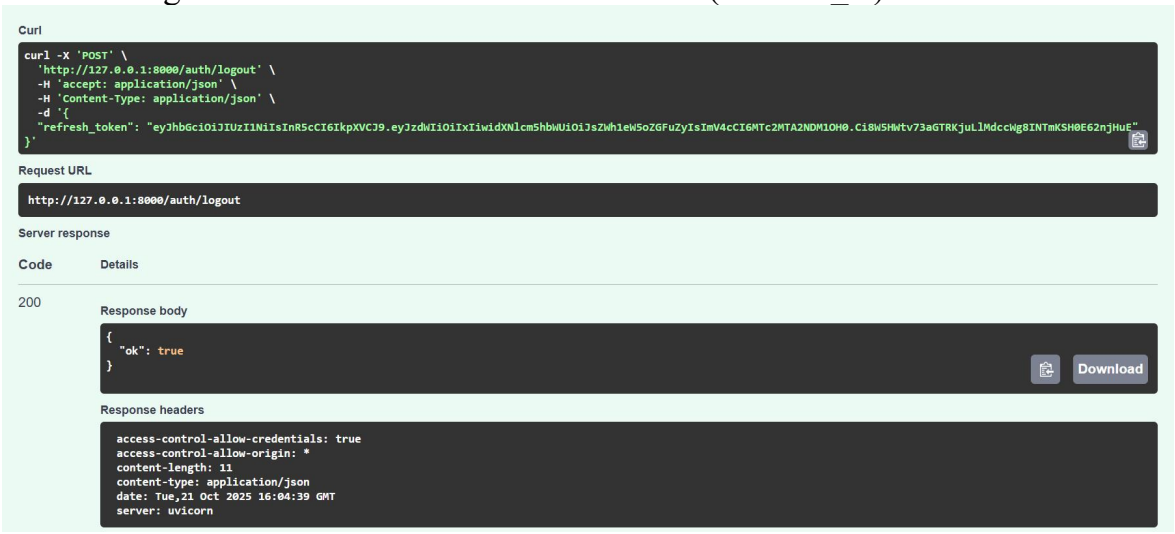

Figure 27. Swagger UI for logout

Conversations & Messages:

GET/POST/PATCH/DELETE /conversations, GET /conversations/{id}/messages.

Figure 28. Swagger UI for listing conversations

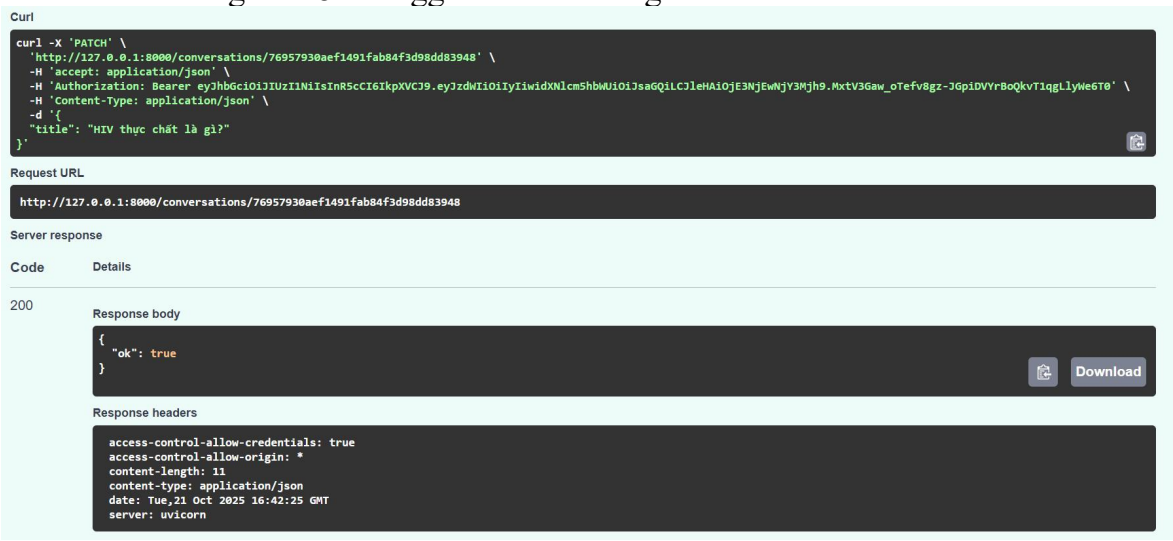

Figure 29. Swagger UI for changing conversation name

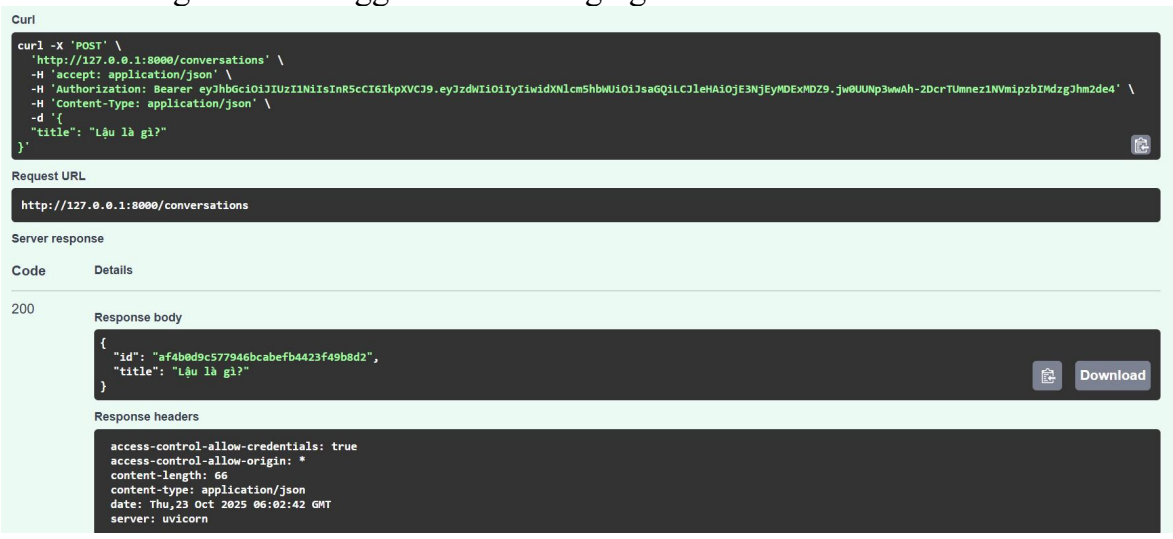

Figure 30. Swagger UI for creating a conversation

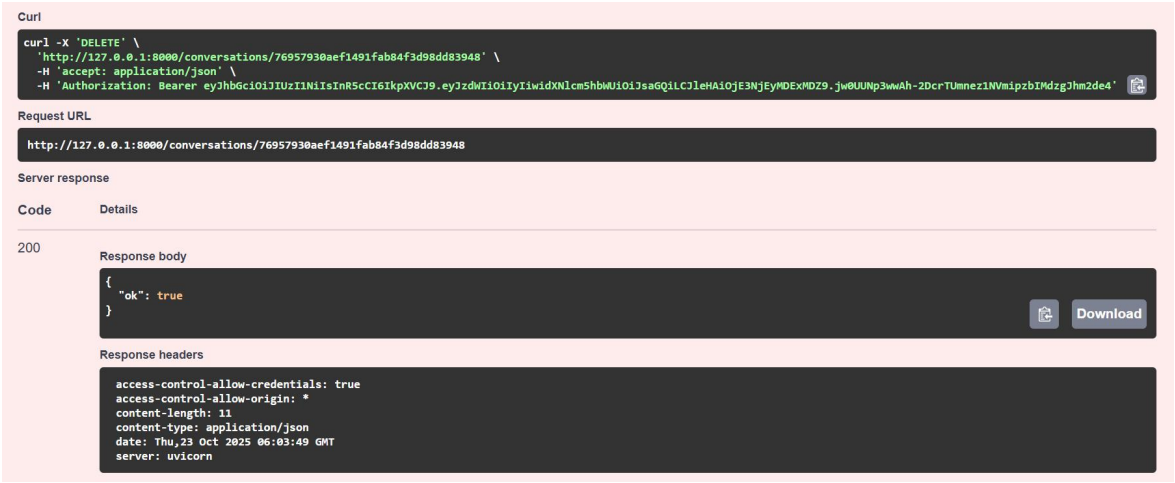

Figure 31. Swagger UI for deleting a conversation

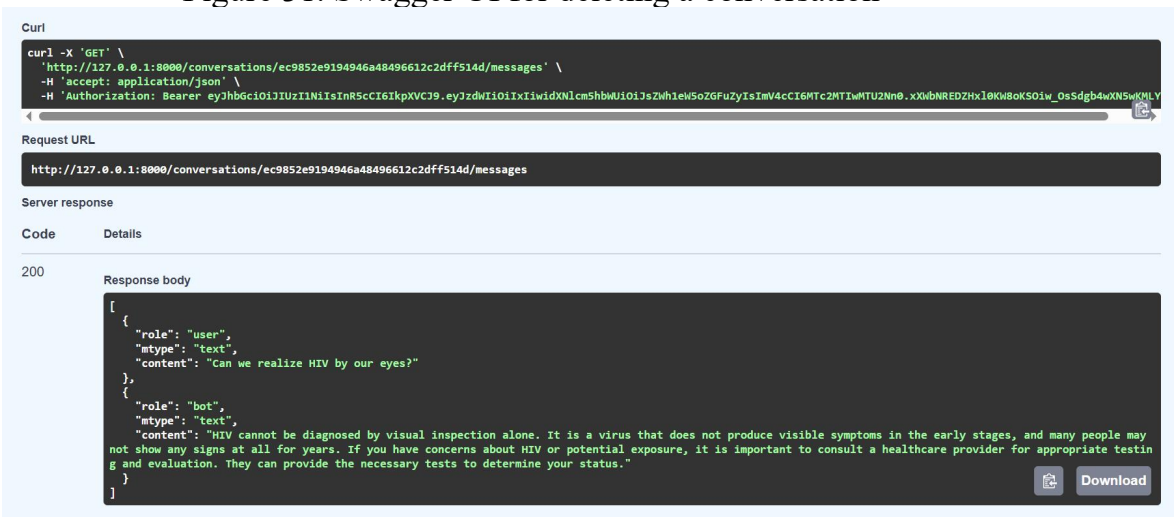

Figure 32. Swagger UI for getting a message

Chat:

POST /chat (text) and POST /chat-image (image + text).

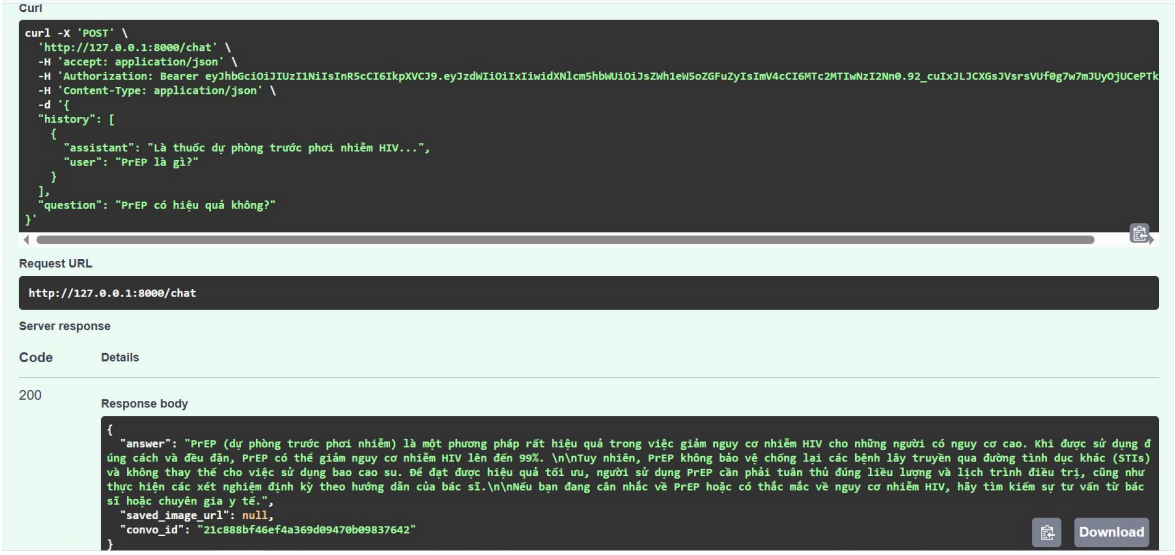

Figure 33. Swagger UI for asking a question

Figure 34. Swagger UI for asking a question by image

Voice

POST /voice/tts, POST /voice/transcribe, GET /voice/voices

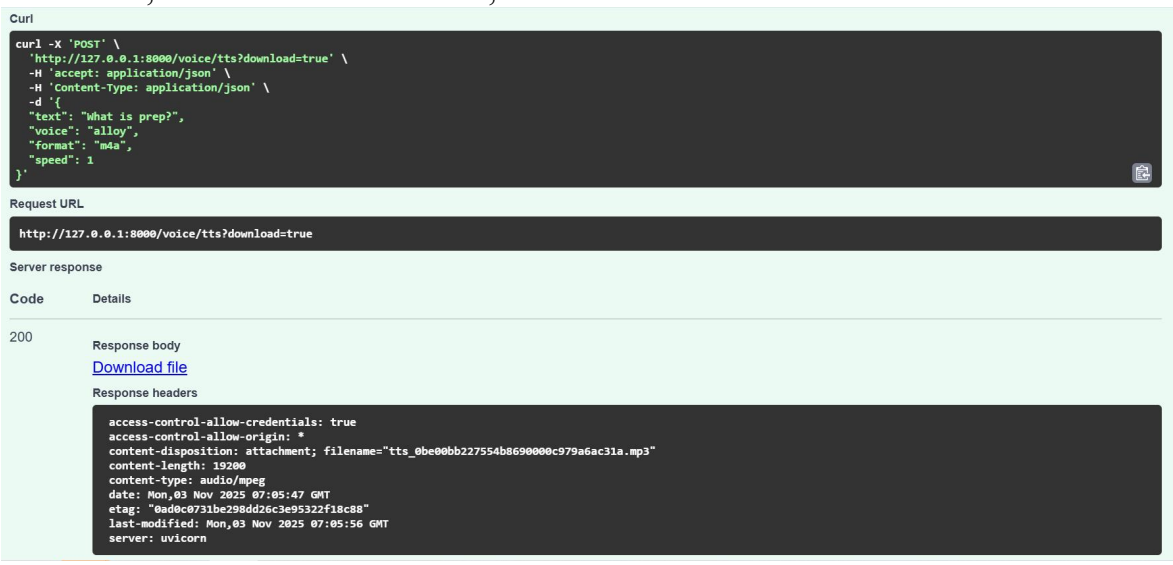

Figure 35. Swagger UI for the Text-to-Speech

Figure 36. Swagger UI for the Speech-to-Text

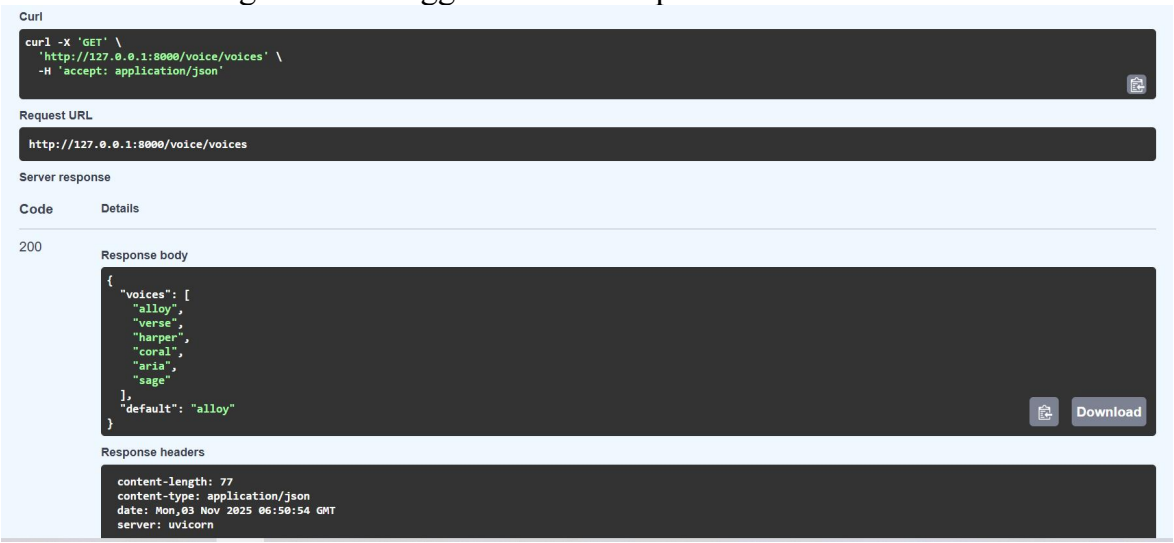

Figure 37. Swagger UI for listing available TTS voices

Ops:

GET /healthz; **Swagger UI** at /docs with bearer auth.

Figure 38. Swagger UI for getting healthz

7.2 Retrieval & generation lifecycle (/chat)

When a user submits a message to /chat, the backend executes a fixed, end-to-end pipeline that balances **safety**, **relevance**, and **latency**. It authenticates the caller, manages the conversation state, detects the user's intent, then retrieves candidates via a **hybrid ranker** (BM25 + FAISS with RRF fusion). A bounded, citation-ready context is constructed and passed—with a safety system prompt—to the LLM. The resulting answer, together with provenance and optional trace, is persisted and returned to the client. The detailed steps are:

- **JWT validation**
  The API validates the bearer token to identify the user and rejects invalid or expired credentials.
- **Conversation management**
  If the conversation ID is missing, a new conversation is created and its title is initialized from the first user question.
- **Intent detection**
  A bilingual (VI/EN) keyword map assigns the query to one of the sections {Symptoms, Diagnosis, Treatment, Prevention}; if no match is found, the intent is left as None.
- **Hybrid candidate retrieval**
  • **BM25 (lexical):** Rank cleaned chunks by lexical relevance using a tokenizer that preserves Vietnamese diacritics.
  • **FAISS (semantic):** Rank the same chunk set by semantic similarity. Chunks and the query are embedded and L2-normalized; similarity is computed via IndexFlatIP (dot-product ≡ cosine on unit vectors).
  • **RRF fusion:** Merge the top-8 results from each list using Reciprocal Rank Fusion with $k_{bias} = 60$to obtain a single ranked list.
- **Post-processing**
  De-duplication: Keep at most one passage per (source, section) pair to increase coverage.
  Intent-aware filtering: Prefer passages whose section matches the detected intent.
  Selection: Keep $C = 6$ top passages as the retrieval context.
- **Context construction**
  Build a bounded context block that lists **[title · section]** and a concise snippet for each selected passage (with provenance retained for citation).
- **Answer generation**
  Call the LLM with a **safety system prompt** (non-diagnostic, privacy-preserving, refer for red-flag symptoms), the constructed context, and the user question. Use a low temperature for stability.

- **Persistence**
  Write a single row per interaction to chat_messages (question, answer, optional image_path) and update the conversation timestamp.
- **Response**
  Return the generated answer together with **Sources** (title · section · URL). If tracing is enabled, also return a compact retrieval trace (candidate lists, fused ranks/scores, and timing).

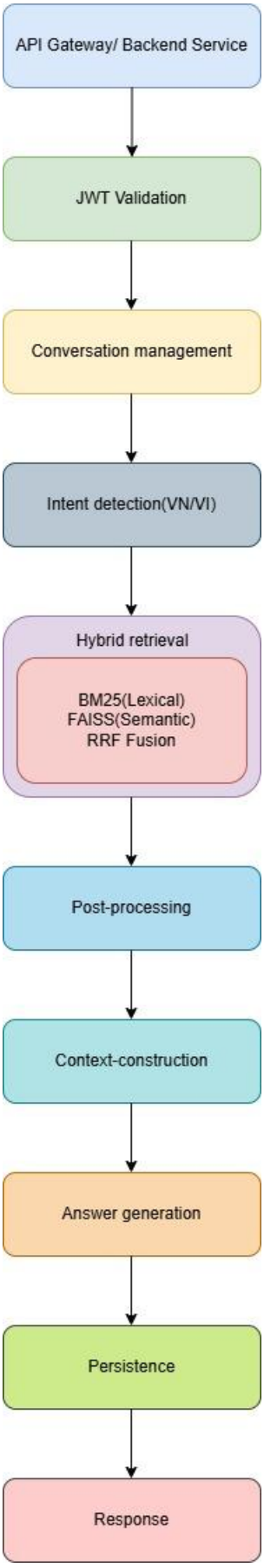
API Gateway/ Backend Service
JWT Validation
Conversation management
Intent detection(VN/VI)
Hybrid retrieval
BM25(Lexical)
FAISS(Semantic)
RRF Fusion
Post-processing
Context-construction
Answer generation
Persistence
Response

Figure 39. Chat work-flow

7.3. Image-assisted lifecycle (/chat-image → /chat)

When a message includes an image, the backend orchestrates a two-phase flow that preserves safety, relevance, and latency. It first accepts the upload and issues a stable URL, optionally derives lightweight visual descriptors, then proceeds with the standard **/chat** pipeline using the image as additional evidence. **The flow is:**

- **Upload.** The client sends the image via **POST /chat-image** (multipart); the server persists it under /uploads/<uuid>.<ext> and returns { "image_url": "…" }. The URL is also saved in chat_messages.image_path when the message is persisted.
- **Consolidate query.** Combine the user's text with extracted tags (if any) to form the final query string.
- **Retrieval & generation.** Call **POST /chat** with the text, the **image_url**, and the conversation ID. The pipeline in §6.2 runs as-is (BM25 + FAISS with **RRF**, de-dup, intent filter, **C=6** passages). The prompt adds a short **"Image evidence"** block (URL + descriptors) so the LLM can reference the image safely.
- **Persistence & response.** Save the interaction (question, answer, image_path, citations). Return the answer, **Sources** (title · section · URL), and the image_url so the front-end can render the image inline.

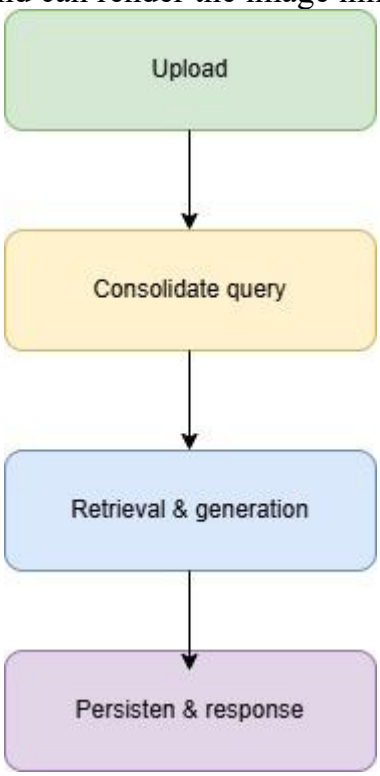

Figure 40. Chat-image work-flow

7.4. Error handling & fallbacks

The /chat and /chat-image pipelines are wrapped in defensive guards so that a single failure does not break the user flow. Errors are logged on the backend

(including the phase and conversation id), while the UI always receives either an answer or a short error message.

A) Retrieval and GraphRAG

The system currently supports a simple **LLM-only** mode and an optional **GraphRAG** mode:

- If GraphRAG is disabled (missing graph/alias files, env flag off) or raises an exception during entity linking, expansion, or context building, the backend silently **degrades to LLM-only**.
- In this case the chat still returns a normal answer; the failure is only recorded in the server logs with a trace object such as {mode: "llm_only", used_context: false}.
- When GraphRAG succeeds, the trace contains {mode: "graphrag", used_context: true, candidates: [...]}. This trace is logged on the server; the UI can optionally display it in a developer view but is hidden from end-users.

No BM25/FAISS hybrid retrieval is used in the current implementation.

B) Image handling

The /chat-image endpoint focuses on reliable upload rather than vision reasoning:

- If no file is provided, or the upload fails validation, the server returns **400** with a clear error message.
- On success, the image is saved under /uploads/… and a simple confirmation message is added to the conversation ("The image has been received; visual analysis is not yet enabled.").
- If any server-side error occurs after upload, the file path is still preserved when possible and the user receives a textual fallback instead of the request failing silently.

C) Model and network resilience

Calls to the OpenAI API are wrapped in a try/except block:

- On success, the model answer is returned and stored in the chat_messages table.
- On any exception (network error, timeout, rate limit, or other API failure), the backend returns a short apology in Vietnamese indicating that the service is temporarily unavailable and inviting the user to try again.
- The exception details are logged on the server for debugging. No multi-step retry or fine-grained error categorization is performed in the current version.

D) Auth, validation, and persistence

- Invalid or expired JWT tokens result in **401 Unauthorized**. The frontend reacts by showing the login overlay while keeping the conversation history intact.
- Missing required fields (e.g., an empty question) are rejected with **400 Bad Request** and a human-readable error string.
- Database write errors are surfaced as server errors and logged. The application aims to send the generated answer back to the client even if

persistence fails, but persistence failures are not yet marked explicitly in the response payload.

E) User-visible behaviour (frontend)

- Every user message is followed by either a concrete assistant message or an explicit error bubble (prefixed with ✕), so the thread never "hangs" in a pending state.
- If a request fails, the error is shown in the conversation and (for auth problems) a toast or login prompt is displayed; the existing conversation remains readable.
- Text and image messages that were already added to the thread remain in place after an error, so the user does not lose context when resending a question.

7.5. Relational database

7.5.1. Core schema

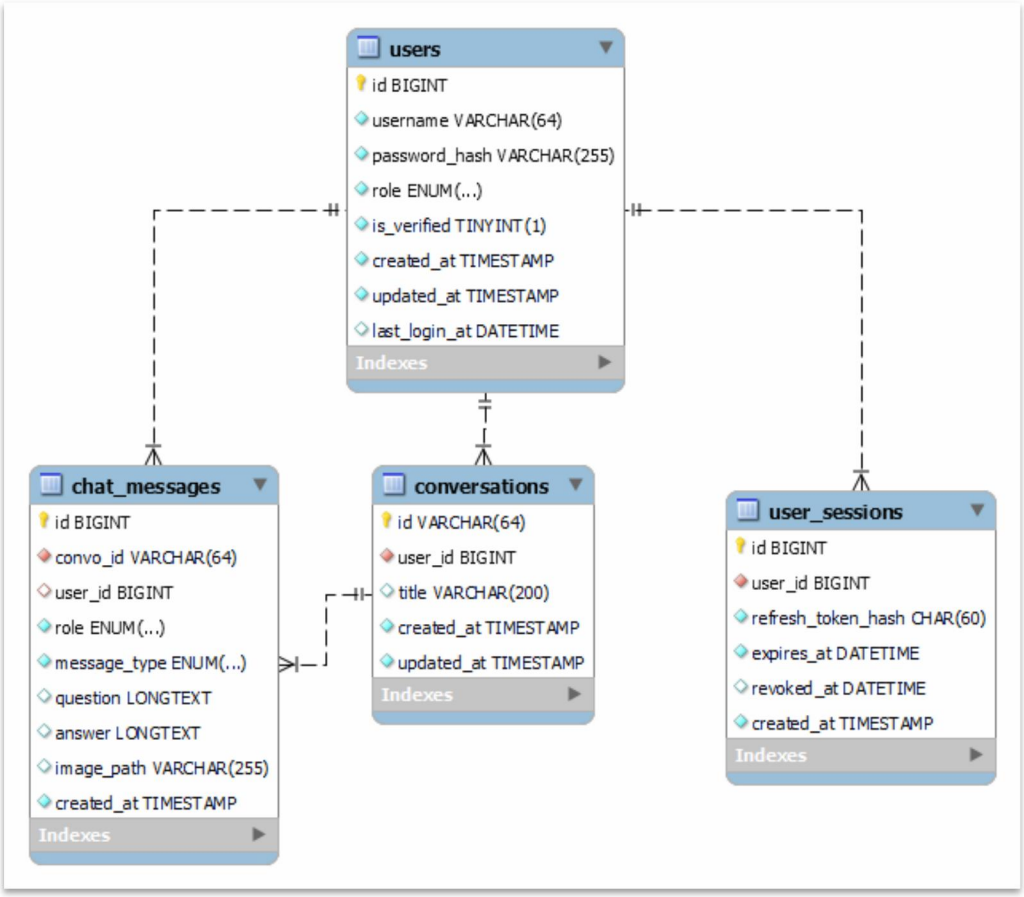

Figure 41. Core relational schema (ER diagram)

The schema with four tables: users, user_sessions, conversations, and chat_messages. users holds account and audit fields, while user_sessions stores only a **hash** of each refresh token plus expiry/revocation times so sessions can be invalidated without touching content. A conversation is a user-owned thread (UUID/ULID id and optional title), and every chat_message belongs to exactly one conversation; messages record role (user/assistant/system), type (text/image), the

payloads, an optional image_path, and a timestamp so the thread can be read in order. Foreign keys enforce ownership, with cascade from conversations to messages to avoid orphans. Targeted indexes—username for login, (user_id, created_at) for listing threads, (convo_id, created_at) for loading messages—keep the hot paths fast. Storing images by path keeps the database lean, while LONGTEXT fields capture full prompts and answers. This separation of identity, session state, threads, and messages makes permission checks simple, revocation safe, and analytics straightforward.

7.5.2. Deployment & Operations

To ensure a smooth and scalable deployment, the system follows a well-defined startup procedure and environment configuration:

Startup:

On initialization, the system performs several critical loading steps:

- Load chunks.jsonl (document chunks)
- Initialize or warm-load the BM25 index
- Load FAISS components: faiss.index and faiss.ids.npy

Environment Configuration:

Runtime behavior is controlled through environment variables, including:

- DB_* (database connections)
- JWT_SECRET (authentication)
- OPENAI_* (API keys and settings)
- EMB_MODEL (embedding model configuration)
- Feature flags to toggle experimental or optional features

CORS Policy:

CORS is permissive in development to facilitate testing and local integration. In production, it is restricted to the official domain to ensure security and prevent unauthorized access.

Performance Considerations:

- Keep context windows compact (ideally ~6 passages) for faster responses
- L2-normalize embeddings for consistent similarity calculations
- Use IndexFlatIP for exact search, or scale with IVF/HNSW as needed

Observability & Logging:

- Emit structured logs with timing metrics and top-k result IDs
- Optionally include a trace field in responses for offline debugging and analytics

8. Risks and mitigations

In any complex system, identifying potential risks and implementing strategies to mitigate them is essential to ensure reliability and accuracy. The following key risks have been identified along with their corresponding mitigation approaches:

Domain Drift:

Medical treatment recommendations may evolve over time, potentially causing outdated guidance. To mitigate this risk, provenance metadata is maintained for traceability, and regular corpus refresh procedures are implemented to keep the knowledge base up to date.

Long Passages:

Handling lengthy documents can lead to truncation issues due to token limits (512 tokens). This is addressed by applying a sliding window approach with overlap, ensuring no important context is lost during chunking.

Language Mixing:

Since the system may encounter multilingual inputs, multilingual encoders are preferred to handle diverse languages effectively. For predominantly Vietnamese sources, alignment quality is verified through bilingual development queries to ensure semantic consistency.

Latency Spikes:

To prevent sudden increases in response time caused by CPU overload, batch sizes are capped. Additionally, Approximate Nearest Neighbor (ANN) indexing is introduced when the number of rows (Ngrows) grows significantly, enabling scalable and efficient retrieval.

9. Deployment architecture

A minimal yet robust deployment setup consists of the following components:

- **Application Server:**
  The backend runs on FastAPI served by Gunicorn or Uvicorn, placed behind an HTTP reverse proxy such as Nginx. HTTPS is enabled to secure all communications.
- **Database:**
  A managed MySQL instance is used with network access restricted to authorized hosts only. Daily snapshots are taken to ensure data durability and easy recovery.
- **Static Uploads:**
  Uploaded files are stored either in object storage services or dedicated storage volumes. Access to sensitive images is controlled via signed URLs or authenticated routes to maintain security.
- **Secrets Management:**
  Environment variables are handled securely through .env files or secret managers, with regular key rotation policies enforced.
- **CORS Configuration:**
  Cross-Origin Resource Sharing is strictly limited to the production domain to prevent unauthorized external access.

10. Conclusion

This chapter has translated the problem formulation into a concrete system design for MedChat (STDs). Starting from the functional and non-functional requirements, it specified how the assistant must support bilingual (VI/EN) conversations, image and voice input, persistent chat history, and strict safety constraints. The chapter then detailed how curated medical sources are cleaned, chunked, and embedded, and how these artifacts are organized into FAISS, BM25, and GraphRAG structures so that the chatbot can ground its answers in up-to-date, domain-specific evidence rather than relying solely on the base LLM.

On top of this knowledge layer, the chapter described the overall architecture: a FastAPI backend with JWT-based authentication, a MySQL database for users and conversations, and a single-page web frontend that exposes dark/light themes, VI/EN toggles, and multimodal input. The retrieval pipeline combines semantic and lexical scores via RRF and optionally augments them with graph-based expansion, while safety prompting, constrained answer templates, and robust error handling aim to reduce hallucinations and provide transparent fallbacks when any component fails. Voice and image flows are integrated as thin extensions that reuse the same conversation and logging infrastructure.

Finally, the chapter discussed operational aspects such as logging, observability, deployment, and risk mitigation. Design choices like compact context windows, normalized embeddings, and rate-limit handling are intended to keep latency and cost manageable, while structured traces and token-level logs support later analysis. Overall, Chapter 3 establishes a coherent, implementable blueprint that connects the theoretical ideas of Chapter 2 with the practical experiments in Chapter 4, ensuring that the implemented prototype is secure, maintainable, and ready for systematic evaluation.

CHAPTER 4: EXPERIMENTATION AND EVALUATION

1. Implementation process

1.1. Embedding medical documents

Before the chatbot can answer questions, the medical sources are preprocessed and embedded into a retrieval index. Raw HTML and PDF documents from trusted STI websites are first cleaned, Unicode-normalized, and split into overlapping passages. Each passage is stored in data/chunks.jsonl together with provenance metadata such as URL, section title, and access date.

Next, a vectorization script loads these chunks and calls the OpenAI embedding model to produce 1 536-dimensional L2-normalized vectors. The vectors are written into a FAISS index (data/faiss.index) and an ID mapping file (data/faiss.ids.npy), which together support fast approximate nearest-neighbour search at runtime. In parallel, a lightweight lexical channel is prepared (BM25) over the same chunk set, and an alias map plus knowledge graph are built from entity links, producing alias_map.json and data/graph.json.

When the backend starts (Figure 40), it prints the resolved paths for these artifacts and loads them into memory. The console log confirms that DATA_DIR, CHUNKS_PATH, GRAPH_PATH, and ALIAS_PATH are detected correctly and that GraphRAG is enabled with the expected number of aliases. From this point onward, every user query can be grounded in the embedded medical corpus through hybrid retrieval and, when configured, graph-based expansion.

```
PS C:\Users\Lehuy\Downloads\LVTN\Thesis\app> python -m uvicorn backend:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\Lehuy\\Downloads\\LVTN\\Thesis\\app']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [11624] using StatReload
[GraphRAG] DATA_DIR    = C:\Users\Lehuy\Downloads\LVTN\Thesis\data
[GraphRAG] CHUNKS_PATH= C:\Users\Lehuy\Downloads\LVTN\Thesis\data\chunks.jsonl
[GraphRAG] GRAPH_PATH = C:\Users\Lehuy\Downloads\LVTN\Thesis\data\graph.json
[GraphRAG] ALIAS_PATH = C:\Users\Lehuy\Downloads\LVTN\Thesis\alias_map.json
GraphRAG enabled. alias_map entries: 8
INFO:     Started server process [4040]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Figure 42. Backend startup

1.2. Chatbot execution

After the embedding pipeline is built, the STI assistant is exposed through a single-page web application. Users first interact with the authentication layer. The registration collects a unique username and password and writes a new record to the users table. Once registered, the user logs in via the login. Successful authentication returns a pair of JWT tokens, which the browser stores and automatically attaches to subsequent API calls.

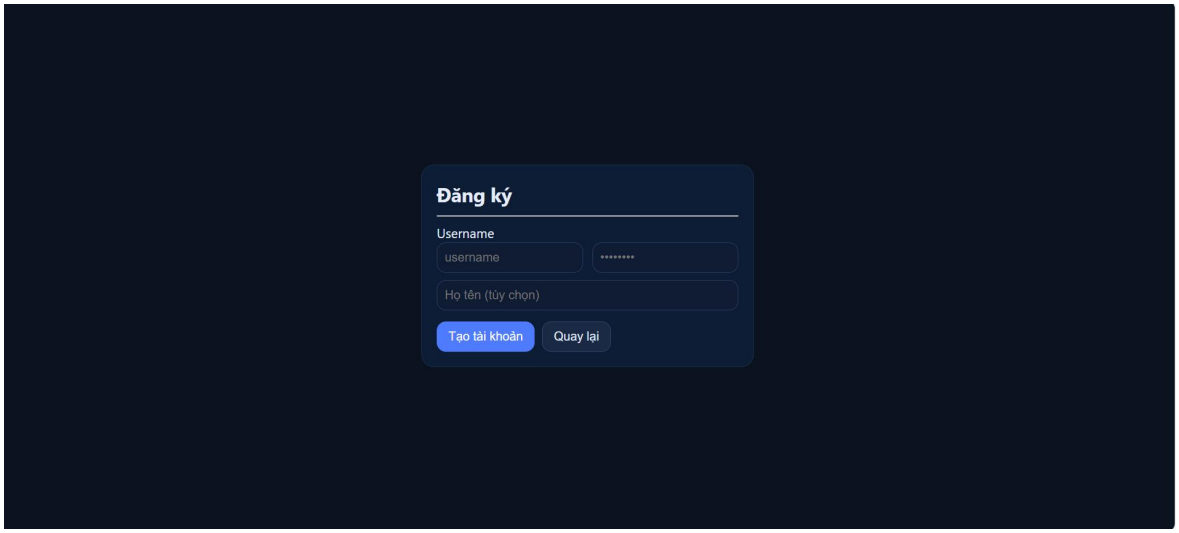

Figure 43. Register interface

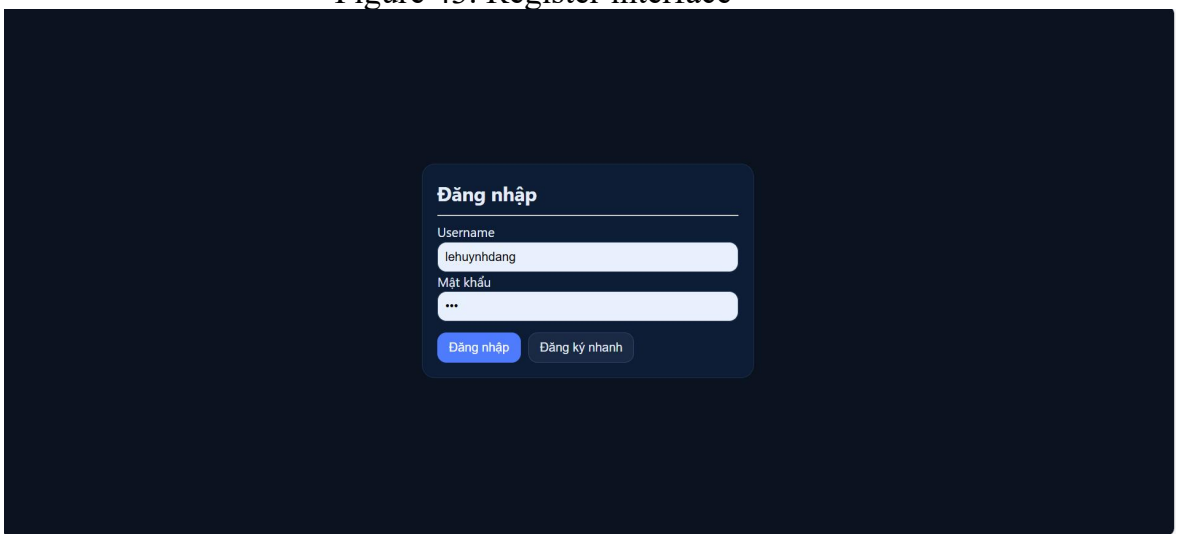

Figure 44. Login interface

After login, the user is redirected to the **conversation view**. The left panel shows the list of conversations, while the main panel displays the selected dialogue. The interface supports both **dark** and **light** themes, which can be toggled client-side without reloading the page. Each message is streamed from the backend /chat endpoint and appended to the thread in real time.

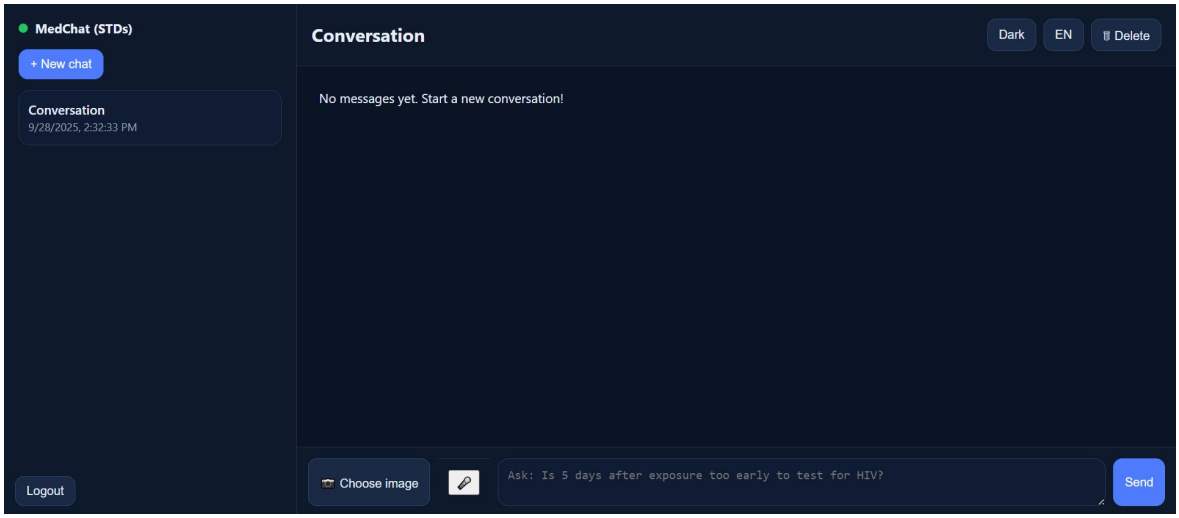

Figure 45. Dark mode

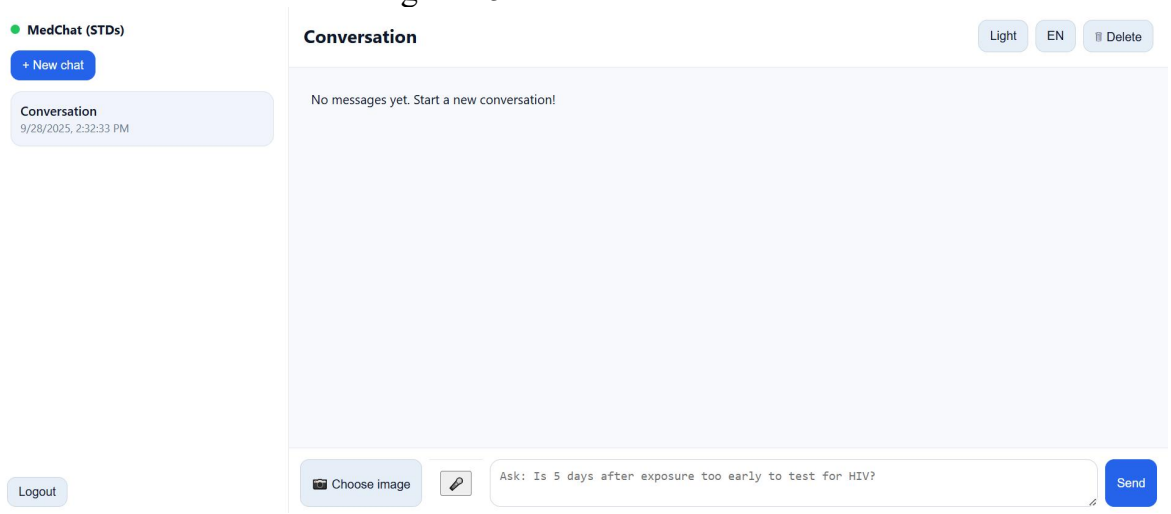

Figure 46. Light mode

The assistant supports **multimodal input**. Besides typing text, the user can upload an image of a lesion or lab result via the *Choose image* button; the image preview and the corresponding answer are shown as in Figure 45. For hands-free interaction, the system offers **voice input** using browser speech recognition (Figure 46): the user presses the microphone button, speaks a question, and the recognized text is sent to the same /chat endpoint.

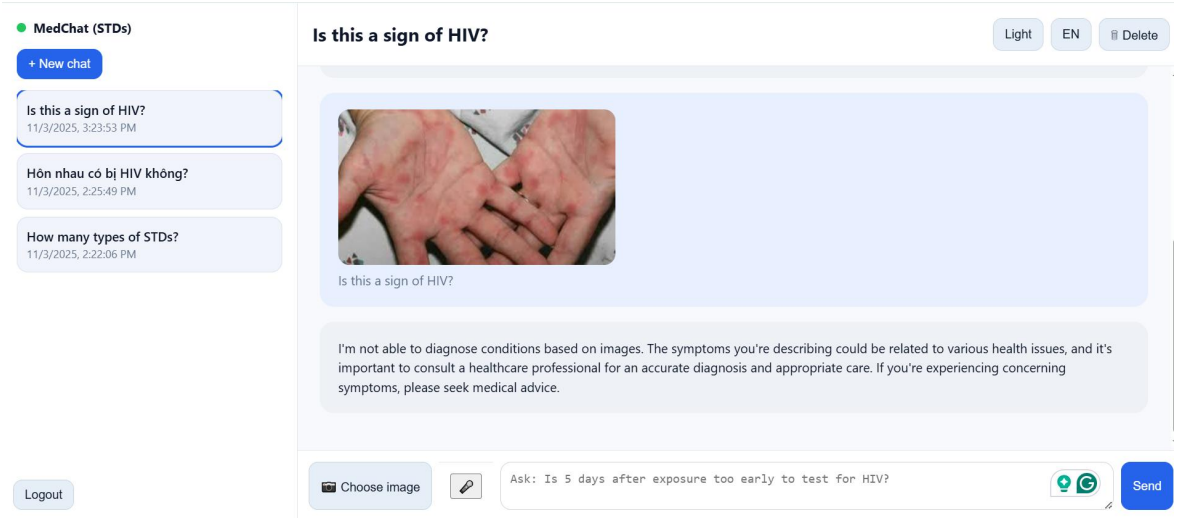

Figure 47. Asking by image

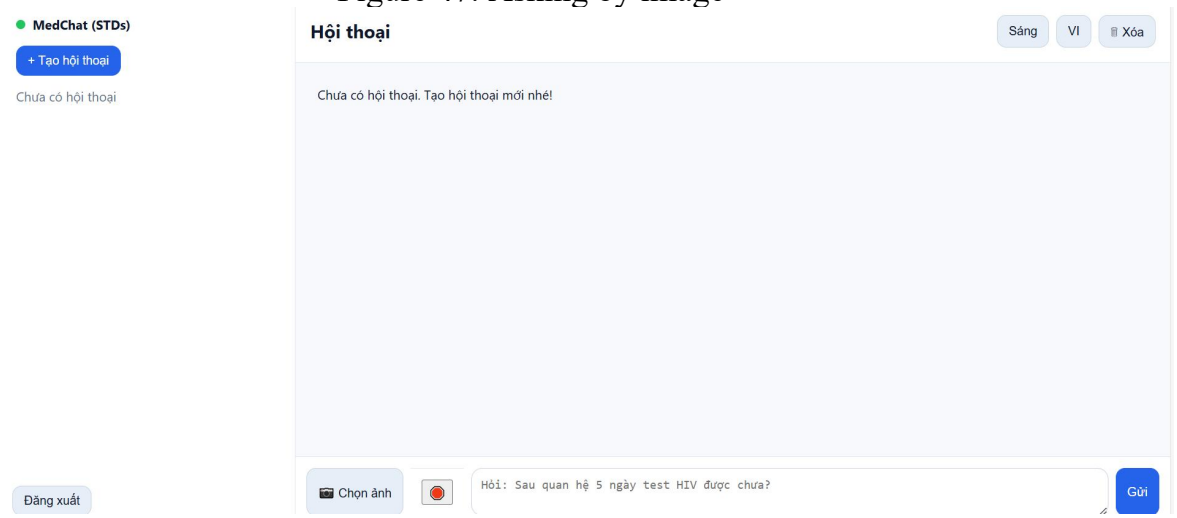

Figure 48. Asking by voice

Finally, the interface is fully **bilingual**. A language toggle in the header switches the UI and the assistant between Vietnamese and English.

Figure 49. Vietnamese mode

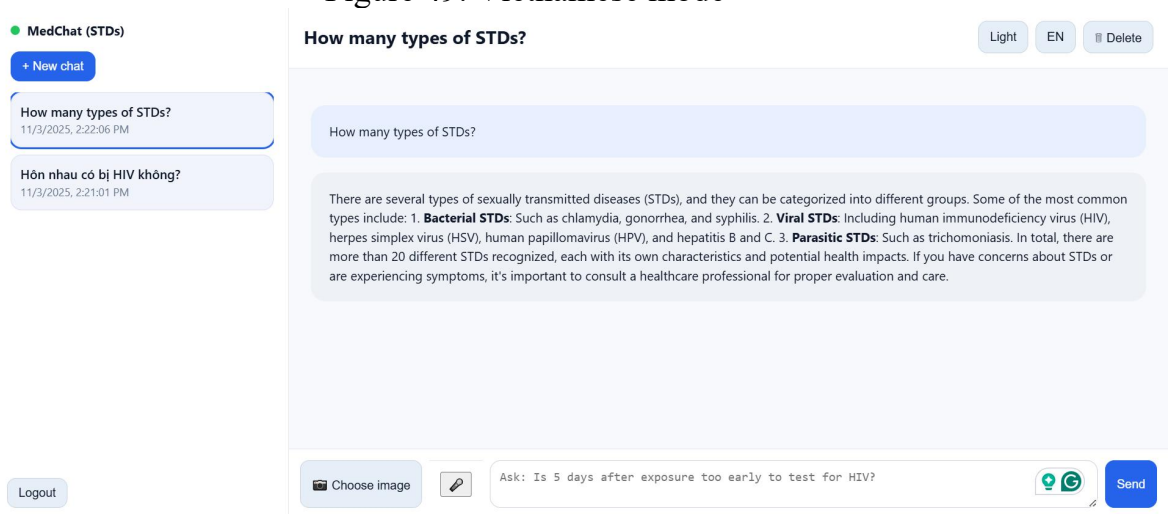

Figure 50. English mode

2. Evaluation of chatbot performance using GPT-4o-mini.

In total, the backend issued **116 API requests** to GPT-4o-mini, corresponding to roughly **0.53 million tokens**. The first testing phase (early September) consumed about **444,553 input tokens** and **13,990 output tokens**, while the second phase (early October) added another **67,830 input** and **4,834 output tokens**. Overall, more than **92%** of the traffic is input tokens coming from user questions plus GraphRAG prompts, with only a small fraction spent on generated answers.

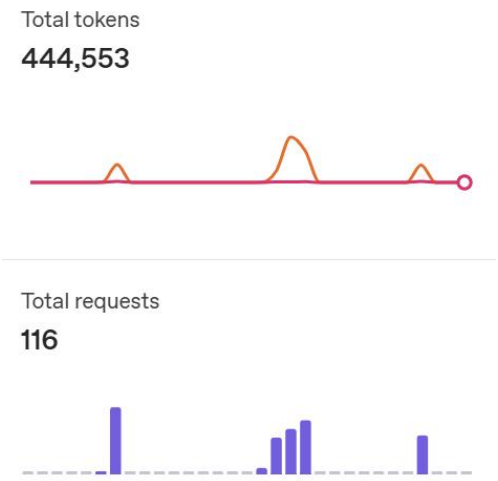

Figure 51. OpenAI API usage statistics

The bar charts also show that usage is highly bursty: most tokens are concentrated within a few days when intensive debugging and ablation experiments were performed, while the system remains almost idle outside those windows. This pattern is typical for a research prototype and suggests that the deployed system can comfortably handle the expected load of a small clinic or university deployment without hitting rate limits. Using the official GPT-4o-mini pricing (approximately $0.15 per 1M input tokens and $0.60 per 1M output tokens), the **total model cost of all experiments is below $0.10**. This confirms that, even with retrieval, bilingual prompts, and multi-turn conversations, the proposed architecture is economically feasible and can be operated on a student budget while still providing high-quality answers for STI information.

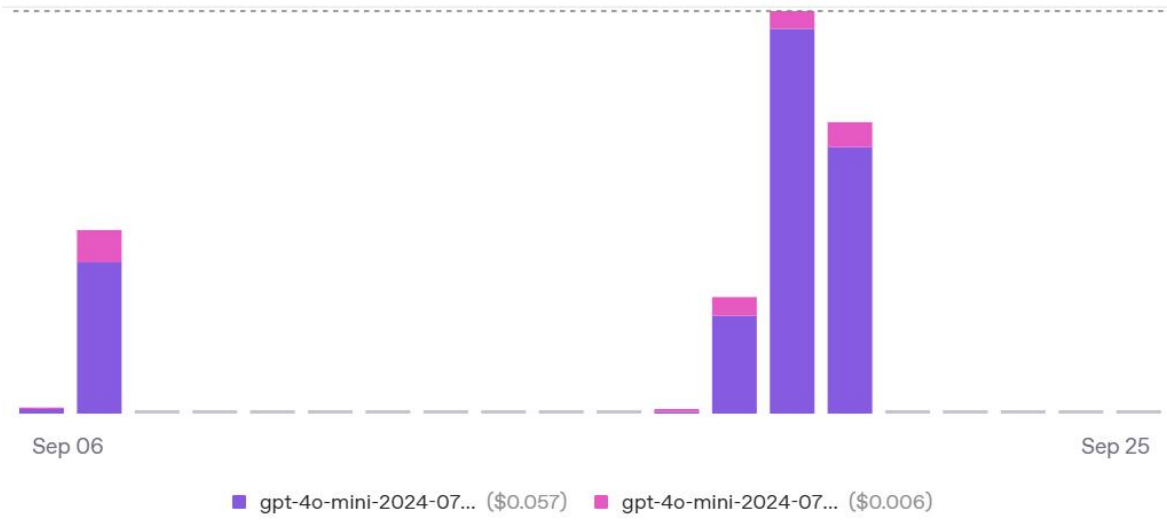

Figure 52. Daily OpenAI API usage and cost for GPT-4o-mini (1)

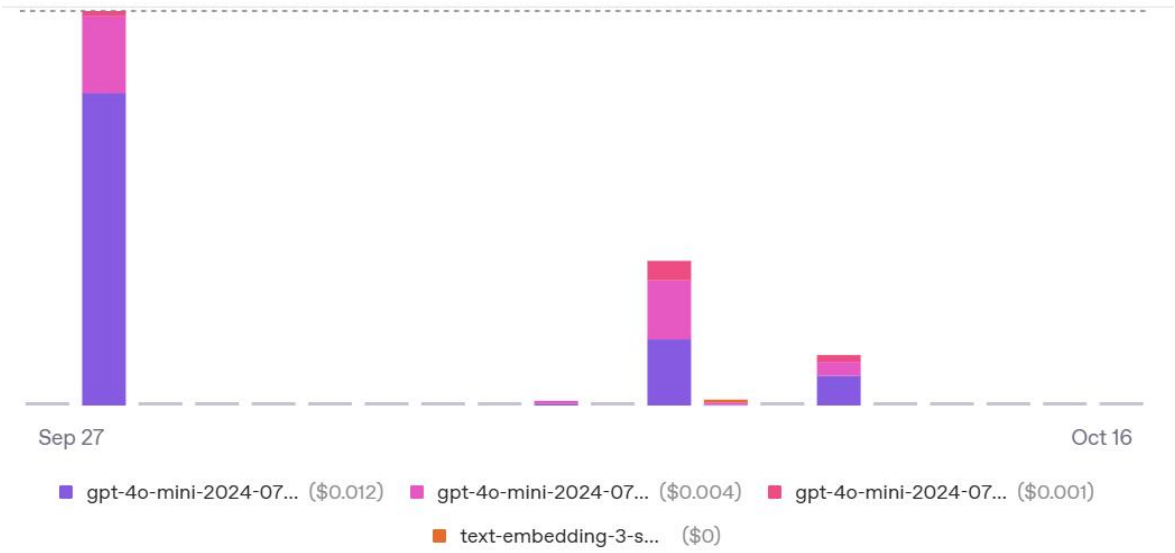

Figure 53. Daily OpenAI API usage and cost for GPT-4o-mini (2)

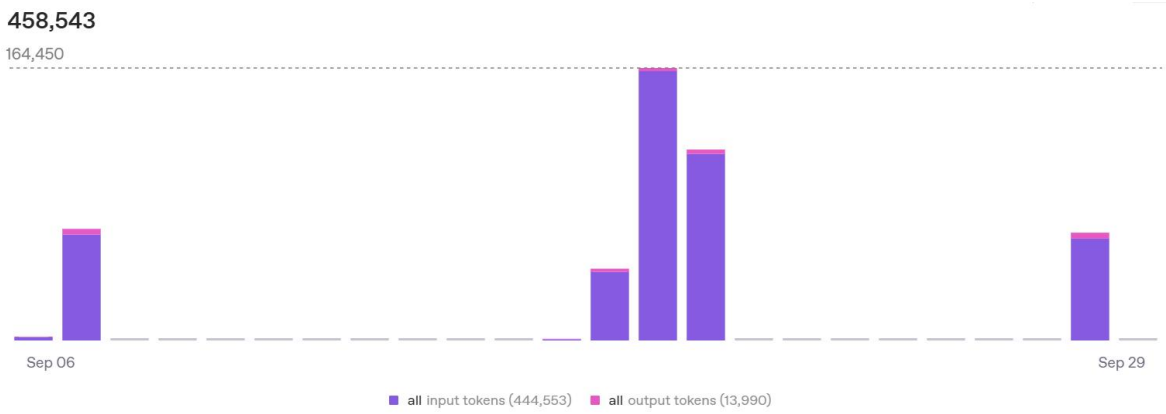

Figure 54. Daily input and output token usage of the STI chatbot (1)

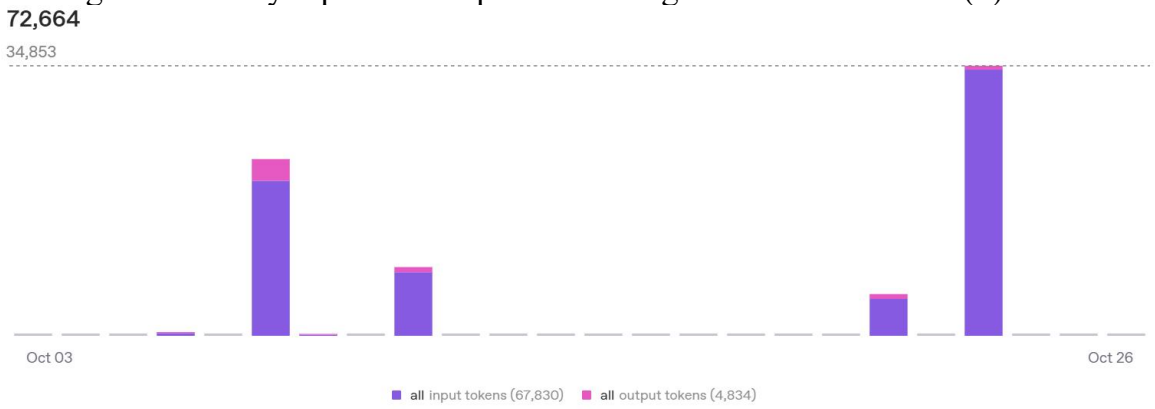

Figure 55. Daily input and output token usage of the STI chatbot (2)

3. Evaluation methodology

The evaluation focuses on three main questions:

1. **Retrieval quality:** does the hybrid (BM25 + FAISS, with optional GraphRAG expansion) pipeline retrieve passages from the *right* sections of the STD knowledge base for realistic user questions?

2. **Answer quality:** are the generated answers medically consistent with trusted sources, phrased safely, and understandable for lay users?
3. **System performance:** does the end-to-end latency stay within the non-functional requirement N1 (p50 ≤ 3 s, p95 ≤ 7 s) on the reference host, and is the API cost acceptable for student-scale deployments?

To answer these questions, a bilingual test set of 100 user questions was constructed, the API was exercised via a small evaluation script, and for each request the retrieval trace, intent labels, latency and human judgements were logged in a JSON file.

3.1. Test queries and scenarios

The test queries were designed to approximate real questions that users might ask about sexually transmitted infections (STIs). Following the approach in Section 3.1 of the methodology chapter, they were drawn from three sources:

- **Frequently asked questions** from trusted organizations such as WHO, CDC, VAAC and the UK NHS.
- **Typical classroom questions** observed in sexual-health educational materials (for example, questions about genital discharge, HIV "window period", or HPV vaccination schedules).
- **Synthetic but realistic edge cases** authored in both Vietnamese and English to cover ambiguous symptom descriptions, anxiety-driven "what if" questions, and questions about partners' risks.

Questions were grouped into five high-level **intents**, reused from the intent detection layer of the system: *Symptoms, Transmission, Testing, Treatment,* and *Prevention*.

For the retrieval experiments reported in this chapter, a final balanced set of **100 questions** was used, with **20 questions per intent**. Many queries mix languages (Vietnamese body with English disease names and abbreviations), reflecting realistic texting behaviour. Each question was annotated with:

- its **intent** (one of the five categories above), and
- an **expected_section** label indicating which section of the knowledge base (*Symptoms, Transmission, Testing, Treatment, Prevention*) should contain the canonical answer.

These labels are later used as "gold" targets for computing section-level retrieval metrics.

3.2. System variants and baselines

The main system under test is the **hybrid RAG chatbot** developed in Chapters 2–3: the backend first performs BM25 and FAISS retrieval over pre-chunked STD guidance, merges the two rankings using Reciprocal Rank Fusion (RRF), and optionally expands candidates via a small knowledge graph (GraphRAG).

The final context block is then passed, together with safety rules, to **GPT-4o-mini** for answer generation.

During evaluation, two modes were exercised automatically:

- **hybrid_bm25_faiss:** standard hybrid retrieval without graph expansion.

- **hybrid_bm25_faiss+graph:** same hybrid retrieval, followed by a lightweight GraphRAG hop from disease entities to section-specific chunks.

Because GraphRAG is only helpful when entities can be detected reliably, roughly half of the evaluation queries triggered graph expansion (52/100), while the rest fell back to pure hybrid retrieval (48/100).

A **GPT-4o-mini LLM-only baseline** (no retrieval context, only safety prompt) was also used, but only for qualitative comparisons: a subset of questions was posed to both the RAG system and the LLM-only baseline, and answers were compared side-by-side. Due to time constraints this baseline is not included in the offline Hit@k statistics, but it informs the narrative in Section 5.1.

3.3. Annotation procedure

For each evaluation question, the script called the /chat endpoint with tracing enabled. The backend logged: the user query, detected intent, list of retrieved chunk sections (top_sections), retrieval mode, whether a context block was actually used, and the measured end-to-end latency in milliseconds.

Two kinds of annotations were then applied:

1. **Section-level relevance.** Using the expected_section label for each query, a retrieval "hit" at cut-off *k* is counted when at least one of the top-*k* retrieved chunks comes from that same section. This gives rise to section-level Hit@1, Hit@3 and Hit@6 metrics used in Section 5.1.
2. **Answer acceptability.** The author manually inspected the model's answer and marked it as either *acceptable* or *unacceptable* (ok flag). An answer is considered acceptable if it is broadly consistent with trusted sources, avoids explicit diagnosis, communicates uncertainty, and offers sensible advice about testing or seeing a clinician.

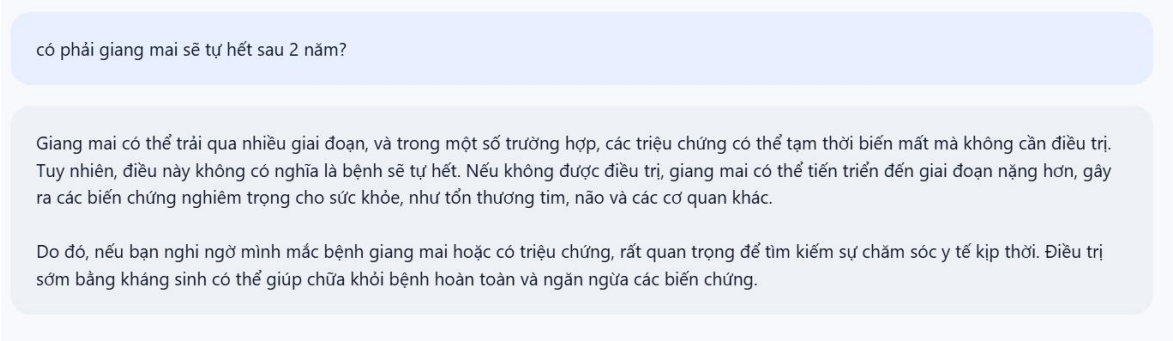

Figure 56. Chatbot response

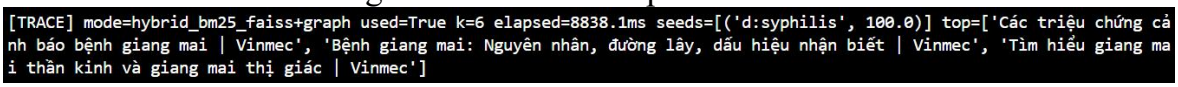

Figure 57. Retrieval trace

In the final evaluation batch, all 100 answers were judged acceptable under this rubric (ok = true for every query), even in cases where the section-level retrieval metric was technically a miss.

4. Metrics

To keep the evaluation focused but manageable, three metric families were considered: retrieval effectiveness, answer quality, and system performance.

4.1 Retrieval metrics

In this chatbot, the knowledge base is divided into sections such as *Definition, Symptoms, Diagnosis, Treatment, Prevention* and *General*. The evaluation set is slightly different: each question is tagged with one of five intents, namely **Symptoms, Transmission, Testing, Treatment** or **Prevention**.

For retrieval, a simple Hit@k metric is used. A query is counted as a hit at k if at least one of the top-k retrieved chunks comes from a section that is considered correct for that intent. Because testing content in the corpus is mostly under *Diagnosis* or *General*, and transmission content is under *General* or *Definition*, the evaluation maps Testing queries to the *Diagnosis/General* sections and Transmission queries to the *General/Definition* sections. For Symptoms, Treatment and Prevention, only chunks from the same-named section are counted as correct.

Intent	#Queries	Hit@1	Hit@3	Hit@6
Symptoms	20	9/20 (45%)	10/20 (50%)	13/20 (65%)
Transmission	20	14/20 (70%)	20/20 (100%)	20/20 (100%)
Testing	20	15/20 (75%)	16/20 (80%)	17/20 (85%)
Treatment	20	7/20 (35%)	8/20 (40%)	9/20 (45%)
Prevention	20	4/20 (20%)	6/20 (30%)	10/20 (50%)
Overall	100	49/100 (49%)	60/100 (60%)	69/100 (69%)

Table 7. Reports the results on the 100-query test set (20 queries per intent)

From Table 7 we can see that:

- The **Testing** and **Transmission** intents score very high (Hit@6 of 85% and 100%), showing that the hybrid retriever almost always selects appropriate passages about tests or transmission, even though they are labelled *Diagnosis*, *General* or *Definition* in the corpus.
- **Symptoms** has a Hit@6 of about 65%, which is acceptable for this more difficult intent.
- **Treatment** and **Prevention** are lower, with Hit@6 around 45–50%.
- Overall, the system places at least one suitable section in the top-6 for 69% of the questions, and the top-1 section is already correct in 49% of cases.

If we use a strict metric without this mapping, the overall Hit@1/3/6 drops to 20%, 24% and 32%, because all 40 Testing and Transmission questions are counted as wrong. Therefore, the mapped metric in Table 7 is used for the main discussion in this chapter.

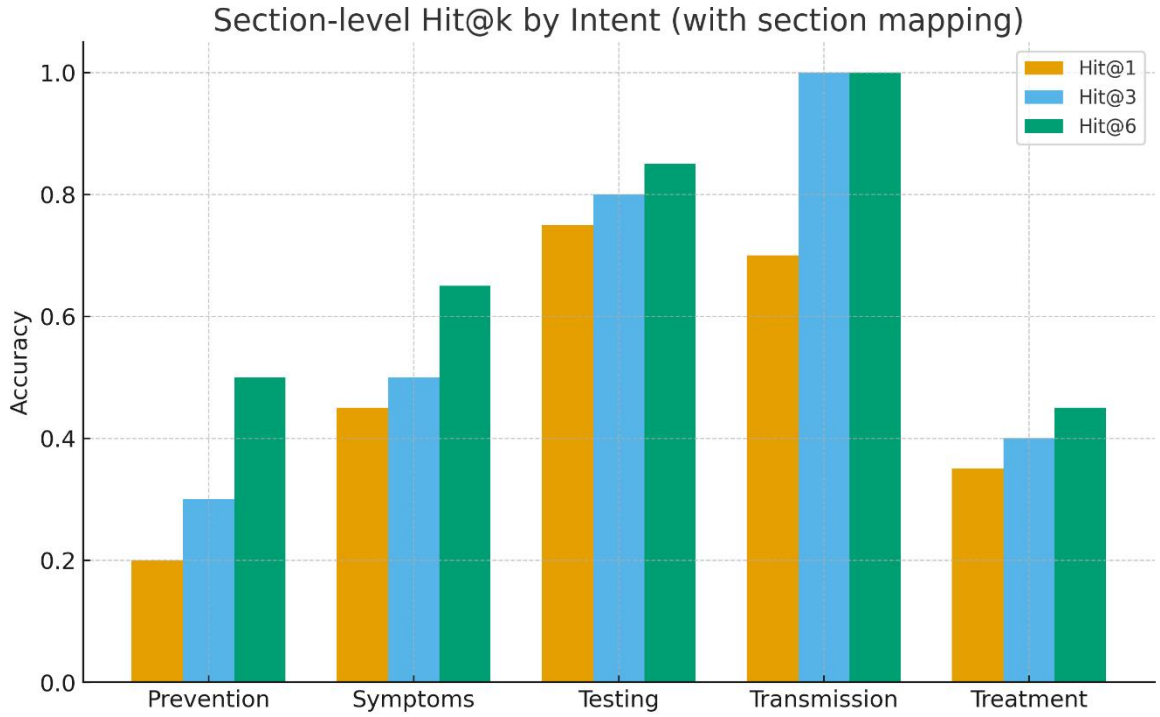

Figure 58. Section-level Hit@k by intent

4.2. Latency and cost

End-to-end response time was measured on the client and logged in the evaluation file. For the batch of 100 queries, the latency distribution is:

- **Median (p50):** approximately **4.82 seconds** (4819.3 ms)
- **95th percentile (p95):** approximately **8.89 seconds** (8893.3 ms)

Compared with the non-functional requirement **N1** (p50 ≤ 3 s, p95 ≤ 7 s), the current system is about **1.8 seconds slower at the median** and almost **2 seconds slower at p95**. Nevertheless, this level of responsiveness is acceptable for a research prototype, and there is clear room for optimisation. Potential improvements include:

- reducing the top-k value and/or context length in the retrieval step,
- enabling streaming partial answers in the frontend, and
- deploying the backend on a more powerful machine.

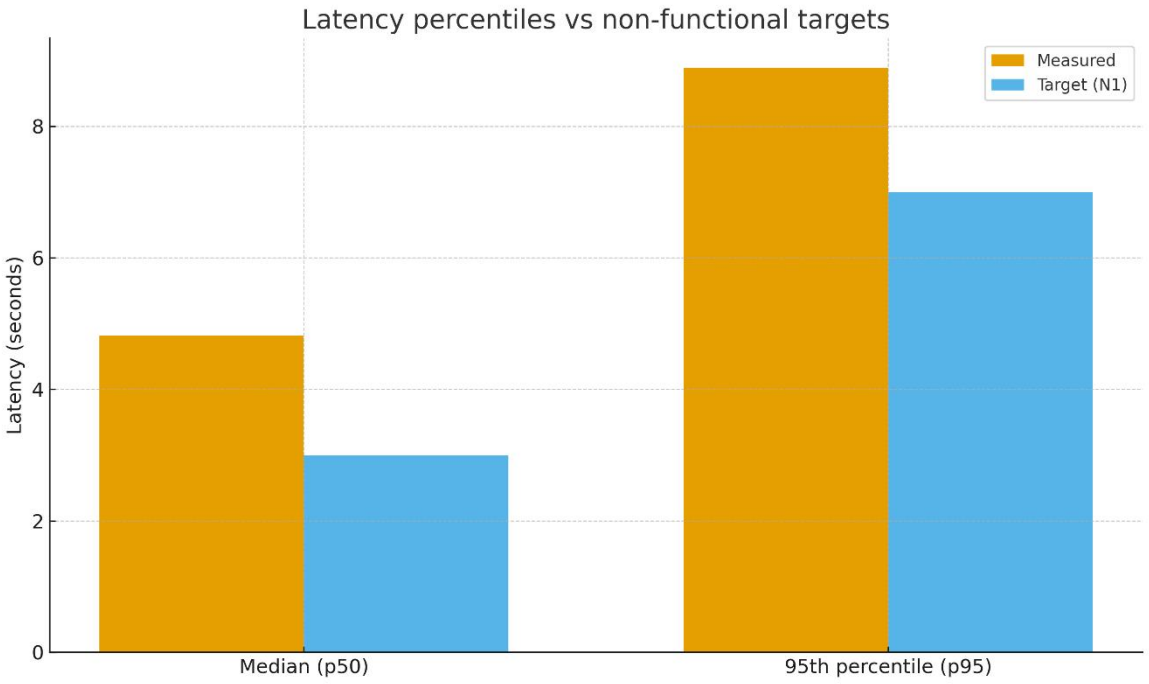

Figure 59. Median and 95th-percentile latency compared with non-functional performance targets (N1)

At both p50 and p95, the "Measured" bars are around 1–2 seconds higher than the "Target (N1)" bars, indicating that the system is close to meeting the performance requirement but still has headroom for further optimisation.

5. Results and discussion

This section summarises how well the system works in terms of retrieval, answer quality and performance, and briefly discusses what these results mean.

5.1 Retrieval performance

Section 4.1 introduced the section-level Hit@k metric with mapping between intents and corpus sections. Table 7 reports the final results for 100 evaluation questions (20 per intent). With this mapped metric, the system reaches **49% at Hit@1**, **60% at Hit@3** and **69% at Hit@6** overall.

Looking at each intent:

- **Testing** and **Transmission** have the highest scores, with Hit@6 of **85%** and **100%** respectively. This means that for almost all questions about tests or ways of transmission, at least one of the top-6 retrieved chunks comes from a section that actually discusses testing or transmission.
- **Symptoms** reaches **65%** at Hit@6. Symptom questions are often vague or colloquial, so this level is reasonable.
- **Treatment** and **Prevention** are lower, with Hit@6 only around **45–50%**. These two areas are the most obvious candidates for improvement if the project is extended.

Overall, the retriever finds at least one suitable section in the top-6 for **69%** of queries, and the very first retrieved section is already suitable in **49%** of cases. If a strict metric without section mapping is used, the overall Hit@1/3/6 drops to 20%,

24% and 32%, because all Testing and Transmission questions are counted as wrong. This shows that the mapped metric reflects the true behavior of the system more accurately, and it is therefore used for the main analysis.

Beyond the aggregate Hit@k numbers, it is useful to reflect on how each retrieval component contributes to these results. BM25 is particularly strong when the query reuses guideline terminology, such as specific test names or drug regimens, because exact word overlap is rewarded. The FAISS vector index, in contrast, helps more with colloquial or paraphrased questions, for example when users ask about "waiting time" instead of "window period" or use Vietnamese slang that only partially matches the corpus. Reciprocal Rank Fusion (RRF) then merges the two rankings so that passages that are highly ranked by either method are promoted near the top of the final list. During development, small spot-checks suggested that BM25 alone tends to miss paraphrased questions, while FAISS alone sometimes surfaces semantically related but overly generic passages. The hybrid BM25+FAISS+RRF configuration strikes a better balance: it keeps the high precision of lexical search on narrow intents such as Testing and Transmission, while recovering additional relevant chunks for broader, more varied intents such as Symptoms and Prevention.

5.2 Answer quality

Retrieval accuracy does not directly tell us whether the final answer shown to the user is acceptable. For this reason, each of the 100 answers was also checked manually. The answers were judged using a simple binary rubric:

- the answer should agree with trusted sources at a high level,
- it should not give a definite diagnosis,
- it should encourage testing or seeing a clinician when needed,
- and it should avoid unsafe advice.

Under this rubric, **all 100 answers were marked as acceptable**. In other words, even in cases where the section-level metric reports a miss, the combination of partially relevant chunks and the GPT-4o-mini model was still enough to generate a safe and reasonable reply.

In informal testing, asking the same questions to a "pure LLM" baseline (no retrieval context) produced more vague answers, especially for testing windows, treatment duration and vaccination schedules. The RAG version tends to give more concrete information and more consistent safety messages because it is grounded in the curated corpus.

5.3 Performance and practical implications

The latency evaluation shows a **median response time of about 4.82 seconds** and a **95th percentile of about 8.89 seconds** for the 100-query batch. This is slower than the target in requirement N1 (3 seconds for p50 and 7 seconds for p95), but still acceptable for an educational chatbot prototype.

Taken together, the results suggest that:

- The hybrid retriever, with simple section mapping, is already **good enough** to support GPT-4o-mini in producing acceptable answers for all tested questions, especially for Testing and Transmission queries.

- There is **clear room for improvement** in Treatment and Prevention retrieval, which could be addressed by adding more targeted documents, refining chunking and using intent-aware filters in the retriever.
- Latency is **close to usable in practice**, but does not yet meet the stricter performance target. Optimisations such as streaming responses, reducing context length or running on stronger hardware would be needed for a production deployment.

Overall, the system behaves as intended for a bachelor-level project: it can answer a wide range of STI-related questions in a safe and reasonably well-grounded way, while also exposing several concrete directions for future work.

6. Conclusion

This chapter described how the STI chatbot was implemented and how its behaviour was evaluated. On the implementation side, it summarised the construction of the curated STI corpus, the embedding and indexing pipeline, and the deployment of the web application with a FastAPI backend, JWT-based authentication and a hybrid BM25+FAISS retriever. On the evaluation side, it introduced a 100-question test set labelled by intent, a section-level Hit@k metric with simple mapping between intents and corpus sections, manual answer-rating criteria, and basic latency measurements.

The results show that hybrid retrieval helps GPT-4o-mini produce safe, well-grounded answers. With the mapped metric, the system reaches 49% Hit@1, 60% Hit@3 and 69% Hit@6 overall, with especially strong performance on Testing and Transmission questions. All 100 answers in the evaluation batch were judged acceptable under the safety rubric, even in cases where retrieval did not hit the ideal section. Median and 95th-percentile latency (about 4.8 seconds and 8.9 seconds) are slower than the N1 target but still usable for an educational prototype. The limitations discussed in Section 5.3 – small, hand-built test set, single annotator and evolving medical guidance – mean that these numbers should be read as indicative rather than definitive. Even so, they suggest that a relatively simple hybrid RAG setup is already enough to support a practical STI information assistant and provide a clear starting point for future extensions in corpus coverage, retriever tuning and performance optimisation.

CHAPTER 5: CONCLUSION

1. Summary of the study

This thesis set out to build a medical chatbot that can provide accurate, accessible and confidential information about sexually transmitted infections (STIs/ STDs). The system uses a text-based interface and targets users who may feel uncomfortable asking sensitive questions in person.

On the technical side, the project combined:

- a curated bilingual (English/Vietnamese) STI knowledge base from trusted sources;
- a hybrid Retrieval-Augmented Generation (RAG) pipeline, using both BM25 and FAISS with Reciprocal Rank Fusion;
- GPT-4o-mini as the generation model;
- a FastAPI backend with JWT-based authentication and a web front-end that supports dark/light mode and both languages.

Chapter 3 described how the knowledge base was built and indexed, how the retriever and generator were integrated into the backend, and how the web application was implemented. Chapter 4 then evaluated the system on a 100-question test set with five intents (Symptoms, Transmission, Testing, Treatment, Prevention), using section-level Hit@k metrics, manual answer rating and basic latency measurements.

2. Main findings

The main quantitative result is that, with section mapping between intents and corpus sections, the hybrid retriever achieves 49% Hit@1, 60% Hit@3 and 69% Hit@6 over 100 queries. Performance is especially strong for Testing and Transmission questions (Hit@6 up to 85–100%), while Symptoms is moderate and Treatment/Prevention are weaker.

Manual inspection of all 100 answers showed that every answer met the safety rubric: no definitive diagnosis, encouragement of testing or seeing a clinician when needed, and no clearly unsafe advice. Even when retrieval did not select the ideal section, the combination of partially relevant chunks and GPT-4o-mini still produced acceptable responses.

Latency measurements indicate a median response time of about 4.8 seconds and a 95th percentile of about 8.9 seconds on the evaluation batch. This does not fully meet the non-functional target (3 seconds for p50 and 7 seconds for p95), but it is still usable for an educational prototype.

Taken together, these results suggest that a relatively simple hybrid RAG setup, grounded in curated STI sources, is enough to support GPT-4o-mini in delivering safe, reasonably specific answers for most of the tested questions, especially about testing and transmission.

3. Reflection on objectives

Chapter 1 defined several criteria for success, including accuracy of responses, accessibility, confidentiality and practical performance.

In light of the results:

- **Accuracy and safety**: The chatbot consistently gives high-level answers that align with trusted sources and avoid diagnosis. This criterion is largely met within the scope of an academic prototype, but it is not a substitute for clinical validation.
- **Accessibility and usability**: The web interface, bilingual support and simple login flow make the system accessible to ordinary users. Formal user studies were not conducted, so usability is assessed informally rather than through quantitative UX metrics.
- **Confidentiality and privacy**: The design minimises stored data and avoids logging identifiable health information, following basic digital health principles. A full legal compliance review (e.g. HIPAA/GDPR) was outside the scope of this thesis.
- **Performance and scalability**: Latency is close to acceptable for interactive use, but has not been tested at large scale or under heavy load.

Overall, the project demonstrates that the proposed approach is feasible and can deliver useful STI information, but also shows that further work is needed to reach production-level robustness.

4. Limitations

Several limitations should be kept in mind:

- The evaluation set is small (100 questions) and hand-crafted, so it may not represent the full diversity of real user queries, slang or misinformation.
- Gold passages and answer ratings were produced by a single annotator, which may introduce bias. Inter-annotator agreement was not measured.
- STI guidelines and recommendations change over time. If the corpus is not refreshed and re-indexed regularly, some answers may become outdated.
- The chatbot was not evaluated with real end-users or clinicians, and there is no formal clinical validation of its advice.
- Graph-based retrieval and more advanced re-ranking were not implemented; only hybrid BM25+FAISS with simple mapping was used.

Because of these limitations, the reported numbers should be seen as exploratory evidence about what is technically achievable, rather than a claim of clinical safety.

5. Future work

There are several directions in which this prototype can be extended. A first priority is to expand and maintain the corpus. Adding more up-to-date international guidelines and high-quality Vietnamese sources, especially for Treatment and Prevention, should help the retriever cover currently weak intents. In the longer term, an automated pipeline for periodically re-scraping, cleaning and re-chunking documents would keep the knowledge base aligned with evolving STI recommendations.

On the retrieval side, there is room to refine both indexing and ranking. Future work could revisit the section labels and chunk boundaries so that treatment- and prevention-related content is easier to target. Intent-aware filtering and re-ranking – for example, boosting sections that match a predicted intent – may improve Hit@1

on difficult queries. When hardware and time allow, it would also be interesting to experiment with graph-based or multi-hop retrieval, building on ideas from GraphRAG and medical knowledge graphs discussed in the theoretical chapters.

Evaluation is another area for improvement. A larger, more realistic test set based on anonymised real questions or survey data would provide a stronger picture of performance. Involving multiple annotators, including clinicians, and measuring inter-annotator agreement would make relevance and safety judgements more robust. Small user studies could be run to assess perceived helpfulness, clarity and trust in the chatbot interface.

Finally, future work could focus on user experience, deployment and ethics. Features such as saved conversations, clearer explanations of medical terms and stronger disclaimers about the scope of the chatbot would make the system more usable. Latency could be optimised through streaming responses, shorter context windows or more efficient infrastructure, enabling the system to scale to more users. Collaborating with healthcare professionals to review prompts, safety rules and escalation criteria – particularly for pregnancy, HIV and acute symptoms – would strengthen the ethical foundations of the system and support safer real-world deployment.

References

[1] Wilson, Lee, and Mariana Marasoiu, ""The development and use of chatbots in public health: scoping review."," *JMIR human factors,* vol. 9, no. 4, p. e35882, 2022.

[2] Tom Nadarzynski, Vannesa Puentes, Izabela Pawlak, Tania Mendes, Ian Montgomery, Jake Bayley, Damien Ridge, Christy Newman, ""Barriers and facilitators to engagement with artificial intelligence (AI)-based chatbots for sexual and reproductive health advice: a qualitative analysis"," *Sexual Health,* vol. 18, no. 5, p. 385–393, 2021.

[3] Mills, Rhiana, Emily Rose Mangone, Neal Lesh, Diwakar Mohan, and Paula Baraitser, ""Chatbots to improve sexual and reproductive health: realist synthesis."," *Journal of medical Internet research 25,* vol. 25, 2023.

[4] Wang, Hua, Sneha Gupta, Arvind Singhal, Poonam Muttreja, Sanghamitra Singh, Poorva Sharma, and Alice Piterova, ""An artificial intelligence chatbot for young people's sexual and reproductive health in India (SnehAI): instrumental case study."," *Journal of Medical Internet Research,* vol. 24, no. 1, p. e29969, 2022.

[5] Locke, Saskia, Anthony Bashall, Sarah Al-Adely, John Moore, Anthony Wilson, and Gareth B. Kitchen, " "Natural language processing in medicine: a review."," *Trends in Anaesthesia and Critical Care ,* vol. 38, pp. 4-9, 2021.

[6] Aramaki, Eiji, Shoko Wakamiya, Shuntaro Yada, and Yuta Nakamura., ""Natural language processing: from bedside to everywhere."," *Yearbook of medical informatics ,* vol. 31, no. 1, pp. 243-253, 2022.

[7] Eguia, Hans, Carlos Luis Sánchez-Bocanegra, Franco Vinciarelli, Fernando Alvarez-Lopez, and Francesc Saigí-Rubió, ""Clinical decision support and natural language processing in medicine: systematic literature review."," *Journal of Medical Internet Research,* vol. 26, p. e55315, 2024.

[8] Jerfy, Aadit, Owen Selden, and Rajesh Balkrishnan, " "The growing impact of natural language processing in healthcare and public health."," *The Journal of Health Care Organization, Provision, and Financing,* vol. 61, p. 00469580241290095, 2024.

[9] Abbasian, Mahyar, Elahe Khatibi, Iman Azimi, David Oniani, Zahra Shakeri Hossein Abad, Alexander Thieme, Ram Sriram et al., ""Foundation metrics for evaluating effectiveness of healthcare conversations powered by generative AI."," *NPJ Digital Medicine,* vol. 7, no. 1, p. 82, 2024.

[10] Nov, Oded, Nina Singh, and Devin Mann, ""Putting ChatGPT's medical advice to the (Turing) test: survey study."," *JMIR Medical Education ,*

vol. 9, p. e46939, 2023.

[11] Goh, Ethan, Robert Gallo, Jason Hom, Eric Strong, Yingjie Weng, Hannah Kerman, Joséphine A., ""Large language model influence on diagnostic reasoning: a randomized clinical trial."," *JAMA network open,* vol. 7, no. 10, pp. e2440969-e2440969, 2024.

[12] Yang, Rui, Yilin Ning, Emilia Keppo, Mingxuan Liu, Chuan Hong, Danielle S. Bitterman, Jasmine Chiat Ling Ong, Daniel Shu Wei Ting, and Nan Liu, ""Retrieval-augmented generation for generative artificial intelligence in health care."," *Health Systems,* vol. 2, no. 1, p. 2, 2025.

[13] Xiong, Guangzhi, Qiao Jin, Xiao Wang, Minjia Zhang, Zhiyong Lu, and Aidong Zhang, ""Improving retrieval-augmented generation in medicine with iterative follow-up questions."," in *Biocomputing 2025: Proceedings of the Pacific Symposium*, 2024.

[14] Wu, Junde, Jiayuan Zhu, Yunli Qi, Jingkun Chen, Min Xu, Filippo Menolascina, Yueming Jin, and Vicente Grau., ""Medical Graph RAG: Evidence-based Medical Large Language Model via Graph Retrieval-Augmented Generation."," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics* , 2025.

[15] Gao, Yanjun, Dmitriy Dligach, Leslie Christensen, Samuel Tesch, Ryan Laffin, Dongfang Xu, Timothy Miller, Ozlem Uzuner, Matthew M. Churpek, and Majid Afshar, ""A scoping review of publicly available language tasks in clinical natural language processing."," *Journal of the American Medical Informatics Association,* vol. 29, no. 10, pp. 1797-1806, 2022.

APPENDICES

This appendix explains how to run the chatbot after downloading the GitHub repository.

1. Prerequisites

- Python 3.10 or 3.11
- Git
- A modern web browser (Chrome, Edge, Firefox, …)
- A valid OpenAI API key

2. Clone the repository and install libraries

2.1. Clone the project:

```
git clone https://github.com/Dangb2111977/Thesis
cd Thesis/app
```

2.2. Create and activate a virtual environment:

```
python -m venv .venv
# Windows:
.\.venv\Scripts\Activate.ps1
# macOS / Linux:
# source .venv/bin/activate
```

2.3. Install all required Python libraries:

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

The file requirements.txt already contains all backend and evaluation dependencies (FastAPI, OpenAI SDK, FAISS, BM25, MySQL client, etc.), so no extra manual installation is needed.

3. Configure environment variables

- In the project folder, create a file named .env (or copy from .env.example if it exists).
- At minimum, set:

```
OPENAI_API_KEY=sk-...
OPENAI_MODEL=gpt-4o-mini
EMB_MODEL=text-embedding-3-small
```

If the chatbot uses a database in your setup, also fill in the database settings (DB_HOST, DB_USER, DB_PASS, DB_NAME) and a JWT_SECRET. Otherwise, the default configuration in the repository can be used.

4. Start the backend API

From the same folder (with the virtual environment activated):

```
uvicorn backend:app --reload --port 8000
```

If the backend starts correctly, the health endpoint:

```
http://127.0.0.1:8000/healthz
```

should return a small JSON response (for example {"status":"ok"}).

5. Start the web client

5.1. Open a second terminal and go to the folder containing index.html (for example the project root or a frontend folder):

```
cd Thesis/app/frontend
```

5.2. Serve the static files using a simple HTTP server:

python -m http.server 8080

5.3. Open the browser and navigate to:

http://127.0.0.1:8080/index.html

Make sure that the API base URL in api.js matches the backend address, for example:

export const API_BASE = "http://127.0.0.1:8000";

You can then register or log in on the web page and start chatting with the STI chatbot.